INFORUM

Interindustry Forecasting at the University of Maryland

Inforum Help Documentation

Release 1.09

Inforum

Jan 17, 2024

CONTENTS

1 An Introduction to Inforum 3
- 1.1 An Introduction to *G7* 3
- 1.2 *G7*: New for 2024 4
- 1.3 *G7*: New for 2023 4
- 1.4 *G7*: New for 2022 5
- 1.5 *G7*: New for 2021 6
- 1.6 *G7*: New for 2020 6
- 1.7 *G7*: New for 2019 7
- 1.8 *G7*: New for 2018 7
- 1.9 *G7*: New for 2017 7
- 1.10 *G7*: New for 2016 8
- 1.11 *G7*: New for 2015 10
- 1.12 *G7*: New for 2014 12
- 1.13 *G7*: New for 2013 13
- 1.14 *G7*: New for 2012 15
- 1.15 *G7*: New for 2011 17
- 1.16 *G7*: New for 2010 18
- 1.17 *G7*: New for 2009 19
- 1.18 *G7*: New for 2008 20
- 1.19 *G7*: New for 2007 20

2 *G7* Users' Guide 21
- 2.1 Getting Started 21
  - 2.1.1 What is *G7*? 21
  - 2.1.2 Installation 22
  - 2.1.3 Starting *G7* 24
  - 2.1.4 Looking at a Series in a Data Bank 26
  - 2.1.5 The *G7* Editor 27
  - 2.1.6 The *G7* Command Reference 29
  - 2.1.7 The 11 Basic *G7* Commands 30
  - 2.1.8 Conventions 33
  - 2.1.9 Basic Regression 33
  - 2.1.10 Command Files and the *G7* Editor 34
  - 2.1.11 Saving Results 35
  - 2.1.12 Setting Graph Styles 36
  - 2.1.13 Command Files Within Command Files 38
  - 2.1.14 Setting the Properties of the Results Area 38
- 2.2 Data Banks 39
  - 2.2.1 Assigned Data Banks 39
  - 2.2.2 The Workspace Bank: Making, Compressing, and Splicing Banks 42

2.3 *G7* Commands and Scripts . . . 43
2.3.1 Dates and Frequencies . . . 43
2.3.2 Forming Variables . . . 45
2.3.3 G7 Functions . . . 47
2.3.4 Command Files, Groups, and Do Loops . . . 51
2.3.5 Flow Control . . . 57
2.3.6 User-specified Error Handling . . . 60
2.3.7 Strings in *G7* . . . 61
2.3.8 Variables and Arguments in *G7* . . . 63
2.3.9 Reading Data Into *G7* . . . 68
2.3.10 Writing ASCII Data From *G7* . . . 71
2.3.11 How To Read and Write in Excel Format with *G7* . . . 73
2.3.12 Examples For Using the XL Commands . . . 87
2.3.13 Writing Data To Lotus WK1 . . . 89
2.3.14 Making Tables . . . 90
2.3.15 Drawing Graphs . . . 91
2.3.16 Miscellaneous Commands and Other Information . . . 96
2.4 Regression in *G7* . . . 99
2.4.1 Ordinary Regression . . . 99
2.4.2 Soft Constraints . . . 104
2.4.3 Distributed Lags . . . 105
2.4.4 Regression Tests . . . 105
2.4.5 Hildreth-Lu Technique for Autocorrelation Correction . . . 106
2.4.6 ARIMA Techniques . . . 106
2.4.7 Seemingly Unrelated Regressions . . . 107
2.4.8 Common-Coefficient, Panel-Data, or Pooled Regressions . . . 107
2.4.9 Non-linear Regression . . . 108
2.4.10 2SLS and 3SLS . . . 110
2.5 Modeling with *G7* . . . 111
2.5.1 Building a Model . . . 111
2.5.2 Running a Model - Deterministic Simulation . . . 114
2.5.3 Optimizing a Model . . . 116
2.5.4 Stochastic Simulation . . . 118
2.5.5 Graphing Model Results . . . 118
2.5.6 Tabulating Model Results or Data . . . 119
2.5.7 Model Building . . . 122
2.5.8 Running an *InterDyme* Model in *G7* . . . 124
2.6 Vectors And Matrices . . . 126
2.6.1 Working with Vam Files in *G7* . . . 126
2.6.2 The Vam Configuration File . . . 126
2.6.3 Creating, Assigning, Setting Defaults, and Closing a Vam File . . . 128
2.6.4 Input of Time Series into Vam Files . . . 129
2.6.5 Input of Vectors into Vam Files . . . 130
2.6.6 Input of Matrices to Vam Files . . . 133
2.6.7 Display of Vam File Data . . . 135
2.6.8 Vector Calculations . . . 136
2.6.9 Matrix Calculations in VAM files . . . 138
2.6.10 Groups of Sector Numbers . . . 140
2.6.11 The Resector Command . . . 141
2.6.12 Linear Interpolation of Vectors and Matrices . . . 143
2.6.13 Moving Vectors and Matrices by Indexing . . . 144
2.6.14 Methods of Scaling . . . 145
2.6.15 The Scale Command . . . 147
2.6.16 The Ctrl Command . . . 148

2.6.17 Methods of Balancing Matrices . . . 148
2.6.18 Matrix Balancing by the Ordinary RAS Method . . . 149
2.6.19 Balancing Matrices with Uncertain Totals . . . 150
2.6.20 Balancing Entire Input-Output Tables . . . 150
2.6.21 Balancing Matrices with Externally Imposed Preconditions . . . 151
2.6.22 The Coef, Flow, Getsum, Load, and Store Commands . . . 153
2.6.23 Writing Vam File Data to ASCII Files . . . 153
2.6.24 Titles for Vam Files . . . 154
2.6.25 Regressions on Industry Data . . . 154
2.6.26 *Vam2Vam* – Selective Copying From One Vam File to Another . . . 157
2.6.27 *VamToG* – Creating a *G7* bank From Series in a Vam File . . . 158
2.7 Other *G7* Documentation . . . 158

3 *G7* Reference Manual 159
3.1 *G7* Commands: Symbols . . . 159
3.2 *G7* Commands: A . . . 159
3.3 *G7* Commands: B . . . 162
3.4 *G7* Commands: C . . . 163
3.5 *G7* Commands: D . . . 166
3.6 *G7* Commands: E . . . 170
3.7 *G7* Commands: F . . . 171
3.8 *G7* Commands: G . . . 176
3.9 *G7* Commands: H . . . 179
3.10 *G7* Commands: I . . . 181
3.11 *G7* Commands: L . . . 186
3.12 *G7* Commands: M . . . 192
3.13 *G7* Commands: N . . . 199
3.14 *G7* Commands: P . . . 201
3.15 *G7* Commands: Q . . . 204
3.16 *G7* Commands: R . . . 205
3.17 *G7* Commands: S . . . 211
3.18 *G7* Commands: T . . . 218
3.19 *G7* Commands: U . . . 221
3.20 *G7* Commands: V . . . 221
3.21 *G7* Commands: W . . . 228
3.22 *G7* Commands: X . . . 229
3.23 *G7* Commands: Y . . . 243
3.24 *G7* Commands: Z . . . 244

4 *G7* Tutorial 245
4.1 *G7* Tutorial . . . 245
4.2 The *G7* Main Window and Some Basic *G7* Commands . . . 246
4.3 The *G7* Command Box and Button Bar . . . 250
4.4 More on the *f* Command . . . 251
4.5 Running *G7* Commands in Batch (Add files) . . . 251
4.6 Using the *G7* Editor . . . 252
4.7 Using *Compare* to Make Tables . . . 253
4.8 Where to Go to Learn More . . . 257

5 *Compare* Manual 259
5.1 Introduction to *Compare* . . . 259
5.2 Preparing the *Compare* Stub File . . . 259
5.3 Choosing a File Type . . . 260
5.4 The Most Commonly Used *Compare* Commands . . . 262

5.5 *Compare* Commands That Control The Table Format . . . 264
5.6 Commands That Control The Printer . . . 266
5.7 Commands to Build Tables in Excel Format . . . 275
5.8 Calculation of Data Within *Compare* . . . 281
5.9 Calculation and Display of Running Totals . . . 282
5.10 Commands for Sorting or Ranking . . . 282
5.11 Miscellaneous Commands . . . 283
5.12 Groups in *Compare* . . . 284
5.13 Commands Used With Vam Files . . . 285
5.14 Variables, Expressions, and Functions . . . 286
5.15 Running the *Compare* Program . . . 287
5.16 Special Uses with *Compare* . . . 288

6 *Build* Macro-Equation Processor 291

7 *Banker* : The G-Bank Maker 293
7.1 What is Banker? . . . 293
7.2 Usage . . . 293
7.3 How to Format Input Data for Banker . . . 294
7.4 Notes for Programmers . . . 295

8 *Fixer* and *MacFixer* Programs 301
8.1 *MacFixer*: Fixing Macro Variables in *InterDyme* Models . . . 301
8.2 *Fixer*: Fixing General Vectors and Matrices in *InterDyme* Models . . . 306

9 *IdBuild* Macro-Equation Processor for InterDyme 311

Index 317

This Help system provides documentation for Inforum software, including *G7*, *Compare*, *MacFixer*, *Fixer*, and *IdBuild*. Please visit the web site for the latest software and documentation updates. Please report any problems with the software, and please send questions and comments by email to the Inforum webmaster.

Contents:

CHAPTER

ONE

AN INTRODUCTION TO INFORUM

Inforum was founded more than 50 years ago by Dr. Clopper Almon, now Professor Emeritus of the University of Maryland. It is dedicated to improving business planning, government policy analysis, and the general understanding of the economic environment. Inforum accomplishes this mission through:

- Building and using structural economic models of U.S. and other economies. Inforum pioneered the construction of dynamic interindustry-macroeconomic models that portray the economy in a unique "bottom-up" fashion.
- Working with government and private sector research sponsors to investigate a variety of issues. Economic projections and analysis using Inforum econometric models are distinguished by detail at the industrial and product level.
- Serving as a training crucible for graduate and undergraduate students who receive valuable training in empirical economics. Indeed, Inforum graduate research assistants have completed over 40 Ph.D. dissertations, most of which have contributed directly to the infrastructure of Inforum.
- Maintaining active and productive ties with a world-wide network of research associates, each of which uses Inforum modeling methods and software. The Inforum partners have held annual conferences since 1993 to foster cooperation and development of economic knowledge and techniques.

Since its founding in 1967 by Dr. Clopper Almon, Inforum has served government agencies and private sector entities interested in economic analysis facilitated by Inforum's approach. In particular, Interindustry-Macroeconomic (IM) models combine input-output structure with econometric equations in a dynamic and detailed framework. Because of their ability to portray the detailed structure of economies over actual time periods, these models fill an important gap in the inventory of existing models of the U.S. and foreign economies.

1.1 An Introduction to *G7*

G7 is a data base, regression, and econometric model-building program for Windows. It is designed to estimate regression equations with annual, quarterly, or monthly time-series data. *G7* takes its name from Carl Friedrich Gauss, the originator of the method of least squares. The *G7* project is maintained at the University of Maryland, but is supported by contributions of Inforum colleagues around the world. Please visit the Inforum web site for more information and the latest version of the *G7* software. The latest edition of these help files and additional documentation also may be found there. Please report any problems with the software, and please send questions and comments by email to the Inforum webmaster.

Serveral sources of information are available for *G7*. These include the *G7* Help Files that include the *G7* User Guide and Reference Manual, the *G7* Tutorial, and the Software Demonstrations routines that are available from the Inforum web site.

1.2 *G7*: New for 2024

Work in 2024 provided new features, improvements to existing features, and a variety of bug fixes. The most important improvements and new capabilities provided in the *G7* software are these:

- Substantial effort began in 2021 to shift development of *G7* from Borland Builder 6 (a 1990s-vintage IDE) to Embarcadero RAD Studio. Early in 2023, development again shifted to the newer C++ Builder 11. Most fatures now are working in the new environment, though additional testing of the new edition is needed before general release. G7.3x indicates older versions compiled under Builder 6, and G7.41 and later versions are compiled with C++ Builder.
- Improved Documentation: The *G7* documentation was revised heavily in 2012 and has been updated again in 2023, with new Help files, Users' Guide, Reference Manual, Tutorial, and other documentation. Companion documents in HTML and PDF formats also were updated. In particular, recent software improvements and extensions have been documented, and cross reference hyperlinks have been added to the Reference Manual.
- Miscellaneous Improvements:

  a) *Forthcoming…*
- An Abridged Listing of Bug Fixes:

  a) Fixed several problems affecting the *vmake routine* that caused 1) date misalignment between source and destination objects and 2) failure with vector-to-vector operations.

The version of G7 in January 2024 was 7.415.

1.3 *G7*: New for 2023

Work in 2023 provided new features, improvements to existing features, and a variety of bug fixes. The most important improvements and new capabilities provided in the *G7* software are these:

- Substantial effort began in 2021 to shift development of *G7* from Borland Builder 6 (a 1990s-vintage IDE) to Embarcadero RAD Studio. Early in 2023, development again shifted to the newer C++ Builder 11. Most fatures now are working in the new environment, though additional testing of the new edition is needed before general release. G7.3x indicates older versions compiled under Builder 6, and G7.41 and later versions are compiled with C++ Builder.
- Improved Documentation: The *G7* documentation was revised heavily in 2012 and has been updated again in 2023, with new Help files, Users' Guide, Reference Manual, Tutorial, and other documentation. Companion documents in HTML and PDF formats also were updated. In particular, recent software improvements and extensions have been documented, and cross reference hyperlinks have been added to the Reference Manual.
- Miscellaneous Improvements:

  a) Improved the *type routine* to improve formatting and to avoid data overlap when the *width* setting is too low (see *format*). Forced stricter consistency of *tdates setting* and the *frequency* of the series to be printed. Allow longer functions (<arguments>) to be evaluated as *type (<argument>)*.
- An Abridged Listing of Bug Fixes:

a) Fixed subtle problems affecting the *ctrl routine* and other features; *G7* now should be more stable.

b) Corrected obscure problems with *str routines* that particularly afflicted *%eliteral*, *str replace*, and related routines.

c) Corrected problems with e.g. "r slmedrdi = ((gpop1+gpop2)/pt)" where "/pt" was not printed to file for reading by IdBuild.

d) Corrected bad checks with vector sizes and group specifications in ras.

e) Prevent buffer overrun in when bank series length exceeds the workspace size.

The version of G7 in January 2023 was 7.410.

1.4 *G7*: New for 2022

Work in 2022 provided new features, improvements to existing features, and a variety of bug fixes. The most important improvements and new capabilities provided in the *G7* software are these:

- Substantial effort began in 2021 to shift development of G7 from Borland Builder 6 (a 1990s-vintage IDE) to Embarcadero RAD Studio. Most features now are working in the new environment, and the new edition nearly is ready for general release. G7.3x indicates older versions compiled under Builder 6, and G7.4x versions are compiled with RAD Studio.
- Improved Documentation: The *G7* documentation was revised heavily in 2012 and has been updated again in 2022, with new Help files, Users' Guide, Reference Manual, Tutorial, and other documentation. Companion documents in HTML and PDF formats also were updated. In particular, recent software improvements and extensions have been documented, and cross reference hyperlinks have been added to the Reference Manual.
- Miscellaneous Improvements:

  a) Suppressed *data command* output to screen when the *addprint* setting is 'n' to speed processing.

- An Abridged Listing of Bug Fixes:

  a) Fixed problems in the *@lint routine*.

  b) Fixed error message reporting failure when Help file not found upon startup.

  c) Fixed problems in the *@qchwt routine*.

  d) Add missing fdate specification in graph routine for calculation of variable, where variable specification is made within parentheses.

  e) When reading packed matrices, prevent reading beyond the last pmx-file date when fdates extends further. Fixed problem when loading packed matrix, when pmx date range is greater than vam date range.

  f) In G7 vector and matrix calculations, ignore MISSING values; treat them as zeros to to avoid spurrious missing values in results.

  g) Found trouble with the bank open GUI, where filenames that were specified subsequently were lost.

The final version of G7 in 2022 was 7.400.

1.5 *G7*: New for 2021

Work in 2021 provided new features, improvements to existing features, and a variety of bug fixes. The most important improvements and new capabilities provided in the *G7* software are these:

- Substantial progress was made in 2021 to shift development of G7 from Borland Builder 6 (a 1990s-vintage IDE) to Embarcadero RAD Studio. Most features now are working in the new environment, though the new edition is not ready yet for general release, and the change is indicated by the version number. G7.3x indicates older versions compiled under Builder, and G7.4x versions are compiled with RAD Studio.
- Improved Documentation: The *G7* documentation was revised heavily in 2012 and has been updated again in 2021, with new Help files, Users' Guide, Reference Manual, Tutorial, and other documentation. Companion documents in HTML and PDF formats also were updated. In particular, recent software improvements and extensions have been documented, and cross reference hyperlinks have been added to the Reference Manual.
- Miscellaneous Improvements:

  a) Added *%version keyword* to recover the G7 version number.

  b) Raised width and precision limits in the *punch5 command*.
- An Abridged Listing of Bug Fixes:

  a) Fixed problems in the *vmake command*.

The final version of G7 in 2021 was 7.400.

1.6 *G7*: New for 2020

Work in 2020 provided new features, improvements to existing features, and a variety of bug fixes. The most important improvements and new capabilities provided in the *G7* software are these:

- Improved Documentation: The *G7* documentation was revised heavily in 2012 and has been updated again in 2020, with new Help files, Users' Guide, Reference Manual, Tutorial, and other documentation. Companion documents in HTML and PDF formats also were updated. In particular, recent software improvements and extensions have been documented, and cross reference hyperlinks have been added to the Reference Manual.
- Miscellaneous Improvements:

  a) Added intelligent search for .pmx files to various VAM routines, where the file location is specified in the .vam file by the *pmfile command* but sometimes *G7* has trouble finding it once files are moved.
- An Abridged Listing of Bug Fixes:

  a) Fixed a bug with the *getsum command*.

The final version of G7 in 2020 was 7.38994.

1.7 *G7*: New for 2019

Work in 2019 provided new features, improvements to existing features, and a variety of bug fixes. The most important improvements and new capabilities provided in the *G7* software are these:

- Improved Documentation: The *G7* documentation was revised heavily in 2012 and has been updated again in 2019, with new Help files, Users' Guide, Reference Manual, Tutorial, and other documentation. Companion documents in HTML and PDF formats also were updated. In particular, recent software improvements and extensions have been documented, and cross reference hyperlinks have been added to the Reference Manual.
- Miscellaneous Improvements:

  a) Add *xl column delete* <column> and *xl row delete* <row> commands.

  b) Improved documentation in the Help resources.
- An Abridged Listing of Bug Fixes:

  a) Fixed a bug with the *(lis)tnames command* where *lis -s f gfnd*R* caused a hang on series *gfdn*.

The final version of G7 in 2019 was 7.38992.

1.8 *G7*: New for 2018

Work in 2018 mainly provided a variety of bug fixes. The most important improvements and new capabilities provided in the *G7* software are these:

- Improved Documentation: The *G7* documentation was revised heavily in 2012 and has been updated again in 2018, with new Help files, Users' Guide, Reference Manual, Tutorial, and other documentation. Companion documents in HTML and PDF formats also were updated. In particular, recent software improvements and extensions have been documented, and cross reference hyperlinks have been added to the Reference Manual.
- An Abridged Listing of Bug Fixes:

  a) Fixed faulty check in *show command*.

  b) Fixed a bug in *%strncmpi routine*.

  c) Attempted bug fix in the *vmake command*.

  d) Attempted bug fix in *wsreset command*, where chdir sometimes did not work properly.

The final version of G7 in 2018 was 7.38991.

1.9 *G7*: New for 2017

Work in 2017 provided new features, improvements to existing features, and a variety of bug fixes. The most important improvements and new capabilities provided in the *G7* software are these:

- Improved Documentation: The *G7* documentation was revised heavily in 2012 and has been updated again in 2017, with new Help files, Users' Guide, Reference Manual, Tutorial, and other documentation. Companion documents in HTML and PDF formats also were updated.

In particular, recent software improvements and extensions have been documented, and cross reference hyperlinks have been added to the Reference Manual.

- Graphical Interface Improvements:

  a) Modified code for *look command window* in order to speed loading of large stub files.

- Miscellaneous Improvements:

  a) Add bank title to *listbank command* output.

  b) Add sanity checks to *gridtype command* to improve stability and the users' experience.

  c) Modified the *vc command*, as *vc y = 1 / x* did not perform element-by-element division $y_i = 1/x_i$, where y and x are vectors. Instead, it seemed always to calculate $y_i = 1 \times x_i$, which perhaps is consistent with certain other features but seldom is helpful in cases like this.

  d) Improved documentation in the Help resources.

- An Abridged Listing of Bug Fixes:

  a) Fix printing problem with *dir command*.

  b) Force the graph control bar to appear in front of all other windows.

  c) Check placement of all windows to ensure that each is visible and that the user maintains control of the program.

  d) Fixed bug in xl commands that was causing incorrect columns to be read.

  e) Modified the specification of the frequency parameter when creating new workspace banks; the previous setting seemed to cause trouble in some cases.

  f) Fixed an infinite loop caused by "# %getval<newline>" where an incomplete function specification is included in a comment. Similar problems might persist.

  g) Restored lost capabilities of the *xl freeze command*.

1.10 *G7*: New for 2016

Work in 2016 provided new features, improvements to existing features, and a variety of bug fixes. The most important improvements and new capabilities provided in the *G7* software are these:

- Improved Documentation: The *G7* documentation was revised heavily in 2012 and has been updated again in 2016, with new Help files, Users' Guide, Reference Manual, Tutorial, and other documentation. Companion documents in HTML and PDF formats also were updated. In particular, recent software improvements and extensions have been documented, and cross reference hyperlinks have been added to the Reference Manual.

- Excel Interface Improvements:

  a) Extend *xl font command* to allow specification of system fonts, to control numerical separators (e.g. ',') and to provide control of text wrapping.

  b) Extend *xl font command* to allow specification of general, number, currency, accounting, date, short date, long date, time, percentage, fraction, scientific, text, off, or custom settings.

  c) Add *xl row height command* to set row height.

  d) Improve function of reading dates and corresponding data with *xl read command*.

- Graphical Interface Improvements:

  a) Allow display of data with the *show command* when title files are missing. Allow longer abreviated titles (up to 50 characters); auto-adjust row|column widths for longer abreviated titles. Show full row|column title as Hint when abreviated row|column title is clicked. Allow show command to substitute row|column titles from another file. When opening show to a specified cell, select the cell so that pressing an arrow key starts navigation from the specified location.

  b) Keep focus on *G7 Editor window* Find and Replace dialog boxes after execution to ease subsequent searches. When Editor is in focus and F3 is pressed, if a match is found then do not shift focus to Find GUI.

  c) Add Close All Banks and Assign Data Banks buttons to the *Currently Assigned Banks GUI.*

  d) Close the *Look window* when the associated bank is closed.

- Miscellaneous Improvements:

  a) Increased maximum dimensions of *resector command* keys to 2,500 rows and 20 columns (from 1,500 rows and 10 columns).

  b) Change syntax of *rs aggmatrix command* to allow operations on rectangular matrices. In rs aggmatrix and rs aggmatrows, allow key dimensions to exceed matrix dimensions (i.e. skip excessive index values).

  c) Make *type* (<expression>) print the expression before printing the data.

  d) In *G7 scripts*, in some cases allow strings to be expanded and other text within %{}.

- An Abridged Listing of Bug Fixes:

  a) Eliminate numerical problems in coef and @bmk(„g) routines.

  b) Fix memory problems in @bmk(), load, and pmload.

  c) Modify changeDir() to fix trouble that appeared in wsb.

  d) Add error checking in data command.

  e) Improve dates check in xl routines. In xl read, allow '"xldates"' as well as 'xldates'. Fixed various problems with reading an array of Excel dates. Fix problem with xl setfrequency.

  f) Fix search for existence of series when macro bank is not linked to vam bank (affecting %exists()).

  g) Close title file after reading by show command.

  h) In wsreset, close groups.bin file and reopen in new directory.

  i) Fix problem with specified target directory|file in wsreset.

  j) Make rs commands robust and improve error messages. Fix bug that required number-of-sectors specifications in the rs config file to exceed the actual number of sectors.

  k) Fix broken rs aggmatrix routine.

  l) In type command, correct problem with check of the second *tdate* that allowed buffer overrun.

  m) Improve checking of fdates range vs vam bank date range.

  n) Fix bug in @yoy().

o) Eliminated unnecessary save prompt in *G7* Config GUI.

p) Fix bug in vc command where (vc vector_x = vector_y / scalar * 100) failed when size(x) > size(y).

Improvements also were made to the companion software *Compare*.

- *Compare*:

  a) Print matrix listing titles when printing to Excel documents.

  b) Automatically set precision to zero when printing integers to Excel documents.

  c) Intelligently call integer/float/string printing function when printing dates to Excel file. Note that *Compare* currently does not print dates in Excel date format.

  d) Turn off subtitles when no text is given in the \subti command.

1.11 *G7*: New for 2015

Work in 2015 provided new features, improvements to existing features, and a variety of bug fixes. The most important improvements and new capabilities provided in the *G7* software are these:

- Improved Documentation: The *G7* documentation was revised heavily in 2012 and has been updated again in 2015, with new Help files, Users' Guide, Reference Manual, Tutorial, and other documentation. Companion documents in HTML and PDF formats also were updated. In particular, recent software improvements and extensions have been documented, and cross reference hyperlinks have been added to the Reference Manual.

- Excel Interface Improvements:

  a) Introduced ability to read a range of dates with the *xl read command*, and then to recover the dates and components with keywords and to read a corresponding range of data with possibly non-contiguous dates. The set of dates are stored in Excel date format, where a single date may be recovered with the %xldate *keyword*. The same may be recovered in *G7* date format with %xlgdate, or the date components may be recovered as %xlyear, %xlquarter, %xlmonth, and %xlday.

  b) Introduced corresponding *xl setfrequency command* to clarify the intended frequency of the dates read with the *xl read command*.

  c) Made *xl write command* print floating point values or integers to the spreadsheet when numbers are specified as text. Previously, numbers were written as strings, though strings still can be forced with "'<number>".

- Graphical Interface Improvements:

  a) Added *Run-To-Here F11 shortcut* in *G7* editor.

  b) *Added keywords* to recover main window and editor window color and font settings.

  c) Added commands to specify *main window* and *editor window* color and font settings.

  d) Added Close (Esc) to gridtype menu.

  e) Began revisions of BankOpen GUI.

- Miscellaneous Improvements:

  a) Added *setveclag command* to get|set number of lags for a vector in the default vam file.

b) Added 'c|r' option to the *coef command* to divide columns (default) | rows by corresponding elements the given vector.

c) Began work to allow regressions equations to be saved in *G7* equation format with the "gf" option with the *save command*. Change regression coefficient printing to save files to avoid "+-" sequences with negative parameter values.

d) Added an optional numerical argument for the *pause command* to wait for the specified number of seconds before continuing.

e) Add check to the *mcopy command* to ensure that source bank frequency matches destination bank frequency and that both match *fdates* frequency or specified date range frequency.

f) When *addprint* is off, do not print "add <filename>" statements to screen.

- An Abridged Listing of Bug Fixes:

a) Added sanity checks to reading of g.ini file and improved default values.

b) Improved reading of g.cfg, and eliminated infinite loop when valid g.cfg file is not found.

c) Fixed memory problems related to series titles in *graph* routine.

d) Fixed buffer overflow problem with *gtf-gti* and long titles that appeared when printing data to file.

e) Fixed bad printing of graph series names.

f) Fixed error message in showcmd.

g) Consecutive views of packed matrices in the *show* window could fail to display changes made to the data by *pmatin*; forced storage of loaded vector at top of *pmatin*, as this will force all subsequent commands to reload data and thus should fix the problem.

h) Fixed bugs in the *show* and *ras* commands.

i) Fixed up *rs ctrlmatrows* command.

j) Fixed problems affecting *xl freeze* and similar commands, where changes were not made to the intended worksheet.

k) Fixed problems in *xl vecwrite*.

l) Revised *coef* command code to avoid numerical problems.

Improvements also were made to the companion software *Compare*.

- *Compare*:

a) Added intelligent search for .pmx file locations for packed matrices. Improved error handling for matrix listing.

b) Improved control of printing zeros with the *\nzr command* when multiple banks are loaded, and fix problems with *nzr* when printing to .xls files.

c) Improved *.in config file* reading to allow "..\filename" specification, etc.

d) Added checks, error messages, and error handling to *sort routines*, particularly to the maximum number of sectors to be sorted.

1.12 *G7*: New for 2014

Work in 2014 provided a few new features, improvements to existing features, and a variety of bug fixes. The most important improvements and new capabilities provided in the *G7* software are these:

- Improved Documentation: The *G7* documentation was revised heavily in 2012 and has been updated in 2014, with new Help files, Users' Guide, Reference Manual, Tutorial, and other documentation. Companion documents in HTML and PDF formats also were updated. In particular, recent software improvements and extensions have been added, and cross reference hyperlinks have been added to the Reference Manual.
- Excel Interface Improvements:
  a) Added *xl gridlines command* to control the display and color of worksheet gridlines.
  b) Added *xl border command* to set cell borders for a given area, including color, weight, style, and position.
  c) Added the optional color specification *RGB(int1,int2,int3)* for all xl color settings.
  d) Added the *xl printer command* to set page orientation and print area, and also the ability to print directly.
  e) Extended existing routines to select and control graph sheets in addition to worksheets. These include the *xl open*, *xl create*, and *xl name* commands.
  f) Added the *xl graph title* command to set or recover a title after the graph has been created. In addition, the title font may be specified.
  g) Modified the *xl create* command to allow the new sheet to be placed either before or after the currently-active sheet.
  h) Added conditional formatting capability with the *xl cf command*.
- Strings, Functions, and Keywords:
  a) Allow multiple series to be specified in the *del command*.
  b) Added a *str store args* <root> command to store function or add-file arguments to strings named root1, root2, ...
  c) Improved the *%eliteral()* routine so that it can be applied more widely.
- Miscellaneous Improvements and Bug Fixes:
  a) Eliminated graph width problems in the G7 *graph command* that appeared when displaying long time series.
  b) Fixed problems in the *freq command*, added frequency checks to the *ls command*, and prevented the *ls command* from changing the frequency of a series.

1.13 *G7*: New for 2013

Work in 2013 provided a few new features, improvements to existing features, and a variety of bug fixes. The most important improvements and new capabilities provided in the *G7* software are these:

- Improved Documentation: The *G7* documentation was revised heavily in 2012 and has been updated in 2013, with new Help files, Users' Guide, Reference Manual, Tutorial, and other documentation. Companion documents in HTML and PDF formats also were updated.
- Interface Improvements:
  - a) Main menu items were added to load in a web browser the software documentation presented on Inforum web site and to assist the user when checking for software updates.
  - b) The *Editor F1 (Help)* key replaces the nonstandard F2 shortcut for the launch of Help files.
  - c) The G7 icon and images on the splash, configuration, and about dialogs pages was updated. The configuration dialog and About window were revised and improved.
  - d) Graph Series: Add *right-click menu item* in the Editor to display data in a graph for a selected series name or algebraic expression.
- Strings, Functions, and Keywords:
  - a) Redefined *xl name ws* <name> so that the name is optional. If not specified, then the existing worksheet name can be recovered with the *%xls* keyword.
  - b) Modified *%xlcol()* to take either a column number and return equivalent column letters or to take column letters and return the equivalent number.
  - c) Added a *-l rootname* option to the listnames routines to store the series names as strings, where the string names are *<root><integer>* and the string definitions are the series names.
  - d) Added *%nseries(bank_letter)* function to return the number of series in the specified bank.
  - e) Replaced the *%{. . . }* routine with one that does proper Parentheses/Multiplication/Division/Addition/Subtraction integer arithmetic. Other %-routines and recursion may be employed within the function.
  - f) Added *%exists(series_name)* to return 1 (0) when the series is found (not found). The *exists(<series_name>)* routine provided in the *if()* command now is depricated.
  - g) Added *%sexists(string_name)* to return 1 (0) when the string is found (not found).
- Numerical Routines:
  - a) Added *@sign()* that returns +1 (-1) if the argument is non-negative (negative).
  - b) Added *%fabs()*, *%log()* , *%exp()* scripting functions that accept a scalar argument.
  - c) Added a tolerance option to the set of *ras commands* .
- Miscellaneous Improvements:
  - a) Added *wsreset* command to allow the workspace to be cleared and a new workspace established with new parameters and possibly in a new location.
  - b) Allow "update" as a command type for the *save command*; previously it was the default option but could not be specified manually.

c) Rewrote routines that handle the scripts that specify sector groups and @-functions. This corrects garbled printing of equations by the regression commands, equation saving with the *save* command, and title printing for the () shortcut in graph command.

d) In the *resector* routines, increase the maximum number of sectors from 600 to 1500.

e) Several internal codes were respecified to allow use of character sets with 8-bit extended ASCII representations.

f) Added sanity checks for the dates in *g.cfg* files.

- An Abridged Listing of Bug Fixes:

a) Prevent overflow or run-on words when printing long series names with the *listnames* commands.

b) Fixed a problem with focus with the "file-execute" box on the main window.

c) Fixed problem in the *vc* command that caused incorrect date alignment between Vam banks and packed matrix files.

d) Fixed += in *f* and *vf* to handle missing values on both the LHS and RHS.

e) Added check to *@chain* routines to ensure that the specified base date frequecy matches that of the *fdates*.

f) Force destruction of existing *show* window before a new one may be created; this finally should fix persistent bugs. Improved the show window save functions though problems remain.

g) Fixed crash in Editor pop-up menu *Type Series* command when multiple lines were selected.

h) Fixed problem with expansion of *%linelen*.

i) Fixed problem with printing of settings with the *line* command.

j) In *ipch* command, switch from scientific notation to standard notation for printed coefficients.

k) In *wsdump*, fixed problem when printing a series with only missing values, and fixed problem with dates that could lead to incomplete printing of data, and eliminate automatic zapping after dumping data.

l) Attempt to fix bug in *function* command where an existing argument list could be lost, and thus the attempt to restore it fails after the function was executed with an alternative list.

m) In *findvec*, ensure that the Vam file is open before attempting to search it.

n) Fixed *f* command to allow "f x{2008-%fdates2} = y", where previously the *%fdates2* keyword was not processed properly.

o) Fixed problem in *%strcmp()* and related functions where the return value was not set properly.

p) Reworked *find()* to do a proper Vam-file search when called by *G7* routines like *%exists()*.

q) Attempted to fix a problem with *listnames* command where the G bank associated with a Vam file was not searched.

Improvements also were made to companion software, including the table-making program *Compare* and the model-building program *IdBuild*.

- *Compare*:

  a) Updated the *chain-weighting routine* to incorporate *G7* updates.

  b) Added many new *printer control* (*pc*) commmands and added documentation for all. Added documentation for the *printing of these files*.

  c) Improved the appearance of screen formatting.

  d) When performing zero/zero or zero/nonzero calculations, set the result to zero instead of a missing value.

- *IdBuild*:

  a) Increased buffer sizes for bank paths/names. Fixed problem with bank names that are surrounded by quotes.

  b) Fixed memory leaks and other problems. Merged code with non-optimizing version of *IdBuild* to incorporate recent improvements to that version. Improved check for divide-by-zero problems. Fixed problem with *@csum()* groups.

  c) Changed byte alignment to byte from quad word to ensure consistency with other programs.

  d) Added code to initialize vectors and matrices to zero to eliminate problems with garbage in unused portions of series (i.e. dates beyond *fdates* ranges).

  e) Improved the appearance of screen formatting.

1.14 *G7*: New for 2012

Work in 2012 provided some important new features, improvements to existing features, and a variety of bug fixes. The *G7* documentation was revised heavily, with new Help files, Users' Guide, Reference Manual, Tutorial, and other documentation. Several new routines have been added to the Software Demonstrations section of the Inforum web site to display the capabilities of *G7*.

The most important new capabilities are these:

- Improved Documentation: The Help system for *G7* and related software has been revised. The new system has native support in recent operating systems, including Windows 8, 7, and Vista. Companion documents also were created in HTML and PDF formats.
- Workspace Enhancements:

  a) Added *wscache <on/off>* command. When *caching* is on, reading and writing to the standard workspace (WS.*) is suspended and data are held in memory. When first turned on, data are read from the WS.* files, and when first turned off data are written to the hard drive in the WS.* files. Capacity in the cache is greater than on disk because of known limitations of the GBanks (WS.*) design, and so pushing data from memory to disk will fail if the WS.* bank capacity is exceeded. In some cases, script processing speed will increase dramatically with use of this routine. The size of the cache typically is limited by the two-gigabyte memory limit for 32-bit programs.

  b) Added *wsdump* command to *dump data* from the cache to a text file. Data may be printed according to specified *tdates*. Otherwise, the routine will strip from the series any leading and trailing zeros, missing values, and garbage, and it only will print data that appear to be valid. The resulting text file may be read later by *G7* or it may be compiled by *Banker* as a compressed bank.

- Data Display Improvements:

a) Show Command: Restored the ability to store values that were modified in the *show window*. Added a “Save” button to the show menu, and users now will be prompted to save changes when the window closes. If the *show* window is closed by typing ESC, then upon exit the ESC code will be cleared to prevent the script from being killed since ESC also is used to interupt the processing of a *G7* script.

b) Look Command: Improved performance for display of *look* window, where loading of a long stub file was noticably sluggish. The *look window* now automatically will adjust the frequency of *gdates* and *tdates* according to the frequency of the selected series.

c) Graphs: Improve internal error code returns for *type* and *graph* commands; this improves the user experience, particularly when these commands are employed within loops or subroutines. Improved layout of graph *legends* and allow longer series names to be displayed. Partially restored the display of *line styles*. Allow up to seven series to be shown at once. Allow *algebraic expressions* to be graphed.

d) Type: Add *right-click menu item* in the Editor to type data for a selected series name or algebraic expression.

- Improvements to the Excel Interface:

a) Added *xl freeze* command to allow a *spreadsheet to be frozen* at a specified location.

b) Introduced the *xl background <color>* routine to set *background colors* of cells.

c) Added *xl merge* to allow *cells to be merged*.

d) Added *xl subscript* and *xl superscript* to *modify the contents of cells*.

e) Introduced the *xl graph* family of commands to create graph sheets built with data from the current worksheet.

f) Modified the *xl write* routine to print a series of dates in Excel date format.

- Strings, Functions, and Keywords:

a) Introduced the *%formatgiven1*, *%formatgiven2*, *%formatgiven3*, *%format1*, *%format2*, and *%format3* keywords to indicate whether the *format* has been set and to recover the current settings.

b) Enhancement of the *%s() routine* that recovers the contents of named strings.

c) Added %if(<condition>, <if true>, <if false>) *function* for general usage within scripts, where specification of <condition> is identical to that of the numerical *@if()* command.

d) Added “+=” operator to extend string contents with the *str command*.

- Miscellaneous Improvements

a) Improved editor *Find and Replace* routines.

b) The *Editor F10 (Execute)* routine now operates in memory and no longer employs temporary text files; this improves speed and eliminates file clutter.

c) Added script controls for main menu items that control the *command box history*: *commandcache <clear>*, *commandcache <print>*, and *commandcache <replay>*.

d) Modified the *@mean()* and *@stdev()* *functions* to have the *@ggr()* syntax and fixed problems in all three.

e) Add a counter check to the *chain-weighting routines* to prevent overflow when reading series names. Increased the maximum number of series in the weighting process to 1500.

f) Increase the *maximum number of variables* for the *matty* command from 30 to 100.

g) Improved the printing of bank contents with the *lis* and *lnc* *commands*.

h) Added *function clear* command to remove *user-defined functions*.

i) Added *timer* command to *calculate time intervals*.

j) Added an optional date to control the *extent of operations* in the *ls* command.

- An Abridged Listing of Bug Fixes:

  a) Interface: Enabled the Edit button in the *show* window. Fixed an interface problem with the *Compare* window. Fixed a broken menu item in the Editor (Insert Text From File).

  b) Strings, Functions, and Keywords: Fixed problems in *%getval()*. Fixed problems with parsing of *%<integer>*. Fixed precision problems that caused trouble for the *%ceiling()* and *%floor()* functions. Fix an endless loop condition caused by malformed "%{. . . (. . .). . . }" scripts. Fixed a problem with the specification of the *%xls* keyword when an empty cell is read. Revised parsing of words within % routines so that text wrapped in "" is treated as one word. Modified treatment of unrecognized '%' in routines for which %-expansion is optional.

  c) Numerical Routines: Fixed a problem in *@yoy()* that was causing access violations. Fixed a problem in *@Xchwt()* routines that occured when the *fdates* range did not include the chain-weight base year and fixed divide-by-zero and other problems. Improved error detection in the *ls* command.

  d) Fixed printing problem with *listnames* for compressed banks.

Note that the Visual C++ Redistributable package must be installed in order to use some of the newest features. The VC++ installer should be available at C:\PDG\C++Install. Run this installer (run as Administrator if using Vista or Windows 7) before attempting to run the demo routines for strings or the resector tools.

1.15 *G7*: New for 2011

Work in 2011 provided some important new features, improvements to existing features, and a variety of bug fixes. The *G7* Help files and the G7 Reference Manual also have been revised and updated. Several new routines have been added to the Software Demonstrations section of the Inforum web site to display the capabilities of *G7*.

The most important new capabilities are these:

- Many new keywords and functions have been defined. A page in these Help files provides details on *G7* variables, keywords, and functions.
- The *G7* tools for creating and manipulating strings have been thoroughly redesigned and many extensions have been added.
- A new family of *resector* commands corresponding to the ReSector class in *InterDyme* has been introduced.
- New ability to parse arbitrary text files using the *string* command.
- Many new functions for manipulating text, performing simple calculations, and new keywords.

New graphical features and improvements for *G7* include:

- In the main *G7* window, there now is a command cache to record the entries into the command box. The command history may be displayed, replayed, or cleared using the main menu.

New and modified commands for the *G7* scripting language include:

- The vector calculate (*vc*) command now becomes more consistent with the *f* and *vf* commands with the added ability to set *fdates* command temporarily.

- New *pmmode* command controls the linking of Vam banks to packed matrix binary files.
- The *dos* command that executes system commands from *G7* offers improvesd reliability and scripting capability.
- New '<' and '>' operations to the *mmult* command to select the greater or lesser elements of two matrices.
- In the *mmult* command, either right-hand-side argument now may be a constant.
- New controls are available for the *function* command, where the *function* command allows users to record and execute sequences of commands. The implementation of the routine is improved to allow superior recursive techniques to be employed.

Note that the Visual C++ Redistributable package must be installed in order to use some of the newest features. The VC++ installer should be available at C:\PDG\C++Install. Run this installer (run as Administrator if using Vista or Windows 7) before attempting to run the demo routines for strings or the resector tools.

1.16 *G7*: New for 2010

Work in 2010 provided some important new features, improvements to existing features, and a variety of bug fixes. The *G7* Help files and the G7 Reference Manual also have been revised and updated. Several new routines have been added to the Software Demonstrations section of the Inforum web site to display the capabilities of *G7*.

The most important new capabilities are these:

- A variety of new keywords and functions have been defined. A new page in the User Guide provides details on *G7* variables, keywords, and functions.
- Users now can create and manipulate strings and refer to them by name anywhere within a *G7* script.
- Extended the capabilities of the spreadsheet interface. Provide ability to set fonts when printing text and data to worksheets. Added ability to create documents of various file types, such as comma-separated values (*.CSV) and HTML (*.HTML). Added a routine to print vector data and to insert Excel formulas into a worksheet.

New graphical features and improvements for *G7* include:

- Improvements for the graphical management of data banks.
- Added control over the auto-complete feature of the *G7* command box.
- *G7* now remembers many additional user preferences, including the precise window position for the main window, the editor, the graph window, and the graph button bar; the editor font; the editor background color; and the autocomplete setting for the *G7* command box.

New and modified commands for the *G7* scripting language include:

- Calculations of macro time series with the *f* command and vector elements with the *vf* command now may employ the +=, -=, *=, and /= operators familiar to C++ programmers.
- Up to 500 variables now may be employed in *G7* ordinary regression equations and SUR models.
- Improved stability of *@log()*, *@exp()*, and *@sqrt()* functions.
- New ability to specify additional arguments after the arguments filename in *fadd* command: *fadd <script file> <argument file> [<arg> [<arg>] . . .]*
- Added ability to calculate partial matrix sums with the *getsum* routine and allow the source data to be in a different Vam bank.

- Improved ability to print the names of vectors and matrices contained in a Vam bank.
- Improved ability to build or update data by linking series.

1.17 *G7*: New for 2009

Work in 2009 focused on fixing bugs, extending several features that recently were introduced, and adding a few new features. The *G7* Help files and the G7 Reference Manual Reference Manual also have been revised and updated. Several new demos have been added to the demo section of the Inforum web site to display the capabilities of *G7*.

New features for *G7* include:

- Improved the listing of series contained in data banks. Added the ability to sort the series names as they are printed.
- Introduced sophisticated new algorithm for forming groups. This affects all routines that employ groups, including the *add* function, the *group* routine, *do* loops, *xl* routines, many Vam routines, and others. The routine allows additional flexibility, reliability, and the capability to include named groups or specify spreadsheet column groups.
- Added the capability to include short messages with the pause command.
- Extended the capability of the *if-else* flow control routines by introducing the *eval(<expression>,<date>)* function to extracts a value from a *G7* data bank.
- Extended the spreadsheet interface. Enable creation of new documents and worksheets. Added the ability to save spreadsheets with a command from *G7*. Added capability to name worksheets and to set column widths. Added the ability to read and write in any direction. Improved the recognition of Excel error codes.
- Editor improvements: Enabled redo capability. Enabled the user to select a file name in the text and to open the file in a new editor window. Improved the editor menu. When the text is not saved and the *G7* window is closed, the editor will prompt for saving; if the user cancels, *G7* now will stay open, thus avoiding the loss of data.
- Main *G7* window: Improved the *G7* menus and tool bar.
- Font dialogs: Implement font dialog for graphs. Improve saving of font settings for the *G7* window, editor, and graphs.
- Improved the bank open dialog.
- Changed File | Directory and Change Folder button to load the current working directory when the dialogs open; previously, they loaded the location of G7.exe.
- Improved the interface for generating tables.
- Reworked GUI control of InterDyme models, including the interfaces for *IdBuild*, *Fixer*, *MacFixer*, Run Dyme, Tables, and Express Tables. Implemented control of each auxiliary program through batch files, thus giving the user additional ability to control those operations. *InterDyme* models now may be run simply by typing F8.
- Several new *G7* functions have been added to compute standard deviations, various growth rates, and to make other calculations with time series data.

1.18 *G7*: New for 2008

Work in 2008 focused on fixing bugs and adding a few useful features. In addition, these *G7* Help files and the G7 Reference Manual Reference Manual have been extended and updated. This new documentation better describes the capabilities of *G7* and guides the model builder through difficult steps of data development. Please find a full report on recent improvemnets written by Ronald Horst, entitled A Software Development Summary for the 2008 Inforum World Conference, on the Inforum web site. Finally, we launched a new section of the Inforum web site to host demo scripts and how-to files for *G7*.

New features for *G7* include:

- New *G7* functions and features: *@<high>to<low>*, *@<high>to<low>e*, *@<high>to<low>s*, *@<high>to<low>max*, *@<high>to<low>min*, *@bmk*, and *@hpfilter*.
- Several powerful extensions to the *vmake* command.
- Improved *fadd* command: the maximum number of arguments for the *fadd* command has been increased, along with other improvements.
- New *if-else* tests and other flow control capabilities.

1.19 *G7*: New for 2007

We summarize here the work on a variety of extensions to *G7* that were completed in 2007, along with a number of other improvements. The work leaves *G7* more reliable and powerful, simultaneously lessening the burden on the model builder while extending his capabilities. In addition, these *G7* help files and the G7 Reference Manual have been extended and updated. This new documentation better reports the capabilities of *G7* and guides the model builder through difficult steps of data development. Please find a full report on recent improvments written by Ronald Horst, entitled IT at Inforum in 2007, on the Inforum web site.

Improvements include:

- New regression tools for Industry Data: the *eqpunch*, *ipch*, *tpch*, *punch*, and *titpch* commands.
- New *G7* Functions: *@if*, *@yoy*, *@ggr*, *@min*, *@max*, *@qchwt*, and *@pchwt* functions.
- An extention to the *vf* command: new inline date capability.
- Saving text: now append text to file with the *catch* and *save* routines.
- Dumping data: new options for the *matty* and *save* commands to print data as text in compact formats.
- Creating series from spreadsheet input: an extension to *G7*'s spreadsheet interface.
- Incremental adjustment of dates: extensions to the *fdates* routine and *limit* routine.
- Error handling: new capabilities for the user to manage errors.
- More arguments: the maximum number of arguments for the *do* loop, *add* command, and other routines has been increased.
- Table displays: Specify parameters to control the display of data in the *show* command window.

CHAPTER

TWO

G7 USERS' GUIDE

2.1 Getting Started

2.1.1 What is *G7*?

G7 is the regression analysis member of a closely integrated package of programs for building economic models. It can be used to look at data, to estimate a single equation or a system of equations, to build a simultaneous equation model, or to create a multisectoral model involving hundreds of equations. It has extensive capabilities for:

- Making, using, and exploring data banks. Over twenty different banks can be assigned at once. Data can be shown in graphs or in grids resembling spreadsheets.
- Rapidly drawing and annotating graphs and saving them so that they can be imported into documents.
- Producing tables of data and modeling results.
- Editing and displaying text files.
- Combining and transforming variables. Transformations available include algebraic, exponential, logarithmic, sine, cumulation with decay, interpolation, change of frequency, linking and benchmarking of series, cutting off negatives, filtering, and converting all positive elements to 1.
- Looping, such as over sector numbers and titles.
- Performing regressions by ordinary least squares, least squares with soft constraints and distributed lags, stacked regression with constraints across equations, pooled regression, SUR, 2SLS, 3SLS, recursive regression, ARIMA, and non-linear least squares. Regression results include – besides the usual regression coefficients and t-values, R^2, ρ, and DW statistics – for each variable its marginal explanatory value, elasticity at the mean, beta coefficient, and the normalized sum of squared residuals after its introduction. F statistics for groups of variables are available, as is the leverage variable for finding outlying observations. If requested, the matrix of simple correlations among all the variables, the derivatives of regression coefficients with respect to one another, and the variance-covariance matrix of the regression coefficients can be shown. Statistics for testing whether the residuals are normal are available. Information necessary for stochastic simulation can be saved. Missing values in any variable are noted.
- Building simultaneous equation macro-economic models. An enormously useful feature absent in other programs is the ability to create exogenous variables, such as tax rates, conveniently at the point at which they are used. The building process checks the model for inconsistencies in the definition of variables. A simple check on the correctness of the model's identities is provided. Trend projections of all exogenous variables can be generated automatically. The building process writes a C program for solving the model. This program then is automatically compiled and linked, usually in a few seconds. It then operates extremely rapidly. Also, any calculations not easily expressed in *G7* commands can be accomplished by C code that is passed through to the program for solving the model. Models easily

can be run with various sorts of modifications to the results of the regression equations. Stochastic simulations can be run.

- An objective function, specified by the model builder, can be optimized relative to selected regression coefficients. If the objective function relates to how well the model reproduces history, this ability can be viewed as an advanced econometric method for fitting equations. If it is a measure of welfare, it can be viewed as a method of design of optimal policy.
- Building interindustry models or other models that require extensive matrix and vector operations. Such models can include a linked system of national or regional models or a model of bilateral world trade at a detailed commodity level. Results of these models can be shown in grids resembling spreadsheets. Optimization is also available for these models.

G7 Versions 7.0 and higher require the Windows 2000 or newer operating system.

The first portion of this Help system is a User Guide intended to be read in order by clicking on the *Next* and *Previous* links on the Help reader form. The second portion is a Reference Manual.

2.1.2 Installation

To install *G7* on a machine for the first time, put the first installation CD into the drive of the computer, click on the "Start" button in the lower left corner of the screen, select "Run", and type "d:setup.exe", where 'd' is the drive letter for your CD/DVD drive. If installing from the Internet, download the self-extracting executable file PDG and run it. By default, the files will be extracted to the C:\Temp directory. From Windows Explorer, navigate to the c:\temp\disk1 folder and double-click on the Setup.exe icon. If you are updating an earlier edition of *G7*, please first follow the instructions below for uninstalling the older edition.

When setup is complete, please follow the instructions below to modify the path on your machine before continuing. When finished, open *G7* by clicking the icon on your desktop or the menu button on Start->Programs. A Configuration Utility will open. You may run the introductory demo program by clicking the *G7* icon on the desktop or on the Start menu, then navigating to the C:\PDG\Demo\Intro directory and opening G.CFG in the config window. Once *G7* loads fully, type BASICG.ADD in the *G7* command box.

To install the illustrative macroeconomic model – AMI – download and extract the file. If you intend to build simultaneous equation models, you also must install some version of the Borland C++ compiler. Two files are affected by your C:\PDG\RUN.RES file. As distributed, they assume that the compiler is called BCC32.EXE and resides in C:\Borland\BCC55\BIN. If that is not the case, edit these files to make the obvious changes in them.

THE \PDG DIRECTORY

This directory contains Inforum's *G7* and *Compare* software. Use *G7* to look at data, estimate regression equations, or build simultaneous equation models. Use *Compare* to make tables for printing or for use in spreadsheets. See \PDG\DOC for the reference manuals.

The programs in this directory consist of the following:

- G7.EXE - An econometric and data management program.
- COMPARE.EXE - Table making program.

Helpful Information About Inforum Program Files

The recommended directory for these programs is C:\PDG, although most programs will work if you put them in another directory. Make sure that this directory is on the PATH.

Add the PDG directory to the path in Windows Vista or Windows 7:

1. Using the mouse, right-click on the "Computer" icon (on your desktop or Start menu) and choose "Properties".

2. Click “Advanced system settings” in the left column.
3. Click on the “Environment Variables…” button.
4. Highlight the “Path” System variable (bottom).
5. Click on the “Edit…” button.
6. Append the line with “;C:\PDG;”
7. Click OK (in the “Edit System Variables”)
8. Click OK (in the “Environment Variables” window) and click OK (in the “System Properties” window).
9. If you have a command box open, close it and reopen.

Add the PDG directory to the path in Windows 2000 or XP:

1. Using the mouse, right-click on the “My Computer” icon (on your desktop or Start menu) and choose “Properties”.
2. Click on the “Advanced” tab.
3. Click on the “Environment Variables…” button.
4. Highlight the “Path” System variable (bottom).
5. Click on the “Edit…” button.
6. Append the line with “;C:\PDG;”
7. Click OK (in the “Edit System Variables”)
8. Click OK (in the “Environment Variables” window) and click OK (in the “System Properties” window).
9. If you have a command box open, close it and reopen.

If you had a previous version of *G7* installed, then you might notice some problems the first time that you start *G7*. If so, then you can clear the configuration files that *G7* creates. Problems appear most often with Windows Vista or 7 since a copy of the configuration file is created for each user. First, change the file display settings to show hidden files. Next, find the configuration file at C:\USERS\<USER_NAME>\APPDATA\LOCAL\VIRTUALSTORE\WINDOWS\greg.ini Rename the file or move it to another location. Restart *G7*. It will find the original copy of greg.ini that was installed from the CD. This original copy will be stored in C:\Windows\greg.ini. If those default settings are not suitable, the settings in the file may be modified with a text editor. Windows 2000 and XP maintain only one copy of greg.ini, located at C:\Windows\greg.ini; this file also may be modified using a text editor.

:: To employ some of the newest features in *G7*, your system must have an installation of the Microsoft VC++ 2005 Redistributable package. You can find a copy of the VC++ installer under C:\PDG\VC++Installer. Run the installer before attempting to run *G7*.

HOW TO UNINSTALL PDG

Previous versions of the PDG (or *G7*) installation must be removed before this new version can be installed. First, be sure to make copies of any files in your PDG directory that you want to save. Next, uninstall the old version of PDG or *G7*.

In Windows 7 or Vista:

1. Open the Control Panel.
2. Switch to Classic View.
3. Click on “Programs and Features”.
4. Highlight “G7”.

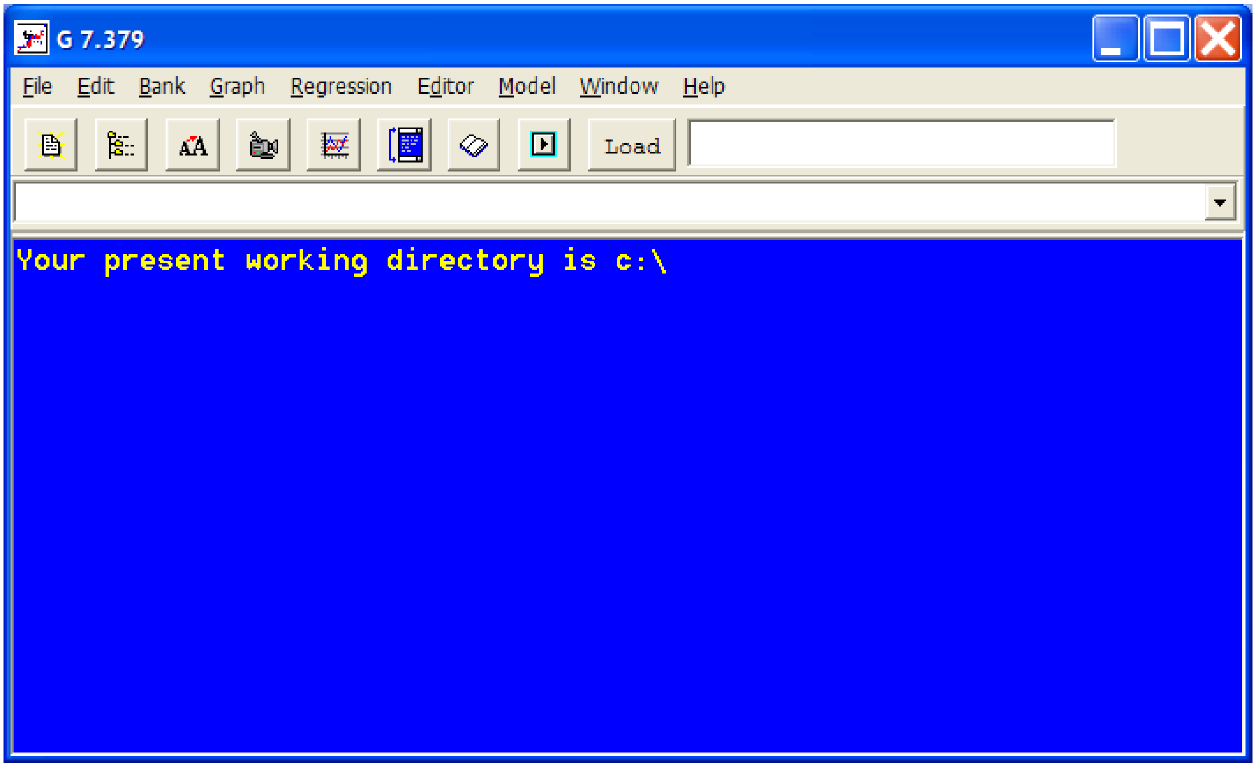
5. Click Uninstall.

In Windows XP or 2000:

1. Open the Control Panel.
2. Switch to Classic View.
3. Click on “Add or Remove Programs.”
4. Highlight “G7”.
5. Click Remove.

2.1.3 Starting *G7*

Double click the *G7* icon to start the program. After an initial splash screen, you will see a configuration utility that specifies various parameters for *G7*. *G7* looks for a G.CFG file in the folder from which it starts, and this file determines the characteristics of the workspace that *G7* creates and the initially assigned data bank. For example, to follow the tutorial, you should select the C:\AMI folder by selecting the “Browse” button in the utility window, navigating to the appropriate directory, and opening the G.CFG file found there. If you exit *G7* normally, your choice of folder will be remembered the next time you return. If you wish to use the same directory and setup, you have only to confirm the initial selection. Click the “Help” button on the configuration utility window for more details.



```
Your present working directory is c:\
```

The default font font for the results window is rather small to allow all of the results of a regression to be seen in a form only half the width of the screen. Many users, however, prefer to use a larger font. Click File | Font to pick your font. Some displays are organized under the assumtion that the font is monospaced, so FixedSys is a good choice, as is Courier in a higher point size. If you exit *G7* normally, your choice of font will be remembered next time you start *G7*. Try it; after changing the font, give *G7* the *q* command in the white command box (or use ‘Alt-F4’, or File | Exit).

Shortcuts

Many shortcuts can be executed by pressing the 'Alt' key plus a character that is underlined in the corresponding *G7* menu command. For example, 'Alt+D' will launch the *G7* editor. Other useful shortcuts include:

F1
Launch *G7* Help Files

F7
Launch the *G7* macro model dialog window

F8
Launch the *G7* InterDyme model dialog window

F9
Execute in Execute command box

Other Feaures

The command box provides an auto-complete feature to speed the typing of repeated commands. Sometimes this feature proves troublesome. It can be turned off by clicking File | Autocomplete, or it can be turned off by typing the following command in the command box:

autocomplete [<setting>]
Settings may be "yes" or "no". If no setting is given, then the current setting is displayed. The setting will be saved when *G7* is closed, and the same setting will be restored when *G7* next is run.

Command Cache
These menu items provide control over the command-line cache, where each command-box entry is recorded. Available menu items allow printing of the cached commands, clearing of the cache, and execution of the cached commands. The same control is offered throuth the commands *commandcache <clear>*, *commandcache <print>*, and *commandcache <replay>*.

G7 Main Window Properties

Set properties of the *G7* main window, where font names and colors may be taken from the lists in the xl help pages and where other options may be specified as <0|1>, <yes|no>, <false|true>, or <off|on>.

mainfontname <name>
Set the typeface in the main *G7* window.

mainfontsize <size>
Set the size of the type in the main *G7* window.

mainfontcolor <color>
Set the font color in the main *G7* window.

maincolor <color>
Set the background color of the main *G7* window.

mainautofontsize <0|1>
Specify whether to adjust the font size automatically in the main *G7* window.

mainfontbold <0|1>
Specify the bold setting in the main *G7* window.

mainfontitalic <0|1>
Specify the italic setting in the main *G7* window.

mainfontunderline <0|1>
Specify the underline setting in the main *G7* window.

mainfontstrikeout <0|1>
Specify the strikeout setting in the main *G7* window.

autocomplete <0|1>
Specify the autocomplete setting for the command box in the main *G7* window.

2.1.4 Looking at a Series in a Data Bank

To look at a series in one of the assigned data banks, click on the Bank menu and select Look (we will refer to this sequence as Bank | Look). A list of the currently assigned banks will appear in a list box. Click on one of these banks, and then a window will open in the upper right corner of the screen. It has a list of the data series in that assigned bank. Run the cursor down the list until you find one that interests you. Tap 'Enter'. Below you will see a graph and to the left, in the results area, the numerical values of the series. (The first number in each line is the date of the first observation in that line; then follow the data for four successive quarterly observations.) The cursor will have gone back to the command box, but if you wish to see another series, click on the look window again and continue looking. If you tap a letter key, the cursor will move to the next entry beginning with that letter. For a deeper search, click Find on the menu bar, and fill in the word or phrase which you wish to find. Notice that you can search up or down, with or without sensitivity to case. A case-sensitive search for "income" will not find "Income" or "INCOME"; without case sensitivity, the same search would find both. Another way to get the *look* command is from the command box. Type "look" if there is only one assigned bank, or "look <bank_letter>" if there is more than one.

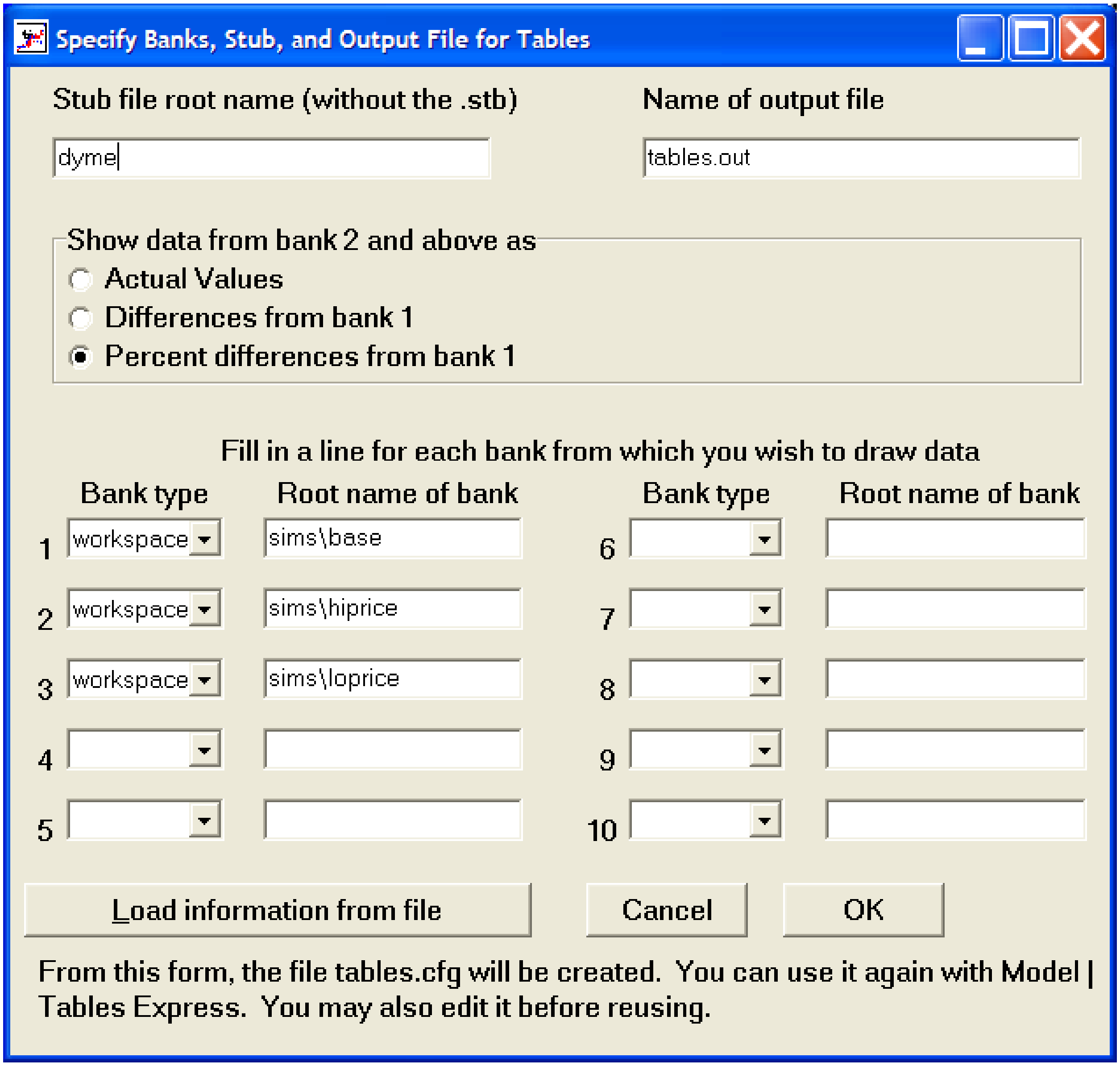
It is not necessary to close the look window while doing other things. It may be closed, however, by clicking the x in the upper right corner.

The look command will work only if there exists in the directory with the bank a file with the same root name as the bank has but with the extension .STB. This file is called a "stub file" and has lines containing the series name, a semi-colon, and a full descriptive title of the series. For example,

```
gdp ; Gross Domestic Product
```

For another example see the QUIP.STB file in the AMI directory.

Several methods are available to load additional or alternative banks. One method is to select Bank | Assign Data Banks from the *G7* menu. The following window will appear, allowing you to browse for banks, type in the name of a bank, or to remove a bank that already is loaded. In addition, findmode may be set, a default vam bank declared, and the vammode setting specified for each vam bank.



Specify Banks, Stub, and Output File for Tables

Stub file root name (without the .stb): dyme

Name of output file: tables.out

Show data from bank 2 and above as

○ Actual Values

○ Differences from bank 1

● Percent differences from bank 1

Fill in a line for each bank from which you wish to draw data

	Bank type	Root name of bank		Bank type	Root name of bank
1	workspace	sims\base	6		
2	workspace	sims\hiprice	7		
3	workspace	sims\loprice	8		
4			9		
5			10		

From this form, the file tables.cfg will be created. You can use it again with Model | Tables Express. You may also edit it before reusing.

Additional means of managing banks will be presented later.

2.1.5 The *G7* Editor

Although you may use any ASCII text editor to edit the scripts, configuration files, and other text input that you may need to operate the software, *G7* features its own text editor that works much like Windows WordPad. (WordPad itself is available under the File item of the main menu.)

To start the *G7* editor, choose File | Editor. The editor window will appear in the upper right corner of the screen. To edit an existing file, choose File | Open . . . , and then pick a file from the dialog box. When you have finished working on the file, select Save or Save As from the File menu. There is no need to close the editor before using the file.

By clicking Editor on the *G7* main menu, you can open a new editor window while leaving the existing one open. The new one, however, will be in the same upper right corner; to see them both, you must drag the top one to a new position or select the desired window from the Window menu in the main *G7* window. You may choose a default position for the editor window, so that *G7* will open each new editor window in the desired place. To establish this default position, open an editor window and then click File | Save Window Position in the editor menu.

Either clicking the Find menu item on the Editor main menu or tapping 'Ctrl-F' will bring up a standard Find dialog to search for a string. To search for the same word again, simply tap the 'F3' key. A Find and Replace tool can be opened by clicking the Replace menu item or by tapping 'Ctrl-R'.

Clicking Save on the Editor's menu or tapping 'Ctrl-S' saves the current contents of the editor. "Save As ..." is on the File menu, as are commands for changing the font and background color. Although you can choose special fonts and colors, none of that formatting is saved when you click the Save button, only the ASCII text. This behavior is because the formatting information would stop the files from working correctly as input to *G7*. However, the selected font information is saved to the *G7* configuration file (GReg.ini), and these settings will be restored each time an editor window is opened; text in the editor will be displayed in the chosen font.

Clicking Run on the Editor's menu or tapping 'F9' saves the current contents and executes the file as a *G7* command file.

Shortcuts

Many shortcuts can be executed by pressing the "Alt" key plus a character that is underlined in the corresponding *G7* editor menu command. For example, "Alt+R" will execute the file in the *G7* editor. Other useful shortcuts include:

Ctrl+f
Find

Ctrl+r
Replace

Ctrl+o
Open a new file in the same editor window, or open the selected file name in a new window.

Ctrl+s
Save

F1
Launch *G7* Help files

F3
Find again

F9
Execute the file that is open in the editor

F10
Execute the text that has been selected in the editor.

F11
Execute the text from beginning of file to the cursor.

G7 Editor Window Properties

Set properties of the *G7* editor windows, where font names and colors may be taken from the lists in the xl help pages and where other options may be specified as <0|1>, <yes|no>, <false|true>, or <off|on>.

edfontname <name>
Set the typeface in the editor window.

edfontsize <size>
Set the size of the type in the editor window.

edfontcolor <color>
Set the font color in the editor window.

edcolor <color>
Set the background color of the editor window.

edfontbold <0|1>
Specify the bold setting in the editor window.

edfontitalic <0|1>
Specify the italic setting in the editor window.

edfontunderline <0|1>
Specify the underline setting in the editor window.

edfontstrikeout <0|1>
Specify the strikeout setting in the editor window.

Mouse Menu

Right-click on the Editor window to bring up a menu. Menu options allow the standard Cut, Copy, and Paste operations. In addition, a filename may be selected from the dialog and the file name will be inserted into the text, or another file may be opened in the current window. Finally, if a series name or algebraic expression has been selected, then the "Type Series" option will print the data to the output window and the "Graph Series" option will display the data in a graph.

2.1.6 The *G7* Command Reference

There are currently over 130 *G7* commands. These can be grouped into several categories as listed below.

The Basic 11 G7 Commands
Learn these commands first; they are used most frequently.

Conventions
This section covers the conventions used in this reference.

Dates
This section discusses the use of annual, quarterly and monthly dates in *G7*.

Assigned Data Banks
Discusses the distinctions between the various kinds of banks in *G7*, and the use of the Workspace Bank.

Forming Variables
This section discusses how to use the *f* command to form new variables as functions or expressions of existing variables.

Functions
This section describes the various functions available with *G7*.

Command Files, Groups, and Do Lists
This section discusses how to use the add command, the fadd command, and related commands to automate your work.

Read Data
How to read data into *G7*.

Write ASCII Data
Write ASCII files containing *G7* data.

Writing Data To Lotus WK1 Files
Write many *G7* series to a WK1 file.

Making Tables
Make tables of *G7* banks using Compare.

Drawing Graphs
Discusses how to make various kinds of graphs using *G7*, including how to save them as files and print them.

Ordinary Regression
How to use the (r)egress command to do a simple linear regression.

Soft Constraints
Use soft constraints to impose a priori expectations on estimated coefficients.

Distributed Lags
How to easily estimate distributed lags, including Almon lags.

Regression Tests
Special types of hypothesis tests available in *G7*.

Hildreth-Lu
Use of the Hildreth-Lu command for autocorrelation corrections.

ARIMA
Use of ARIMA and Box-Jenkins techniques.

SUR
Use of Seemingly Unrelated Regressions.

Miscellaneous
Miscellaneous topics not covered elsewhere.

Non-linear Regression
A few non-linear estimation techniques available in *G7*.

2SLS and 3SLS
How to use two-stage and three-stage least squares in *G7*.

Model Building
This is one of *G7*'s strong points! How to use *G7* for building models using *Build* and *InterDyme*.

The Workspace Bank
Various tricks related to the use of the *G7* workspace bank.

C Regression
How to do pooled regression in *G7*.

2.1.7 The 11 Basic *G7* Commands

This is a quick introduction to these commands. Each is described in greater detail in the command reference of this User Guide and in the Reference Manual. Links to more detailed help are provided. In this basic reference, the full name of the command is given, and the abbreviated version, if available, is enclosed in quotes. Examples are highlighted in Courier fixed width font.

(ti)tle <text>
Provides a title for regressions and graphs.

For exmple,

```
title Consumption Function
```

f <variable> = <expression>
f <variable>{<date1> [- <date2>]} = <expression>

Defines the variable on the left in terms of the variables in the expression on the right.

For example,

```
f x = gnp/(1 + unemp[1]*@exp(0.25*time))
f index = x/x{2005.1}
```

Note:

[]
denotes lags, where (t-1) is denoted [1]

@
introduces a function (see the list of functions)

{<date>}
on the right-hand side denotes a specific observation.

{<date>}
on the left-hand side assigns values over the given range of periods.

(lim)its <date1> <date2> [date3]
(lim)its [<±n1> <±n2> <±n3>]

Sets limits for regressions and for @sum() commands. Estimation begins at date1; ends at date2. Simulation or forecast to date3. Quarterly dates require .1, .2, .3 or .4. Monthly dates require .001 . . . .012.

For example,

```
lim 1975.1 2011.4 2012.3
```

The following example adjusts the limits setting. If limits previously were set to 1975.1 to 2011.4 to 2012.3, then the adjusted range spans 1975.2 to 2011.2 to 2013.2.

```
lim +1 -2 +3
```

(r)egress <y> = <x1>, <x2>, <x3>, . . . , <xN>

Regresses y on x1, x2, etc.

The x's may be simple expressions.

```
r gnp$ = g$, v$-vi$
```

(gr)aph <y1> [y2] [y3] [y4] . . . [y7] [date1 [date2 [date3]]]

Graphs up to seven series from date1 to date2. Dates are retained from previous *gr* commands. (Note: the *(pl)ot* command is identical to the *graph* command.)

For example,

```
gr v$ c$ g$ 1979.1 2011.4
```

gr *

This graphs actual and predicted values from the last regression.

(ty)pe <y> [date1] [date2]

Displays the values of y from date1 to date2 on the screen. Dates need not be repeated if unchanged from previous *ty* commands. (Note: the *(pr)int* command is identical to the *type* command.)

For example,

```
ty gnp$ 1979.1 2011.4
```

data <name>
<date> <observation1> <observationN>

Introduces data into workspace. First number on each line is date of first observation on line. End *data* with a ';'.

For example,

```
data sales
2005.1 117 123 134 142
2006.1 137 143 145;
```

(ba)nk <bankname>

Assigns a data bank. What this means is discussed more fully in the topic on assigned banks.

For example,

```
ba mybank
```

(lis)tnames <w|a>

Displays the series names in the workspace (w) or the assigned bank (a).

For example,

```
lis a
```

This command also has a "wildcard" option, which allows you to list variable names matching a certain pattern.

For example:

```
lis a out*
```

will list all of the series in the assigned bank that start with out (such as out1 through out85).

(ad)d <filename>

Execute commands from named file.

For example,

```
add invest.reg
```

(q)uit

Quits and exits *G7*.

That is,

```
q
```

2.1.8 Conventions

This section describes conventions used in the *G7* command reference, and in specifying dates in *G7*.

In command descriptions, angle brackets <> enclose items which must be specified but which vary each time the command is used. The sign | separates arguments or parameters of which one must be chosen. Arguments that are optional and may be omitted are shown in square brackets []. Command abbreviations are shown in parentheses.

When *G7* commands are referred to in the text they are in italic, and abbreviation of commands are enclosed in parenthesis. File names and extensions are capitalized. For example: "Many features of the *G7* environment are defined in the G.CFG file." Names of keystrokes or key combinations are enclosed in single quotes. For example: "At the end of a *G7* command, hit the 'Enter' key for the command line to be processed." or "Since *G7* is a standard Windows program, it can be minimized with 'Alt-Spacebar'."

Examples in the text that should be typed verbatim by the user, or snippets of files, are shown in highlighted boxes with Courier font.

In describing menu items, X | Y means first select menu X and then from it select item Y. Where available, accelerator keys are underlined. For example: "To change the appearance of the graph, used the Graph | Settings menu."

If you don't know which command does what you want, it may be helpful to search for keywords using the Help files. Open the Help files from the *G7* menu, and then click the Search tab or navigate to the Index page to find the keyword.

2.1.9 Basic Regression

Now let us suppose that you would like to run a regression of real Gross private fixed investment on past changes in real GDP and an estimate of the investment required to replacement capital that is wearing out. If the cursor is not already in the command box, click there. We first set the limits for the regression, the starting and ending dates. To estimate the regression with data from 1998 first quarter to 2011 fourth quarter, type:

```
lim 1998.1 2011.4
```

and tap 'Enter'. (If your data bank has data more recent than the fourth quarter of 2011, then by all means use it.) You should see your command echoed in the results area. Then give a title to the regression with the *title* command:

```
ti Gross Private Domestic Fixed Investment
```

Next, you must form 'd', the first difference of real GDP, by using the *f* command:

```
f d = gdpR - gdpR[1]
```

The remaining variable we need is requirements for replacement, which you may calculate by:

```
f replace = 0.05*@cum(stockf, vfR[4], 0.05)
```

It is not necessary for the regression, but you may be interested to see how this replacement compares to investment spending. To draw a graph with the two, use the *graph* command:

```
gr replace vfR
```

Now you are ready for the regression itself. Just issue the *r* command by typing

```
r vfR = replace,d[1],d[2],d[3],d[4],d[5],d[6],d[7],d[8],d[9],d[10],d[11]
```

There now should appear in the results area the numeric display similar to that shown for Model 5 in Chapter 1 of The Craft of Economic Modeling. The font in the results area is set small so that you see most of the regression output. Many people, however, find that it is uncomfortably small. If you stretch the window to the right with the mouse, the font size will automatically increase. If you want the larger font without stretching the window, click File | Auto Font Size, so as to remove the check mark by this item. Then click File | Font and set the size. Also, a good choice of font is "FixedSys", which has a fixed size. You may also set the font color and "script" or character set. If you are using in titles and comments a language such as Greek, Russian, Turkish, or Polish with characters that do not appear in the Western European languages, be sure to select the proper character set for the results window. It automatically will be carried over to the editor, the "look" command, and other displays. By clicking File | Background color, you can select the background color of the results window. Your choice of font (typeface, size, color, and character set) and your choice of background color will be remembered when you next start *G7*. (This generally pleasant feature can be a confusing if you are using *G7* on a public machine in a computer lab; depending on the operating system, you might start off with the settings of the previous user.)

To obtain the graph of the regression just done, give the command

```
gr *
```

To see the graph better, it may be maximized by clicking the maximize button, the middle button in the upper right corner of the graph. Click again the button in the same place to get the graph back to its original size. If you would like to save the graph for comparison with later graphs, select Graph | Shift1 Graph | Shift1 from the main menu. The graph shifts to a smaller window in the upper right of the screen. You can also use Graph | Shift2 Graph | Shift2. Thus, you can view three different graphs at once.

You now have seen four examples of *G7* commands – *title*, *f*, *r*, and *gr*. Although most of what you need to do can be done with a dozen or so commands, there are more than one hundred in all. Should you wish to see a list of them, see the G7 Reference Manual Reference Manual .

2.1.10 Command Files and the *G7* Editor

In order to save your work, you should put the commands necessary to do regressions into text files and then execute (or add) those files. To put them into a file, open the *G7* editor. Click File | Editor from the main menu and a simple editor opens in the upper right screen. Type into it

```
lim 1975.1 1999.2
```

On the Editor's menu, select File | Save As, and save the file with the name VFR.REG. Now, go back to the main window, and do File | Execute and in the dialog box select the name of the file and then click the OK buton, or type in the name and tap 'Enter'. Either way, you now should see the results and be faced with a dialog box demanding to know what you want to be done with the graph. For the moment, click the Continue button.

A second way to execute the VFR.REG file is to enter the commands

```
add vfR.reg
ti Gross Private Domestic Fixed Investment
f d = gdpR - gdpR[1]
f replace = 0.05*@cum(stockf, vfR[4], 0.05)
r vfR = replace,d[1],d[2],d[3],d[4],d[5],d[6],d[7],d[8],d[9],d[10],d[11]
gr *
```

in the white command box command box in the main *G7* window. The results are exactly the same.

2.1.11 Saving Results

Your command file, VFR.REG, is fine so far, but it does not save the results in a way which can be put into a paper or included into a model. Click on the editor window and modify the file there so that it is

```
addprint y
catch vfR.cat
save vfR.sav
gdates 1975.1 1999.2
lim 1975.1 1999.2
ti Gross Private Domestic Fixed Investment
f d = gdpR - gdpR[1]
f replace = 0.05*@cum(stockf, vfR[4], 0.05)
r vfR = replace,d[1],d[2],d[3],d[4],d[5],d[6],d[7],d[8],d[9],d[10],d[11]
gr *
save off
gname vfR
gr *
catch off
```

Save this file and "add" it again. (You will find that the command box remembers your previous commands. You can get them back either with the down and up arrow keys or by clicking on the arrow at the right end of the command box.) When the program stops and demands to know what to do with the graph, click the "Save" button.

You will have made three new files:

VFR.SAV
: the results to be used later by the computer in building a model. This file was made by the "save vfR.sav . . . save off" pair of commands.

VFR.CAT
: the results to be brought into a word processor. This file was made by the "catch vfR.cat . . . catch off" pair of commands.

VFR.WMF
: the graph in a form to be brought into a word processor. This file was made when you clicked "Save" on the "Continue add file" dialog box.

You can look at VFR.SAV with *G7*'s editor. If the aqua green editor window is showing, click on the Open item on its menu and open the VFR.SAV file. Otherwise, first click File | Editor on the main menu to create an editor window. You will see that the VFR.SAV file contains the commands to create the variables and the regression result written as an equation.

To appreciate the uses of the other two files, you should open your word processor. Open your word processor and import VFR.CAT into it. In WordPerfect or Word, you use the command Insert | File. When you do so, you are apt to find that the columns look wavy. To correct that problem, put the imported text into a monotype font such as Courier or Line printer. To do so, select the text whose font you want to change, click on the button whose caption shows your present font, and select from the list. If you are using fonts that contain a number of different character sets, such as Cyrillic, Greek, West European, East European, you should be sure to select also the "script" (i.e. the character set) that you desire.

When you clicked "Save" on the "Continue add file" dialog box, you caused the graph to be saved as a Windows metafile, a file which has the instructions for redrawing the file on any device supported by Windows. This file has the name VFR.WMF. It may be imported into a word processing program such as OpenOffice or Word. In Word, the command is Insert | Picture. When it first is imported, the graph may be small; but it can be stretched by dragging with the mouse. Getting graphs to go where you want them is a problem with all word processors. You have a choice of attaching the graph to a page, paragraph, or character. Probably character is your best choice.

You now should close the word processor and return to *G7*.

Let us explore the other options on the "Continue add file" dialog box. To do so, give the command "add VFR.REG" again. (By using the up and down arrow keys you can find this command among those already executed and just execute it again.) The program will stop, as before, with the "Continue add file" dialog box demanding to know what to do with the graph. The simplest answer is to click the "Cont." button to continue the add file, throwing away the graph. Next to it on the right is the "Stop" button, which exits from the command file. Further to the right are the "Max" and "Normal" buttons which blow up the graph to fill the full screen and restore it to its normal size, respectively. Then comes the "Save" button, which, as we know, saves the graph as a Windows metafile with the filename specified by a previous gname command. The "Print" button sends the graph directly to the printer. The "Shift1" and "Shift2" buttons move the graph to storage areas 1 and 2 where the remains visible as the add file moves on to subsequent commands. A graph in these storage areas can also be resized. Finally, the "Zip" button will cause *G7* to continue without stopping to display the next graph.

In working with multisectoral models, it is common to have add files that draw graphs for many sectors. Especially for use with them, the *gname* command can be given with a name and number. For example,

```
gname out 1
```

The name of the save file for the first graph drawn will then be OUT1.WMF, for the second OUT2.WMF, and so on. Note that the numbers increase with each graph drawn whether or not it is saved.

If you have a graph showing and would like to save it but are not in an add file and do not have an appropriate name already assigned for the file, use Graph | Save from the main menu. You will be prompted for a file name.

2.1.12 Setting Graph Styles

While it is very simple to get a quick graph in *G7*, it also is possible to refine the graph. In order to produce nice looking reports, it is worth digressing from model building for a moment to learn what sorts of graphs you can make.

The dates over which series are graphed is controlled by the *gdates* command . This command may be followed by two or three dates, for example:

```
gdates 1980.1 2010.1
gdates 1980.1 2010.1 2012.2
```

Graphs drawn following the first of these commands will cover the period 1980.1 – 2010.1. Graphs drawn following the second cover the same period but have a vertical line following 2010.1. This vertical line is useful, for example, to divide history from forecast or the period over which a regression was fit from the period over which it was tested. Exception: graphs drawn with the _gr *_ command use dates set by the *limits* command for the previous regression. For greater variety in your graphs, choose Graph | Settings on the main menu. You will get a window for setting the characteristics of up to seven lines on the graphs. You can select the color, width (in pixels), style (solid, dotted, dashed, dash dot, and dash dot dot – these work only for lines 1 pixel wide), mark, fill, and left and right parameters. The last three apply only if the type "bar" is chosen for the mark. The fill has to do with what sort of hatching appears in bars. The left and right parameters, numbers between 0 and 1, show where, within the space allocated for the bar, the left and right edges of the bar should go. For example, with left = 0.1 and right = 0.9, the bar will be in the center of the allowed space, and the bars will be four times as wide (0.8 units) as the space between them (0.2 units). To eliminate the line connecting points marked by bars, set its width to zero. Experiment with what you can do with these options. If you wish to save your graph settings for a future run of *G7*, click the "Save to file" button at the bottom of the screen. You will be asked for a filename. Any valid file name is acceptable, but let us suppose you say GRAPHS.SET. Then the next time you start *G7* you can give the command "add GRAPHS.SET" and have your settings back. You even can add at the bottom of the G.CFG file the line

```
Initial command; add GRAPHS.SET
```

and you automatically will have back your settings.

Normally, *G7* will adjust the vertical scale automatically to include all points on any series that will be graphed, but only the top, bottom, and middle of the vertical scale are labeled. You can, however, control the vertical range vertical range of the graph with the *vrange* (or *vr*) command. For example,

```
vr 0 500 1000 1500 2000
```

will cause graphs to be drawn with a range from 0 to 2000 and horizontal lines at 0, 500, 1000, 1500, and 2000. These lines will be labeled. The default behavior is to put the labels inside the box of the graph, but you can put them outside by first giving the vertical axis label command, *vaxl*. It can be

```
vaxl out
```

to put them outside of the frame of the graph or

```
vaxl in
```

to put them inside the frame. The *vr* command continues in force until another *vr* command is given. To simply turn it off and go back to the default, use

```
vr off
```

One particularly useful form of the *vr* command is

```
vr 0
```

which sets the lower boundary of the following graphs to be 0, while the top is picked by the program to fit in the highest point in any series. Try graphing gdpR with careful vertical axis labeling done with the *vr* and *vaxl* commands.

In addition to the *title* command, which we have used frequently, there is the *subtitle* command, which can be abbreviated as *subti*, that provides a subtitle for the graph. The *legend* command controls whether or not the legend normally at the bottom of the graph will be included or not. The format is "legend yes" or "legend no". Try using the *subtitle* and *legend* commands.

Besides the ordinary *gr* command, there are three other variants of it. The *mgr* or multi-graph command chooses a separate scale for each series graphed. For example, you may want to put the Treasury bill rate, rtb, and Residential construction, vfrR, on the same graph, but they have totally different scales. The answer is to make the graph with *mgr*; for example,

```
mgr rtb vfrR
```

Semi-logarithmic graphs are popular because series growing at a constant percent per year appear as straight lines. It is, however, often desirable to label the vertical axis in the original units. This is done by the *lgr* command. For example, to graph the Standard and Poor's composite index of 500 stocks from 1980.1 to 2010.2, we could do

```
f lsp500 = @log(sp500)
vr 100 200 400 800 1600 3200 4800
gdates 1980.1 2010.2
lgr lsp500
```

Do it and note the straightness of the line and the labeling of the vertical axis in the original units.

Finally, for graphs in which the horizontal axis is not time but some other variable, use the *sgr* or scatter graph command.

2.1.13 Command Files Within Command Files

The next step is to use add files within add files. Look at the RUNALL.REG file for the AMI model with the editor. You will find

```
add vfR.reg
add viR.reg
add fiR.reg
```

Adding or executing this one file will cause three other files to be executed. Try it. After the first graph has been drawn, you will find that the program stops and demands what you want to do with the graph. The simplest answer is to click the "Cont." button to continue the add file, throwing away the graph. Next to it on the right is the "Stop" button, which exits from the add file. Further to the right are the "Max" and "Normal" buttons that blow up the graph to fill the full screen and restore it to its normal size, respectively. Then comes the "Save" button, which saves the graph to the file specified by a previous *gname* command.

For the use of arguments with add commands, see the section below on *G7* features for multisectoral modeling.

2.1.14 Setting the Properties of the Results Area

At this point, your first results will have scrolled out of sight, but you can use the scroll bars on the blue results window to bring them back into view. You can also use standard Windows cut-and-paste techniques to move material from this window to a word processor. The result may be somewhat disconcerting at first, for the color of the text (yellow) will move with the text but not the color of the background. If your word processor document has a white background, you will then have yellow text on a white background. You might need to use *G7*'s File | Font command to change the color of the font to black before copying the text.

As you work, the results window may become so full that tiny movements of the scroll bar produce big changes in what text is shown. You can clear the results window at any time by clicking File | Clear results or by giving the "clear" command in the command box or in an add file. The results area is large but not infinite. If you come close to filling it up, the oldest part at the top automatically will be deleted.

You will notice also the item File | Auto font size. By clicking it, you can toggle a check mark next to it. If the check mark is visible, then widening or narrowing the results window by dragging its right border with the mouse will cause the size of the font to change so that the basic regression display fills the screen. If you do not want this automatic change in the size of the font, click this item so that the check mark disappears. This feature has no effect if using the FixedSys font.

All of the commands under the Edit item on the main menu apply to the results window.

2.2 Data Banks

2.2.1 Assigned Data Banks

G7 databanks consist of at least two files: an index file that contains a list of series names and a data file. An optional third file, called a stub file, gives code names of the variables and full titles. *G7* always has access to one databank, the workspace bank, and often also has access to a second "assigned" bank. The "assigned bank" may be changed during a *G7* session by one of the "bank" commands explained below. The "workspace" banks normally are not swapped during a session. (See the Workspace Bank topic.) All data (with an important set of exceptions) introduced into *G7* or created during a session will be stored in the workspace (not the assigned bank), no matter how many different banks are assigned during the session. When *G7* needs a series, it scans the name list of the workspace and then, only if it doesn't find the variable there, it scans the name list of the assigned bank. (Preceeding the name of a variable with a letter forces *G7* to skip looking in the workspace and to look only in the assigned bank corresponding to that letter.) The assigned bank at position 'a' to which *G7* has access initially is specified in the G.CFG file in the default directory.

All series created with *f* commands or introduced with *data* commands are put into the workspace, even if the *(up)date* and *mupdate* commands are used. By design *G7* provides no facility for writing directly into an assigned bank. This limitation protects against inadvertently changing a series in a data bank. Building or modifying banks is explained below.

The assigned bank may be one of five types:

Standard Banks
Exactly the same as the workspace banks. All series begin in the same period, all have the same number of observations, and all data points are represented by 4-byte floating point numbers. The file with the body of the bank has the extension .BNK, while the file with the index has the .IND extension. The number of series is limited to about 65,000, though in practice the maximum number of series will be far less. Simply renaming the files of the workspace after exiting from *G7* turns it into a standard bank. The maximum number of observations per series currently is 1,999, but the number may be reduced for efficiency by an appropriate entry in the G.CFG file.

Compressed Banks
The body of the file has been compressed by recording the starting and ending date of each series and eliminating from each series periods for which no data are available. Where possible, the series is stored as first differences represented by two-byte integers. The body of the bank has the extension .CBK, while the index has the extension .CIN. The number of series is limited by the requirement that the total number of letters in the names of all the variables should be less than 65,000. In practice, that limit is about 7000 series. The Press program converts a bank to a compressed bank. Do not use a compressed bank with dummy variables, of values only 0 or 1, as the 0 observations are treated as missing values.

Hashed Banks
The body is similar to the Compressed bank, but the index file has been put through a "hashing" process that permits such banks to contain millions of series, so that the number of series is no longer an effective limitation on the size of banks. The body file has the extension .HBK; and the index file, .HIN. The same warning about dummy variables applies. Hashed banks usually require less memory than compressed banks.

Vam Banks

Vam banks are files that hold vector and matrix data and often are used in the development of *Interdyme* models. *InterDyme* is the system of programs and C++ classes used by Inforum to develop macroeconomic interindustry models. The vector data in these files can be read, manipulated, typed, and graphed in *G7*. If a vector is named “out”, to show the series for sector one, you could refer to it as “out1”. Vam files have the extension .VAM.

Dirfor Banks

Dirfor files are another format used by Inforum in conjunction with macroeconomic interindustry models. These files hold multiple vectors of data, as well as a set of “macrovariables”. Each Dirfor file needs a companion file in the current directory called DIRFOR.DAT, or it cannot be read. Dirfor files have the extension .DFR. Dirfor files now represent an obsolete format replaced by Vam files.

The five different bank types readily are distinguished by the extension part of their file names. These extensions are as follows:

Type of Bank	Data File Extension	Index File Extension
Standard bank (workspace)	.BNK	.IND
Compressed bank	.CBK	.CIN
Hashed bank	.HBK	.HIN
Vam file	.VAM	*None*
Dirfor file	.DFR	*File should be DIRFOR.DAT*

Here are some commonly-used commands for working with banks.

(ba)nk <bank_name> [<bank_location>]

The command will makes the specified standard, workspace-type *G7* databank *bank_ name* the assigned bank.

Example: If a bank consisting of the two files PRICE.BNK and PRICE.IND is in the default directory, the command:

```
ba price
```

makes “price” the assigned bank. If the bank is in another directory, use the entire path name in the command; if the path contains spaces or other special characters, wrap the path and filename in quotation marks (”. . . ”). The <bank_location> argument is optional, and may be any letter between ‘a’ and ‘z’ except ‘w’. To assign the price bank in position ‘c’, you would use the command:

```
ba price c
```

cbk <bank_name> [<bank_location>]

Make the specified compressed bank the assigned bank.

hbk <bank_name> [<bank_location>]

Make the specified hashed bank the assigned bank.

vam <bank_name> [<bank_location>]

Makes the specified vam file the assigned bank. If a workspace-space type bank (that is, of the .BNK, type) exists, it normally also will be assigned to the same letter.

Thus, if BASE.VAM, BASE.BNK, and BASE.IND all exist, the commands

```
vam base b
graph b.gnp
```

will cause *G7* to look for gnp first in the vam file, but if it does not find it, *G7* also will look in the BASE.BNK bank. If this behavior is not desired, the *vammode* command can be used; see below for details.

dfr <bank_name> [<bank_location>]

Makes the specified Dirfor file the assigned bank.

G7 may have up to 25 assigned banks, using the letters ‘a’ through ‘z’ (but not ‘w’), using of any of the five bank types discussed above.

(lis)tnames [-srgv] <w | a> [wildcard]

Types the names of the variables in the workspace (w) or assigned bank (a), where “a” is any letter between ‘a’ and ‘z’ except ‘w’. If the *save* command is on, the list will then appear both on the screen and in the saved file. Use of wildcard is optional, and like in DOS, ‘*’ will match any number of characters. For any single character, however, you need to use ‘ˆ’, instead of ‘?’ as in DOS. Option ‘s’ sorts the series in alphabetical order, and ‘r’ reverses the order. If a Vam file is associated with the G bank, then option ‘g’ prints only macro series and option ‘v’ prints only Vam bank series. If the *-l* option is specified and a root name is provided, then the a set of strings will be defined. The string names will be constructed as the root name followed an integer, and the string definition will be the name of the series. One string will be created for each series in the bank.

listnamescol <bank_location> [<wildcard>]

lnc [-srgv] <w | a> [wildcard]

The *listnamescol* or *lnc* command works just like the *lis* command, but all series names in the workspace (w) or assigned bank (a) are printed in one column. You may find it convenient to capture output to a file, and then use this command to get a list of all of the series in the databank in that file. Then, using an editor that supports macros, you can change all lines to create an addfile that does the same thing with each series in the databank. Use of wildcard is optional, and like in DOS, ‘*’ will match any number of characters. For any single character, however, you need to use ‘ˆ’, instead of ‘?’ as in DOS. Option ‘s’ sorts the series in alphabetical order, and ‘r’ reverses the order. If a Vam file is associated with the G bank, then option ‘g’ prints only macro series and option ‘v’ prints only Vam bank series.

(btit)le <bank_location>

Displays the title and configuration of the workspace (w) or assigned bank (‘a’ through ‘z’ except ‘w’). In the case of the workspace, you can supply a new title or change the existing one. It is useful to record the time and date of creation of data banks. This can be achieved easily by using the keywords “%time” and “%date” in the title. *G7* will replace these keywords with the actual time and date, respectively. This capability also can be used elsewhere in *G7*, including in the *vtitle* and *ic* commands.

close <bank_location> [additional locations]

This command closes the assigned bank at the specified location. If the bank to be closed is a vam file, then this command also will close the associated workspace-type bank, if it is present and open. The command *close all* will close all open banks.

vammode <s | a>

Sets the mode for linking vam banks to corresponding macro banks.

For example,

```
vammode s
```

makes subsequent *vam* commands assign only the vam bank. The ‘s’ option is for simple. The other option, ‘a’ for associated, is the default setting, where

```
vammode a
```

causes subsequent *vam* commands to assign an “associated” workspace-type bank if it exists.

2.2.2 The Workspace Bank: Making, Compressing, and Splicing Banks

At any one time, *G7* has one workspace bank and 0 to 25 assigned banks. The workspace bank is the same physical bank throughout any one run of *G7* unless it is changed using the *wsbank* command explained below. Whenever *G7* needs a series, it looks first in the workspace and then, only if it does not find the variable there, it looks through the assigned banks. The original assigned bank, at position 'a', is specified in the G.CFG file that is read when the program starts. Additional banks may be loaded, replacing the bank in the 'a' slot or assigned to other positions. As *G7* forms variables with *f* commands or introduces variables with *data* commands, the newly created or introduced series are put into the workspace bank. Even the *update* and *mupdate* commands do not change the series in the assigned bank but put the revised series into the workspace.

(lis)tnames [-srgv] <w | assigned> [wildcard]

lnc [-srgv] <w | a> [wildcard]

Types the names of the variables in the workspace (w) or assigned bank (a), where "a" is any letter between 'a' and 'z' except 'w'. If the *save* command is on, the list will then appear both on the screen and in the saved file. Use of wildcard is optional, and like in DOS, '*' will match any number of characters. For any single character, however, you need to use '^', instead of '?' as in DOS. The *lnc* routine is identical to *listnames* but prints the series as a single column. Option 's' sorts the series in alphabetical order, and 'r' reverses the order. If a Vam file is associated with the G bank, then option 'g' prints only macro series and option 'v' prints only Vam bank series.

(bti)tle <w | assigned> [bank title]

Displays the title and configuration of the workspace (w) or assigned bank ('a' through 'z' except 'w'). In the case of the workspace, you can supply a new title or change the existing one. It is useful to record the time and date of creation of data banks. This can be achieved easily by using the keywords "%time" and "%date" in the title. *G7* will replace these keywords with the actual time and date, respectively. This capability also can be used elsewhere in *G7*, including in the *vtitle* and *ic* commands.

(wsb)ank <bank_name>

This makes the named bank the workspace. The bank name is expressed as in the bank command, above. Beware: the workspace now becomes the named bank and this bank will be changed by almost anything you do. In particular, a zap command (see below) will destroy the bank completely. Use this command with utmost caution. It is best to back up a bank before using it with this command.

(wsi)nfo

This command prints information to the screen about the current workspace bank, including its bank title, the number of series in the bank, the maximum number of observations per series, and the default starting year (base year) and period.

wscache <on|off>

wscache <yes|no>

When on, reading and writing to the standard workspace (WS.*) is suspended and data are held in memory. When first turned on, data are read from WS.*, and when turned off data are written to WS.*. Capacity in memory is greater than on disk because of known limitations of the GBanks design, and so pushing data from memory to disk will fail if the WS.* bank capacity is exceeded. In some cases, script processing speed may increase dramatically with use of this routine.

wsdump <filename> [<"tdates">]

This routine will dump data from the cache to a text file. Data may be printed according to the current *tdates* if the <"tdates"> option is specified. Otherwise, the routine will strip from the series any leading and trailing zeros, missing values, and garbage, and it only will print data that appear to be valid. This text file may be employed with *Banker* to compile compressed banks.

zap [<baseyr>[<starting period> [<nobservations>]]]
Sets the number of series in the workspace to zero. Arguments after zap override those in the G.CFG file. "baseyr" overrides the line "Default base year of workspace." "starting period" overrides the line "First month covered." "nobservations" overrides the line "Default maximum number of observations per series."

del <series_name>
Delete the named series from the workspace data bank. The WS.BNK file physically is not reduced in size by a deletion, but if another variable is added to it after the deletion, it will be put in the space formerly occupied by the deleted variable.

(ren)ame <old_name> <new_name>
Changes the name of a series in the workspace bank.

(bu)pdate <x> = <y>
bupdate stands for bank update. This command takes the series x from the workspace, updates it with non-zero entries from series y, and stores it as x in the workspace. If x is not already in the workspace, bup will put it there with values from the assigned bank. "y" may be an expression.

2.3 *G7* Commands and Scripts

2.3.1 Dates and Frequencies

Dates in *G7* either can be 4-digit or 2-digit, but 4-digit dates allow *G7* to work more reliably. A date is indicated by a year, followed by an optional decimal point, and a "period" that is used with quarterly or monthly data. For example, "83" or "83.0" or "1983" all refer to the year 1983. With annual data, the period either is "0" or should be left off. Note that when using "2-digit" dates, the year 2000 should be represented by "100", and the year 2010 by "110".

Quarterly data is indicated by a date with 1 digit after the decimal point, from 1 to 4. Thus, "2010.1" and "2010.4" refer to the first and fourth quarters of 2010.

Monthly data is indicated by three digits after the decimal point, from 001 to 012. For example, "2010.001" is January 2010 and 2010.012 is December 2010.

Note: if gnp is a quarterly series, the command

```
type gnp 1965 2010
```

will NOT print gnp for the years 1965 to 2010, but return the error message "Dates do not match frequency of variable." The correct command is

```
type gnp 1965.1 2010.4
```

freq <series name> [frequency]
Displays a series' frequency and allows you to change it. The series and its new frequency will be put into the workspace bank. Note that this is true even if the series originally was in an assigned bank. (See the topic "Assigned Banks" for more information on data banks. See the function list for functions in *G7* that will change the frequency of data, performing conversions from quarterly to annual and back, etc..)

dfreq [0|1|2|4|6|12]
Set a default frequency for the left-hand side variable of an *f* commands when the right-hand side does not have a frequency. The default is 0, or no frequency.

fdates [date1 [date2]]
fdates off
fdates [<±><n1> <±><n2>]

Sets or resets the dates used by subsequent *f* commands. When an *fdate* command is issued, it defines the time period in which subsequent *f* commands act. For a series that exists in the workspace or in the assigned bank, it modifies the value within that period without affecting the existing values outside of the period, and it puts the series into the workspace. Otherwise, it creates a series, sets the values from the right-hand side of the *f* command within the specified period, and sets missing values, that is, -0.000001, for those outside the period. "Off" resets the *fdates* to the default.

Example:

```
fdates 1980.1 2010.4
```

In this case, subsequent *f* commands only alter the value of a series between the first quarter of 1980 through the last quarter of 2010.

The default fdates are implicitly defined in G.CFG: "fdate1" is the first period of the "Default base year of workspace file" as defined in G.CFG, "fdate2" is the period implied by "Default maximum number of observations per series in workspace" in G.CFG.

Example:

```
fdates +1 -2
```

This example adjusts the *fdates* setting. If *fdates* previously were set to 1980.1 to 2010.4, then the adjusted range spans 1980.2 to 2010.2.

tdates <date1> <date2>

Sets the dates used by subsequent *(ty)pe* (or *(pr)int*) commands.

The command

```
tdates 1980.1 2010.4
```

means that the next *(ty)pe* or *(pr)int* command will display quarterly data from the first quarter of 1980 until the fourth quarter of 2010.

gdates<date1> <date2> [date3]

Sets the dates used by subsequent *(gr)aph* (or *(pl)ot*) commands. With two dates provided, the series will be graphed from "date1" to "date2". If a third date is given, the series will be graphed from "date1" to "date3", with a vertical line drawn at "date2".

gdates a
tdates a

Selects "automatic" dates for *(gr)aph* and *(ty)pe* commands. The automatic dates are the first and last date of the series actually present. The default setting in *look* is for automatic dates, unless specific dates previously have been specified. Automatic dates also adjust automatically to the frequency of the series. This feature is not to be trusted when more than one series is being placed on the same graph.

2.3.2 Forming Variables

Most transformations of existing variables into new variables are done with the *f* command. Such transformations alter the values of a variable over a period specified by an *fdates* command. The default fdates are determined by the maximum number of observations that are defined in the G.CFG file.

The general form of an *f* command is

f <variable> = <expression>

The resulting variable is placed in the workspace bank. If the left-hand variable already is in the workspace bank, then its values may be modified with the "+=", "-=", "*=", or "/=" operators. By using these operators, the right-hand side expression will be added or subtracted from the left-hand side, or the left-hand side will be multiplied by the right-hand side values, or the left-hand side values will be divided by right-hand side values. These operators follow the syntax for the C++ programming language, though here they operate over a range of values. To add the results of an expression to a variable, where the variable already exists in the workspace bank, the syntax is

f <variable> += <expression>

Example:

```
f x{1980.1-2010.4} = y*(z[3] + v*@log(w))/s{2010.1}
```

Variable x is calculated and placed in the workspace using the variables 'y', 'z', 'v', 'w', and 's'. These variables must exist already. Note in the example:

- **x{1980.1-2010.4}** Calculate the right-hand side values from the first quarter of 1980 to the fourth quarter of 2010 and copy the values into series x..
- **z[3]** means z lagged three periods, i.e. z(t-3) in a common notation.
- **s{2010.1}** means the value of s in the first quarter of 2010.
- **@log(w)** means the natural logarithm function of the variable w.

The full list of @ functions are shown in the list of functions.

Rules for f statements

Any algebraically legal expression is allowed, including nested parentheses.

The left side variable must be a single variable, or a variable with a subscript to indicate a specific time. For example, z and z{2010.1} are valid variables on the left side. However, z[1] is invalid on the left side.

For a series that exists in the workspace or in the assigned bank, the *f* command modifies the value within the period specified by *fdates* without affecting the existing values outside of that period, and it puts the series into the workspace. Otherwise, the *f* command creates a series, determines the values using the expression on the right-hand side of the *f* command for the period specified by *fdates*, and sets missing values, (-0.000001), for those observations outside of the period. If *missing n* is in effect, the observations outside of the *fdates* period will be set to zero instead of -0.000001.

For example, if we have in effect the following *fdates* command:

```
fdates 1978 2010
```

then for an *f* command

```
f x = (expression)
```

if ‘x’ is an existing series either in the workspace or the assigned bank, ‘x’ will be placed into the workspace with values from 1978 through 2010 being taken from the expression. Otherwise, ‘x’ is created with values taken from the expression from 1978 through 2010, and taken as missing for the rest of the period.

Variable names must begin with a letter and may contain up to 32 letters, digits, or the ‘$’ or ‘_’characters. Do not use a digit as the first character.

Powers are obtained using the @sq and @pow functions, not ** or ^.

Division by zero in an *f* command is legal and yields zero as a result.

Right-hand-side expressions too long to fit on a single line may be continued by using a +, -, *, or / as the last character on a line.

The commands *f*, *fex*, and *fdup* have exactly the same effect in *G7* but very different effects in *Build*. See the Model Building for the differences.

Other Commands for Transforming Variables

(miss)ing <y|n>

This is ‘y’ by default. If it is set to ‘n’, then all values that are missing in the databank will be set to zeroes. Otherwise, they will remain as missing values (i.e. -0.000001).

ls [-<flag>] <x> <y> <date1> [<date2>] [direction]

This *linkseries* command takes series x and y from the workspace or the assigned bank and calculates their ratio at the specified linking date <date1>. This ratio then is used to move ‘x’ with non-zero entries from series ‘y’. The linking direction can be ‘f’ for forward or ‘b’ for backward. The default is ‘f’. If the variable x is found in the workspace, then the result is stored with name x in the workspace. If a bank letter is provided for x, and if this bank is the default Vam file, then the modified value of x is stored in the Vam file. Otherwise, the result is stored to the workspace. If the optional date <date2> is provided, then linking will extend from the base period <date1> to the terminal period <date2>.

If the signs of the values in x and y are not the same, then the result will move in the opposite direction of the guide series; this usually is not desirable and a warning will be given. Three alternatives to the basic routine may be implemented by specifying a flag.

-f: force the calculation using standard algorithm without reporting problems. -i: if x and y are of opposite sign in the base year, then ensure positive correlation between x and y such that

x = a + b*y b = x{date} / y{date} * sign(x{date}*y{date}) a = x{date} - b * y{date}

-a: if x and y are of opposite sign in the base year and the problem is ill-conditioned such that y lies close to zero in the link year, then ensure positive correlation between x and y such that

x = a + b*y b = x{date}/ AVG(y) * sign(x{date}*AVG(y)) a = x{date} - b * y{date}

where AVG(y) is calculated from fdate1 to <date>, if direction is ‘b’, or from <date> to fdate2, if direction is ‘f’. This may be useful when y{date} is close to zero so that scaling parameter b is large, thus scaling x excessively. If AVG(y) is close to zero, then b = x{date} * sign(x{date}*AVG(y)). It is the user’s responsibility to ensure that the results are useful.

ctrl <x> <concept> <group>

Imposes the values of <x> series as a control total on a <group> of sectors of a given <concept> (such as exports). If the variables to be controlled are found in the workspace, then the results will be stored in workspace with names formed by <concept> and given sectors. If a bank letter is provided for <concept>, and if this bank is the default Vam file, then the modified values will be stored in the default Vam file. Otherwise, the results will be stored to the workspace.

For example, the command:

```
ctrl tot out 1-10 (4-7) 13 15
```

imposes the values of tot as a control total on the named sectors out, i.e. sectors 1 to 10, except for 4 through 7, and then 13 and 15.

2.3.3 G7 Functions

Arithmetic @ Functions

The following arithmetic @ functions are available for use in *G7*:

@bmk(x,y,[d|g])
benchmark function, like @lint, fills in missing values, using y as a mover series. The new series preserves movement of y as much as possible, while smoothly passing through non-missing values of x. The 'd' default method allocates differences by a linear additive adjustment. The optional 'g' growth rate method adjusts the average growth rate of y to match that of x.

@cum(y,x,z)
y equals the cumulation of x with spill rate of z. The calculation is y[t] = (1-z)*y[t-1] + x[t].

@diff(x)
the first difference of x. The calculation is (x[date2] - x[date1]).

@dlog(x)
the first difference of the natural logarithm of x. The calculation is log(x[date2] / x[date1]).

@exp(x)
exponential function of x.

@fabs(x)
absolute value of x.

@gr(x)
the growth rate of x. The calculation is 100 * (x[date2] / x[date1] - 1).

@ggr(x,date1,date2)
the geometric growth rate of x over interval <date1> to <date2>, where <date1> is the base period. The calculation is: 100*(pow(x[date2]/x[date1], (1/(date2-date1))) - 1)

@hpfilter(x,[lambda])
Implement the Hodrick-Prescott filter for x. Default values for lambda are 14400 for monthly data, 1600 for quarterly data, and 100 for annual data.

@if(condition,exp1,exp2)
if the condition is true, then calculate the value to expression exp1. Otherwise, calculate the value of expression exp2. Valid tests include <=, <, ==, >, and >= between legitimate *G7* expressions.

@ifpos(x)
= 1 if x > 0, else 0.

@lint(x)
fills in any missing observations (-0.00001) in the series x by linear interpolation except at the beginning and end.

@log(x)
natural logarithm of x.

@max(x[,date1,date2])
the maximum value of x. Optionally specify an interval from date1 to date2.

@mean(x,date1,date2)
the arithmetic mean of x over interval <date1> to <date2>.

@min(x[,date1,date2])
the minimum value of x. Optionally specify an interval from date1 to date2.

@miss(x)
replaces all true zeroes in x with missing observations.

@normal()
generates random numbers with normal distribution: N(0,1).

@pcl(y,x)
computes y[t] = y[t-1] * (1 + x[t] / 100). Note: x is the percentage increase over last period, and t starts from the first forecasting period as defined by *fdates*.

@peak(y,x,z)
y equals previous peak of x with decline rate of z for peak.

@pos(x)
= x if x > 0, otherwise 0.

@pow(x,z)
raises x to the power z.

@rand()
generates random numbers with uniform distribution over (0,1).

@round(x,n)
rounds x to n decimal places.

@sin(x)
sine of x.

@sign(x)
= 1 if x >=0, else -1.

@sq(x)
the square of x.

@sqrt(x)
the square root of x.

@stdev(x,date1,date2)
the (sample) standard deviation of x over interval <date1> to <date2>.

@sum(x)
computes the sum of x from dates specified in the last "lim" command. The sum appears in the observation of the last date.

@yoy(x)
the year-on-year growth rate of x.

@zero(x)
replaces all missing observation signs in x with true zeroes.

Frequency Conversion Functions

The following functions convert the frequency of a variable by aggregation:

@<high>to<low>(x)
converts the high-frequency series x to a lower frequency series by forming the arithmetic mean.

@<high>to<low>e(x)
converts the high-frequency series x to a lower frequency series by applying the end-of-period values.

@<high>to<low>max(x)
converts the high-frequency series x to a lower frequency series by taking the maximum.

@<high>to<low>min(x)
converts the high-frequency series x to a lower frequency series by taking the minimum.

@<high>to<low>s(x)
converts the high-frequency series x to a lower frequency series by taking the sum.

where ‘high’ and ‘low’ must be replaced with ‘m’ (monthly), ‘q’ (quarterly), ‘s’ (semiannual), or ‘a’ (annual). For example, *@mtoa(x)* converts the monthly series x to annual frequency by forming the average of monthly values. Note that sums and averages are formed over non-missing values, so that missing values are ignored.

Frequency Conversion by Interpolation

The following functions convert the frequency of a variable by interpolation:

@atoq(x)
converts the annual series x to a quarterly series by interpolation. The new quarterly series will have the correct annual total. (The annual series is cumulated; a cubic polynomial is fit to each successive set of four points; the values of the polynomial are calculated at the ends of the quarters. As for the middle two points; these values are differenced to give the quarterly series consistent with the annual totals.

@atoqi(x,y)
similar to *@atoq* but uses the quarterly indicator series y to pick the points for interpolation. To work correctly, if any value of y is available in a particular year, all values of y must be available in that year.

@qtom(x)
converts the quarterly series x to a monthly series by interpolation.

@stoq(x)
converts the semi-annual series to a quarterly series.

The functions *@atoqe* and *@qtome* convert periodicities similarly to the function of the same name except for the ‘e’ on the end. The ‘e’ functions, however, apply to end-of-period series, such as the value of assets.

The conversion functions may be used in *G7* but should not be included in models built with *Build*.

Tools for Industry Data

For use with industry models, or in other situations where variables have a number as a suffix, the *@csum* function may be useful. Its format is:

@csum(<name>[,<group definition>])

For example:

```
f outsum = @csum(a.out,1-3 5-10 (7-8) )
```

In this example, outsum is calculated as the sum of out1, out2, out3, out5, out6, out9, and out10. Numbers specified by a pair of dashes specify a range of sectors or numbers to include. Single numbers separated by spaces or commas specify single sectors to include. Finally, any group specification surrounded by parenthesis will be excluded from the summed group. If no group definition is provided, then *G7* will calculate the sum over all vector elements, either for the vector in the vam bank specified by a bank letter or, if no bank letter is given, for the vector in the default vam file. The function also can sum individual series in the workspace or other bank, but the desired sector numbers must be listed.

Chain Weighting: @pchain() and @qchain()

Since 1996, the NIPA have used Fisher chain-weighting to calculate aggregate variables in constant prices and the corresponding price indices. A Fisher index attempts to avoid distortions in index numbers formed by using weights that are inappropriate. For period-to-period movement, it is the geometric mean of a Laspeyres and Paasche index. For longer periods, the period-to-period indexes are chained together. For example, if we have quantity (Q) and price (P) data on several variables for period 0 and period 1, the Laspeyres quantity index can be written using price weights of period 0:

$$LI = \frac{\sum_i P_{i,0} \times Q_{i,1}}{\sum_i P_{i,0} \times Q_{i,0}}$$

The Paasche index is written using price weights of period 1:

$$PI = \frac{\sum_i P_{i,1} \times Q_{i,1}}{\sum_i P_{i,1} \times Q_{i,0}}$$

The Fisher quantity index then is simply:

$$FI = \sqrt{PI \times LI}$$

The Fisher price index is calculated similarly, except that the Laspeyres and Paasche components use fixed quantity weights. The convention when creating constant price chain aggregates is to define a base year, in which the price is equal to 1.0, and the quantity is equal to the nominal value. With this convenient definition, the chained quantity multiplied by the chained price yields the nominal value.

The syntax of the chain-weighting function in *G7* is:

@xchain(<list of N quantity variables>, <list of N price variables>, base year)

where x may be ‘p’ or ‘q’, and the specification of the quantity and price variables may use group expressions. The function only checks that there is an even number of variables in total. The user is responsible for ensuring that the proper quantity and price variables are entered. Also, take note that the original quantity and price variables must be in the same base year as specified in the function.

Here is an example that creates aggregates of personal consumption from the NIPA bank:

```
f qi = @qchain(c030(3-5,7-9), c03(10,11,13,15-19), d04(22-24,26-30,32,34-38), 2005)
```

Both the *@pchain()* and the *@qchain()* create two variables in the workspace bank. “chwpi” is the chain-weighted price index, and “chwqi” is the chain-weighted quantity index. The use of the name *@pchain* or *@qchain* is only relevant to which series the function expression will return.

One word of warning is in order. You might want to set *fdates* before calculating a chained index to an interval for which you have valid price and quantity data or the function might give unreasonable results.

Alternative Chain Weighting: @pchwt() and @qchwt()

An alternative chain weighting routine also is available in *G7*. Instead of specifying groups of quantity and price indexes as with the *@qchain* and *@pchain* functions, this routine requires groups of nominal levels and groups of quantity indexes. The syntax is given by:

@qchwt(<list of N nominal variables>, <list of N quantity variables>, desired base date [, zero])

@pchwt(<list of N nominal variables>, <list of N quantity variables>, desired base date [, zero])

The *@qchwt()* function returns an aggregate quantity index, and the *@pchwt()* function returns an aggregate price index. Note that both this routine and the original chaining routines now support groups of up to 1500 data series. A base date must be provided in order to scale the result, but it need not be consistent with the base date of the source data. Finally, a “zero” option indicates whether missing values should be interpreted as true zeros. At present, it also controls the routine’s ability to skip zero aggregates that may precede the actual data and following the actual data; this problem occurs when the fdates interval is too wide. If these features can be made dependable, then the detection routine need not be optional and may be made standard. The routine adds three values to the workspace: chwqi, chwpi, and chwni, which are the aggregate quantity index, the aggregate price index, and the nominal aggregate, respectively.

An example is:

```
pce_real = @qchwt( pcez(1-92), pce(1-92), 2000, z )
```

The Interpolation Function

If you do not find the function you want in the above list, but can find the function in some table, you can then help *G7* to provide your function by the general *@interp()* interpolation function. The format is:

@interp(<filename>, <x>)

It applies to x whatever function is specified by interpolation points in the named file. Up to 100 interpolation points may be given. For example, to get the vector of cumulative normal probabilities, y, corresponding to the vector of normal deviates x, do

```
f  y = @interp(cumnorm,x)
```

where the file cumnorm contains interpolation points for the cumulative normal curve. The cumnorm file might contain these lines:

```
-3.3 0.0000
-3.0 0.0013
-2.5 0.0062
-2.0 0.0228
-1.5 0.0668
-1.0 0.1587
-0.5 0.3085
 0.0 0.5000
 0.5 0.6915
 1.0 0.8413
 1.5 0.9332
 2.0 0.9772
 2.5 0.9938
 3.0 0.9987
 3.3 1.0000
```

2.3.4 Command Files, Groups, and Do Loops

Files containing *G7* commands can be prepared with a text editor and then introduced to the program by the command

add <filename> [Arguments]

The commands in the named file then will be executed as if they were being typed in at the keyboard. Add files can contain *add* commands. Add files can “add” themselves. They commonly are used to introduce data, to set up complicated regressions, and to supplement *G7*’s functions. The arguments are optional in an *add* command.

Arguments in Add Files

Add files may have up to 99 arguments. Consider the problem of calculating the percentage change of a quarterly series over the 3-quarter moving average of the series a year earlier. If x is the original series, this growth rate, grx, could be computed by

```
f ma   = (x + x[1] + x[2])/3
f grx  = (x - ma[3])/ma[3]
```

Those are fairly long formulas, so if you had to do this calculation for many different series, you would find it advantageous to make up a file with a short name like AGR.ADD (for annual growth rate) as follows:

```
f ma   = (%1 + %1[1] + %1[2])/3
f gr%1 = (%1 - ma[3])/ma[3]
```

You would use it like this:

```
add AGR.ADD x
```

The effect will be exactly as if the original pair of lines had been executed. Every occurrence of %1 will be replaced by the variable 'x'. You now can calculate the growth rate of any variable; if the name of the variable were n37z2y, the command would be

```
add AGR.ADD n37z2y
```

and the variable grn37z2y would be created.

Up to nine arguments may be used. Here is an example with two. In the example, investment of sector 32 depends on imports of sector 57 in a regression equation.

Make the file INVEST.REG as follows:

```
f d%1    = out%1 - out%1[1]
r inv%1 = out%1, d%1[1], d%1[2], rtb, rep%1, import%2
```

The command "add invest.reg 32 57" will execute these *f* and *r* commands replacing all occurrences of %1 with 32 and all occurences of %2 by 57. What would be executed would be

```
f d32    = out32 - out32[1]
r inv32 = out32, d32[1], d32[2], rtb, rep32, import57
```

Arguments are useful for estimating a similar equation form for different industries, states, or other entities.

An argument may be a text string enclosed in double quotation marks ("). Example: Let the file OUTEXP be

```
ti %1 %2
gr out%1 exp%1
```

then

```
add outexp 17 "Printing and Publishing"
```

would have the same effect as

```
ti 17 Printing and Publishing
gr out17 exp17
```

The last line of the configuration file for *G7*, G.CFG, may contain an *add* command. A typical example might be:

```
Initial add command; add INITIAL.ADD
```

Such a file typically might be used to select a printer or specify screen or graphics options.

Groups and Group Arguments in Add commands

A special type of argument is a group. The concept "group" comes from multi-sectoral modelling where sector variables have names such as out1, out2, . . . , out85. A group is a set of specific sector numbers. For example, all industries include sectors 1-85; non-durable industries include sectors 5-13, and durable industries include sectors 14-24, and so on. Here we have three groups of sector numbers: 1-85, 5-13, and 14-24. To use a group as an argument, we must enclose those sectors in parentheses like this:

```
(1-85), (5-13), and (14-24)
```

Groups also may specify sets of spreadsheet columns. For example, to construct a group of the first 30 columns of a spreadsheet, the group should be

```
(A-AD)
```

Groups also may be specified as named or dynamic groups. For example, to construct a group of all non-chemical manufacturing sectors, where manufacturing and chemical sectors are defined as

```
group Manufacturing
   1-58
group Chemicals
   20-27
```

then a group to include all manufacturing sectors except chemicals may be specified as

```
( :Manufacturing ( :Chemicals ))
```

This is equivalent to

```
( 1-58 ( 20-27 ) )
```

Note finally that these various specifications may be combined. For example, the following is another alternative to the example above:

```
( :Manufacturing ( 20-27 ) )
```

An *add* command allows its last nine arguments to be groups. Take the earlier example of calculating the percentage change of a series over a 3-quarter moving average. Suppose we have 85 quarterly output series: out1 through out85. We may modify the AGR.ADD file in the earlier example to become the following:

```
f ma     = (%1%2 + %1%2[1] + %1%2[2])/3
f gr%1%2 = (%1%2 - ma[3])/ma[3]
```

we then could write out 85 add commands as follows:

```
add AGR.ADD out 1
add AGR.ADD out 2
add AGR.ADD out 85
```

Equivalently, we can write them out in only one add command:

```
add AGR.ADD out (1-85)
```

In this example, the variable ma acts like a temporary variable, and 85 new series, grout1 through grout85, are created in the workspace. If we similarly want to compute the moving average series for another variable, such as industry sales, sale1 through sale55, we may use the command:

```
add AGR.ADD sale (1-55)
```

Again, up to nine group arguments are allowed. However, group arguments must come at the end of the argument list. If two group arguments are used, by default, the outer loop is controlled by the first group argument, and the inner loop by the second group argument. For example,

```
add INVEST.REG (32-34) (52-54)
```

is equivalent to

```
add INVEST.REG 32 (52-54)
add INVEST.REG 33 (52-54)
add INVEST.REG 34 (52-54)
```

There is an alternative to the double loop, parallel matching, with an 'm' option after the second group. For example,

```
add INVEST.REG (32-34) (52-55(53)) m
```

is equivalent to

```
add INVEST.REG 32 52
add INVEST.REG 33 54
add INVEST.REG 34 55
```

That is, the first members of each group are matched to be the first pair of arguments, the second member of each group to be the second pair, and so on. Loops with parallel matching may contain only two groups. The two groups must have equal number of members in parallel matching.

Loop with Do Command

A *do* command allows you to run a set of *G7* commands like an *add* command without creating the add file. Its format is

do{ <G7 commands with variables> }[Arguments]

where variables are %1, %2, etc., just as in an add file. A *do* command can continue on several lines. However, the arguments should be placed on the same line as the closing brace '}'. As with the *add* command, no more than 9 arguments are allowed, and the last several arguments may be groups. In fact, the last nine arguments may be groups.

For example, the AGR.ADD example can be rewritten with a do command like this:

```
do{f ma     = (%1%2 + %1%2[1] + %1%2[2])/3
   f gr%1%2 = (%1%2 - ma[3])/ma[3]
   } out (1-85)
```

In this case, however, the file AGR.ADD need not be created. A *do* command thus provides an easy way to have a loop.

Nesting of *do* loops is possible. A loop may be nested within an add file or within another do loop. For example, the following code computes the sums of the rows of matrix 'X'. The values are stored in vector rowsum. Note that an argument is passed from the outer loop to the inner loop. This is accomplished by using a variable from the outer loop to specify a group for the inner loop.

```
do{f sum = 0
  do{f sum = sum + X%2.%1
    }(%1) (1-5)
  vf rowsum%1 = sum
  }(1-5)
```

Command File with an Argument File

fadd <CommandFile> <ArgumentFile> [<arg> [<arg>] . . .]

The named CommandFile is first executed with the arguments from the first line of named ArgumentFile, then the CommandFile is executed again with the arguments from the second line of the ArgumentFile, and so on until the ArgumentFile is exhausted. Additional arguments may be supplied with the *fadd* command, and these will be added to each line of arguments provided in the arguments file.

For example, the CommandFile might be

```
ti %2
gr %1
```

while the ArgumentFile was

```
gnp$    "Gross National Product"
vfnre$  "Investment in producer durable equipment"
vfnrs$  "Investment in non-residential structures"
```

Then the result would be three properly titled graphs.

User-defined Functions

function <function name> { <G7 commands with variables> }

function clear [<name>]

G7 allows the user to define functions so that similar operations can be repeated easily. This feature is very similar to the *add* command described above, but it allows the entire script to be specified in a single file. Once a function is defined, it may be used repeatedly. *function clear* will remove all defined functions, or if a function name is specified, then that function will be removed.

Functions may contain arguments; see the description of the *add* command for details. A good practice is to specify function names that contains at least one capital letter. This will avoid confusion between the user's functions and *G7* commands and keywords. Once it is defined, a function may be called by entering the function name in the same manner as other *G7* commands. If arguments are to be passed to the function, they should be listed after the function name.

Note that the *addtype* setting automatically is set to "no" within each function, so that the function contents are not printed to screen during processing. For debugging purposes, *addtype* commands can be given within the function to override this practice.

G7 supports recursive techniques. For example, the function command may be employed to construct a Fibonacci sequence. The only restrictions for this sequence are the inherent limits of the 32-bit software, but even so, up to 47 iterations are possible before the limit is reached.

The following code repeats the example above while employing the function command.

```
function GR{
   ti %2
   gr %1
   }
GR gnp$    "Gross National Product"
GR vfnre$  "Investment in producer durable equipment"
GR vfnrs$  "Investment in non-residential structures"
```

function list

This routine will print a list of names of the user-defined functions.

function print [<function_name>]

This routine will print the specification of the function function_name. If no name is provided, then the specification of every function will be printed.

Other Commands That Are Useful in Add Files

(addt)ype <n | y>

(addpr)int

Invoked with an 'n' for "no", this command turns off the typing on the screen of the contents of the "add" files. This speeds up the processing of large data files. Turn printing back on with "addtype y" for "yes". This command is also known as *(addp)rint*.

catch <file_name>

Captures screen displays (except graphs) to the named file. Use to capture the setup of a regression and its results into a single file. A regression display is caught as it stands when you proceed to the next command. To turn off the catching type "catch off"

Commenting Add Files

Add files are like small computer programs, and should be commented. A line beginning with a pound sign, '#', is treated as a comment. A line beginning ith a colon, ':', causes that line and all subsequent lines to be treated as a comment until a line beginning with a colon again is encountered. (The line of the second colon is also a comment.)

Examples:

```
# This is a one line comment.
: This comment, however, is longer.
Clearly, it is a much more important comment,
for it takes three lines.
: Or four.
```

Pausing to View Output in Results Window

If you have a very long add file, the results may all rush by before you have a chance to examine them. There are a number of ways to solve this problem. The simplest is just to scroll the results window up to examine the output that has flashed before you. A second is to use the catch command to capture most of the output to a file, where it can then be examined with the editor. Occasionally it is convenient to just pause the operation of the add file for a moment. One way to do this is with the *pause* command. Just include the command *pause* on a single line in your add file, and *G7* will stop when it reaches that point, and display a message box on the screen. At this point, hit an 'ESC' to stop the addfile or any other key to continue.

pause [<"message">]

The *pause* command causes *G7* to pause, optionally print a message, and prompt the user to continue or to cancel the process. Note that if more than one word is to be included, the message must be surrounded by quotes.

Another way to pause is to use the *addpause* command:

addpause < y | n>

This command cause *G7* to pause just before the beginning of each add file and to display a message box. This pause feature also allows you to hit an 'ESC', or click on the 'ESC' button to stop the add file sequence.

2.3.5 Flow Control

Users can control the execution of a *G7* script by using its flow control capabilities. These commands allow the user to write compact scripts with less repetition and to perform complex tasks with greater ease.

Examples:

```
function TEST{
  if( %NARGS == 0 ){
    ic Zero arguments were passed to the function TEST.
    }
  else if( %NARGS == 1 ){
    ic The argument %1 was passed to TEST.
    }
  else{
    ic Two arguments, %1 and %2, were passed to TEST.
    }
  }

TEST
TEST ONE_ARGUMENT
TEST FIRST_ARGUMENT SECOND_ARGUMENT
```

This simple example demonstrates the use of user-build functions in *G7*. Within the function called TEST, *if-else* commands are used to determine how many arguments were passed to TEST.

```
if( %1 <= B ){
  ic A or B
  }
else if( %1 == C ){
  ic C
  }
else{
  ic D
  }
```

This example demonstrates the ability of the *if-else* command to compare strings and then to carry out a set of tasks based on the lexicographical order of the strings.

The following text documents the *if-else* routines and supporting commands.

break

This command halts execution of a *do* loop or an add file, but processing of the script continues after breaking out of the *do* loop or out of the child script.

Examples:

```
do{
  if( %1 == 2 ){ break }
  else{ ic This is iteration %1 }
  }(1-3)
```

continue

This command causes the current iteration of a *do* loop to end and the next iteration to begin.

Examples:

```
do{
  if( %1 == 2 ){ continue }
  else{ ic This is iteration %1 }
  }(1-3)
```

if([comparisons]){ [. . .] }
[else if([comparisons]){ [. . .] }]
[else{ [. . .] }]

The *if* command evaluates a series of logical expressions. If the arguments are true, then the following block of code, contained in a set of brackets, is executed. If the argument is false, then else if statements are processed in the same way. If all *if* and *else if* arguments are false, then the code following the optional final *else* command is processed.

Three types of comparisons are legal. First, a number may be compared to a second number. Second, a single number may be given and implicitly compared to zero. Finally, a string may be compared to a second string.

Valid comparisons between two strings, with the comparison based on the lexicographical ordering, or between two numbers are

<

is LESS THAN

<=

is LESS THAN OR EQUAL TO

==

is EQUALS

>=

is GREATER THAN OR EQUAL TO

>

is GREATER THAN

!=

is NOT EQUALS

!

is the logical negation operator. If given before a number, then the result is FALSE if the number is nonzero and the result is TRUE if the number is zero. If given before a set of parentheses, then the result is FALSE if the expression within the parentheses is TRUE, and the result is TRUE if the expression within the parentheses is FALSE.

Additional string comparison tools include:

%strcmpi(<string1>,<string2>)
Compares string1 and string2, without case sensitivity. Value is FALSE if strings match, following the standards of the C libraries..

%strncmp(<string1>,<string2>,<N>)
Compares the first N characters of string1 and string2, with case sensitivity. Value is FALSE if the first N characters of the strings match.

%strncmpi(<string1>,<string2>,<N>)
Compares the first N characters of string1 and string2, without case sensitivity. Value is FALSE if the first N characters of strings match.

Multiple comparisons may be performed, with the following operators joining them

> ||
> Logical OR, which returns TRUE if either the left hand side or the right hand side is TRUE.
>
> **&&**
> Logical AND, which returns TRUE if both the left hand side and the right hand sides are TRUE, and returns FALSE otherwise.

The following keywords and functions particularly are helpful, though many additional features also are useful.

> **%NARGS**
> The number of arguments passed to a function, to an *add* file, or to an *fadd* file.
>
> **%{+ - / *}**
> simple arithmetic calculations are posible.
>
> **%mod(x, y)**
> returns the remainder of 'x' divided by 'y'.
>
> **%exists(<series>)**
> returns true if variable x is found in bank a.
>
> **%getval(<series>, <date>)**
> The value of series in period date is retrieved, either from the workspace or from the specified bank. The value may be compared to a scalar or to a second eval result.

Comparisons may be evaluated as a group by surrounding them with parentheses. Blocks of code following the arguments must be surrounded by brackets {}.

Examples:

```
if( 1 - 2 < -0.5 )  { ic Subtraction }
if( 2 < 3 && 3 <= 4){ ic Multiple evaluations }
if( 2 < 3 || 3 <= 4){ ic Multiple evaluations with OR }
if( 1 == 1 )        { ic Equality }
else if( !0 )       { ic Else if and negation }
else                { ic Else }
if( D != N ){ ic String inequality comparisons }
if( D <= N ){ ic String less than or equal to }
if( %mod(7,2) == 1 ) { ic Modulo arithmetic }
do{
  if( %1 == 3){
```

(continues on next page)

(continued from previous page)

```
        ic Continuing on Iteration %1
        continue
        }
    if( %1 == 5 ){
        ic Breaking on Iteration 5
        break
        }
    if( %1 > 5 ){ return ERR }
    }( 1-10 )
```

return [arg]

This command interupts execution of the *G7* script. If *G7* is in the midst of processing a do loop or an add file, then the arguments determine the next actions of *G7*. Arguments may be given in upper or lower case.

Arguments: :OK: The default argument. Exits a do loop or add file, but processing continues. :ESC: Processing halts entirely, but no error messages are shown. :ERR: Processing halts entirely, with error messages displayed.

Examples:

```
return ERR
```

2.3.6 User-specified Error Handling

The *try-catch* combination loosely clones similar routines in C++. Within the brackets following each may be any collection of *G7* routines, including other *try-catch* blocks. If execution of any of the s within the *try* block produces an internal error signal, or if a user-specified error is thrown with the *return* , then execution of the code in the *try* block ceases and execution of the code in the *catch* block begins. Otherwise, the *catch* block is ignored. For example, consider the link-series *ls*. The routine will fail if the value of the guide series is zero in the base period. Ordinarily, this will cause *G7* to stop execution of the script and report an error. If, on the other hand, the *ls* is nested within a *try* block, then if an error occurs the alternative block of code following the *catch* will be executed. In the following example, suppose that we prefer to use guide series y when it is available, and when it is not we rely on guide series z. We first try to extend series x using series y beginning in 2005, and if the value of y is zero in 2005, then we repeat the exercise using series 'z':

```
try{
   ls x y 2005 f
   }
catch{
   try{
      ls x z 2005 f
      }
   catch{
      ic Sorry.  No data in y or z.
      pause
      }
   }
```

The *try-catch* routine works well with many other s and has a wide variety of useful applications. Note again that *try-catch* blocks can be nested, so that several alternatives can be tried before *G7* gives up in despair.

Often, anticipated errors alternatively can be handled with *if-else* routines. For example, the *f* will fail if a right-hand-side variable is not found. *G7* can handle such failures elegantly either by nesting the *f* in a *try* block or by using the *if* , with the *exists* function, to test for the availability of the variable before attempting to use it.

2.3.7 Strings in *G7*

Strings may be defined and referenced by name. These same tools may be used to open an arbitrary text file and parse the content found there.

str <name> = <rhs>

str <name> += <rhs>

Create a string named <name> with definition <rhs>. If the string <name> exists already, then <rhs> may be appended with use of the '+=' operator. The string definition <rhs> may be specified as text presented within quotation marks, given as the name of a previously-defined string, or defined as a sequence of text and/or strings linked by '+'.

For example,

```
str concept  = "Output"
str industry = "Coal Mining"
str display  = concept + "of the " + industry + " industry."
```

str save <filename>

Store all defined strings in a text file <filename>. Afterward, the strings may be loaded into memory again.

For example,

```
str save MYSTRINGS.TXT
add MYSTRINGS.TXT
```

str store args <rootname>

Store all add-file or function arguments that are in scope in a series of strings. Each string name is created from the specified root name and the position (integer) of the argument in the argument list (e.g. root1, root2, . . .).

For example,

```
# Initialize a collection of series from a list of series names. Create new series or set existing series␣
↪to zero.
function Initialize{
   str store args a
   do{ f %s( a%1 ) = 0 }(1 - %NARGS)
   }
Initialize gdp consumption investment
```

str print

Print each string to the screen.

str clear [<stringname>]

Erase all defined strings from memory. If a string name is given, then that string will be removed from the list of user-defined strings.

str replace [<stringname>] <”search_text”> <”replacement_text”>

This function operates on the string <stringname>, if provided, or the last line read by “str getline”. Any text in this string matched by the string <search_text> is replaced by the string <replacement_text>. Matching employs regular expressions as defined in the C++ Boost library. More details for regular expressions are provided on the following page.

The following routines are employed in the parsing of a text file.

str open <filename>

Open a file for parsing with the string .

str close

Close the file that was opened with the *str open* .

str getline [<string_name>]

Read a line of text from the file opened by *str open*. Store the text as a string named <string_name>. Store a copy of the string with the name “line;” this string may be referenced by related routines, but it does not appear in the list of user-defined strings.

str parse [<string>] [“<separators>” [“<string_name>”]]

Split the string named <string> into words separated by any one of the characters listed as separators. If no string name is provided, then the routine acts on the last line read by *str getline*. The default separators list is composed of the space and tab characters, or “ \t”. The words are stored as a list of strings that may be accessed with the *%w()* function, which has a syntax similar to the *%s()* function described below. If a string name is suppied, it will be used as the root name of the stored words.

Several keywords and functions aid in the parsing of text files.

%linelen

Returns the length of the last line read by *str getline*. If *str getline* encounters the end of the file, then a value of -1 is returned by *%linelen*.

%line

The *%line* keyword allows the user to access the string last read by *str getline*. It especially is useful when no string name was provided for the string.

%numwords

The *%numwords* keyword returns the number of words parsed from the line currently in memory or the string last processed by the *str parse* . After calling *str getline*, the number of words is 1 (%line == %w(1) and %numwords == 1) until a subsequent *str parse* is issued.

%w(<index> [,<first>[,<last>]])

The *%w()* function provides access to the list of words created by the *str parse* routine. <index> is an integer that specifies the position of the desired string within the list of words created by *str parse*. The optional <first> and <last> parameters are integers that specify the positions of the first and last characters to define a substring of the word in position index.

The definition of a string may be recovered at nearly any point within a *G7* script by using the *%s(<name>)* function function. To employ our example string “display”, which is defined as “Output of the Coal Mining industry.”, as a graph title, we simply issue the following .

```
gr %s( display )
```

Additional functions are available function that compare strings, calculate the length of the string, and perform other operations.

2.3.8 Variables and Arguments in *G7*

Variables in *G7* were introduced earlier for use in *add* files, *fadd* files, *do* loops, and elsewhere. This section provides additional details.

The treatment of variables in *G7* are similar to the treatment in DOS batch files. Up to 99 variables typically may be passed to an *add* or *fadd* file, within a *do* loop or *function*, and so on. The first argument that is passed to a file is referenced by '%1', the second by '%2', . . . , and '%99'. The appearance of a percent symbol (%) within a *G7* script always signifies that the following character or word should be interpreted as a variable, a keyword, or a function. In each case, when the variable, keyword, or function is encountered and evaluated, the value of the result effectively will be inserted into the script and subsequently processed by *G7*.

Variables

Variables in *G7* were introduced earlier. All arguments are passed as text to the *add* file, *do* loop, or other routine. Variables are referenced as %1, . . . , %99.

Numerical Routines

%{}

Integers often are passed to an *add* file or *function*, and integers are used as indexes for a *do* loop. Sometimes a calculation is needed that is based on that integer, but some adjustment is needed. The *%{}* operation allows such calculations. Numerical variables and constants may be referenced within the brackets, and addition, subtraction, multiplication, and division of variables and constants is permitted, and sub-calculations may be surrounded by parentheses. For example, suppose we need to print vector data to a spreadsheet. Suppose the vector elements range from 1-10, and we want to print these series in rows 6-15. We could use the "xl vecwrite" routine or we could construct a do loop like the following:

```
do{ xl write A %{5 + %1} vec%1 2000 2010 }(1-10)
```

Note that for each iteration '%1', element '%1' is referenced in the "vec" vector, and that series is written to row '5+%1' in the spreadsheet.

Another common use of *%{}* is to provide unambiguous specification of variables within a script. Recall that variables '%1' through '%99' are legal. However, there may be ambiguity between a two-digit variable and a single-digit variable that is followed by another digit that is not part of the variable. For example, if '%9' is followed by a '9', it looks the same as '%99' and so *G7* will attempt to fill in the 99-th variable definition. To remove such ambiguity, use the following approach:

```
f x = %{%9}9 # Variable 9, followed by '9'
f x = %99    # Variable 99
```

In some cases, strings may be expanded and other text also is allowed within %{}.

%floor(<number>)
%ceiling(<number>)

These function return the value of nearest integer, either rounding down or up, respectively.

%fabs(<number>)

This function returns the absolute value of the argument.

%log(<number>)
%exp(<number>)

These function return the logarithm of *number* or the exponential *e* to the *number*.

%min(<number>, <number>)
%max(<number>, <number>)

This function returns the minimum or maximum values of the two numbers provided.

%mod(<number>, <number>)

The modulus function returns the remainder of dividing its first argument by the second.

Keywords

Several keywords were introduced earlier. The following is a summary of keywords in *G7*. Availability of particular regression statistics depends on the type of regression performed.

%date
Insert the date.

%time
Insert the time.

%NARGS
The number of arguments that currently are in scope.

%%
Treat this as text, evaluated as a single '%'.

%fdate<X>
Recover the first or last *fdates* setting, where X = {1,2}.

%gdate<X>
Recover the first, middle, or last *gdates* setting, where X = {1,2,3}.

%tdate<X>
Recover the first or last *tdates* setting, where X = {1,2}.

%lim<X>
Recover the first, middle, or last *limits* setting, where X = {1,2,3}.

%title
Recover the current graph title.

%subtitle
Recover the current graph subtitle.

%vaxtitle
Recover the current graph vertical axis title.

%ncoef
Recover the number of coefficients in the last regression.

%beta<X>
Recover the parameters from the last regression, X={1,...,ncoef}.

%mexval<X>
Recover the mexvals from the last regression, X={1,...,ncoef}.

%see
Recover the standard error of the estimate from the last regression.

%rsq
Recover the r-square value from the last regression.

%rho
Recover rho from the last regression.

%rbarsq
Recover the r-bar-square value from the last regression.

%mape
Recover the mean absolute percentage error from the last regression.

%dw
Recover the Durbin-Watson statistic from the last regression.

%nobs
Recover the number of observations in the last regression.

%obsinregression
Recover the number of observations in the last regression.

%logl
Recover the log likelihood statistic from the last regression.

%xls
Recover spreadsheet cell contents.

%xldate
Recover a spreadsheet date.

%xlgdate
Recover a spreadsheet date in *G7* date format.

%xlyear
Recover the year component of a spreadsheet date.

%xlquarter
Recover the quarter component of a spreadsheet date.

%xlmonth
Recover the month component of a spreadsheet date.

%xlday
Recover the day component of a spreadsheet date.

%mainfontname
Recover the typeface setting of the main *G7* window.

%mainfontsize
Recover the type size of the main *G7* window.

%mainfontcolor
Recover the font color of the main *G7* window.

%maincolor
Recover the background color of the main *G7* window.

%mainfontcharset
Recover the character set setting of the main *G7* window.

%mainautofontsize
Recover the auto font size setting of the main *G7* window.

%mainfontbold
Recover the bold font setting of the main *G7* window.

%mainfontitalic
Recover the italic font setting of the main *G7* window.

%mainfontunderline
Recover the underline font setting of the main *G7* window.

%mainfontstrikeout
Recover the strikeout font setting of the main *G7* window.

%autocomplete
%edfontname
Recover the typeface setting of the editor window.

%edfontsize
Recover the type size of the editor window.

%edfontcolor
Recover the font color of the editor window.

%edcolor
Recover the background color of the editor window.

%edfontcharset
Recover the character set setting of the editor window.

%edfontbold
Recover the bold font setting of the editor window.

%edfontitalic
Recover the italic font setting of the editor window.

%edfontunderline
Recover the underline font setting of the editor window.

%edfontstrikeout
Recover the strikeout font setting of the editor window.

%version
Recover the G7 version number.

Functions

A variety of functions also are available. The following is a summary of such functions in *G7*.

%nseries(<bank_letter>)
This function returns the number of series in the specified bank.

%exists(<series_name>)
This function returns 1 (0) when the series is found (not found).

%sexists(<string_name>)
This function returns 1 (0) when the string name is found (not found).

%xlcol(<integer>)
%xlcol(<letters>)
This function maps positive digits to letters or letters to digits. Typically, <int> will be a variable (e.g. '%1') that is assigned an integer value. For integer arguments greater than 26, the results will follow the convention for identifying Excel spreadsheet columns. The following is the mapping for the positive sequence of integers: A. . . Z AA . . . AZ BA . . . BZ . . . .

%getval(<expression>, <date>)
The value of expression in period date is retrieved. Any algebraic expression that is legal for the *f* command may be given, or a variable may be named either from the workspace or from the specified bank.

%if(<condition>, <if_true>, <if_false>)
Insert <if_true> when <condition> is true, and insert <if_false> otherwise. The specification of <condition> is identical to that of the *if* command. <if_true> and <if_false> either may be named strings or text within quotes.

%s(<string> [, <first>[, <last>]])
Insert the value of the named string <string>, where <string> previously was declared and defined with the *str* command. Optional parameters <first> and <last> indicate the positions of the first and last characters of a desired substring of <string>. <last> will be interpreted either as the character specified or as the end of the word when <last> exceeds the length of the word.

The following functions operate on strings. The strings either may named strings that were defined by the *str* command or they may be provided as text or a variable that is surrounded by quotation marks (e.g. "x1" or "%1").

%strlen(<string>)
Calculates the length of string. This follows the standard C libraries.

%strcmp(<string1>, <string2>)
Compares <string1> and <string2>, with case sensitivity. Value is 0 if strings match, following the standard C libraries.

%strcmpi(<string1>, <string2>)
Compares <string1> and <string2>, without case sensitivity. Value is 0 if strings match, following the standards of the C libraries.

%strncmp(<string1>, <string2>,<N>)
Compares the first N characters of <string1> and <string2>, with case sensitivity. Value is 0 if the first N characters of the strings match, following the standards of the C libraries.

%strncmpi(<string1>, <string2>,<N>)
Compares the first N characters of <string1> and <string2>, without case sensitivity. Value is 0 if the first N characters of strings match, following the standards of the C libraries.

%strstr(<string1>, <string2>)
Looks for the first instance of <string2> within <string1>. If found, then return the portion of <string1> beginning at that point. Otherwise, return an empty string. This is similar to the standard C libraries.

%strnset(<string>, <N>)
Construct a string composed of <N> repetitions of string. The length of the resulting string is limited to 90 characters. This function loosely follows the standard C libraries.

%lower(<string>)
%upper(<string>)
%titlecase(<string>)
Returns a copy of string after converting it to lowercase letters, capital letters, or title case (where the first letter of each word is capitalized).

%trim(<string>[,<separators>])
%trimleft(<string>[,<separators>])

%trimright(<string>[,<separators>])

Returns a copy of string after removing extraneous characters from the beginning, end, or both ends of string. The default behavior is to remove whitespace and control characters. Other characters separators optionally may be given in the syntax of the *%eliteral()* routine.

%literal("<string>")

The *%literal()* function prevents the characters of string from modification by the *G7* parsing engine. This allows character sequences to be passed through that otherwise might prompt an error or other undesirable action.

%eliteral("<string>")

The *%literal()* function is identical to the *%literal()* function except that it escapes the sequences '0', 'a', 'b', 't', 'f', 'n', 'r', and '\', converting each to its corresponding ASCII code. For example, 't' is converted to the ASCII code for the tab character.

2.3.9 Reading Data Into *G7*

G7 has seven commands related to introduction of ASCII data, in addition to speciallized routines for reading other format such as Excel spreadsheets. The seven are:

data

introduces a single series prepared in *G7*'s data input format

matdat

introduces one or more series in a column format usually found in spreadsheet programs

update

adds new observations or revises observations in series already in a data bank

matup

updates from column-format data

monup

updates a quarterly series with monthly data

wrindex

turns on or off the writing of the index file of the workspace

More detail on the commands is given below.

(da)ta <series_name>

Introduces data into the workspace data bank. There are 2 forms the command can take.

Form 1:

```
data sales
2010.1  34.0 56.8 44.5 55.6
2011.1  39.3 41.2 43.9 47.0;
```

The first number on each line is the date of the first observation on that line. Input is terminated by a ";". The ";" may be omitted in an "add" file, except on the last line of the file.

Form 2:

```
data sales 2010.1
34.0 56.8 44.5 55.6
39.3 41.2 43.9 47.0;
```

The date of the first observation is given on the command line immediately after the series name. No dates appear on subsequent lines.

Form 1 is easiest if the data is being entered by hand. Form 2 is easier if the data is generated by a program.

Input data may contain floating point numbers in exponential form. Thus, the number 3 may be represented as 3.0, 3.0E+00, .300E+01, or 30E-01. Only E, not e, is recognized in this context. Any number of spaces between data observations is allowed (the data are free format).

A missing value may be represented by a single question mark (?). Therefore

```
data sales
2010.1 34.0 ? 44.5 55.6
```

indicates that the sales for 2010.2 quarter is missing.

(up)date <series_name>

Works like the data command (Form 1) but updates an existing series. The data following the command contain only the data points to be changed. The updated series is placed in the workspace only.

monup(mup) <series_name>

This is used for updating a quarterly series with monthly data.

Example:

```
mup rtb
   2010.3  9.120 9.390 9.050 8.710 8.710 8.960
   2011.1  8.930 9.030 9.440;
```

'rtb' is a quarterly series being updated with monthly data. 9.120 is the value of 'rtb' in July 2010; 9.390 is the value in August 2010, etc. Note that quarterly dates are the first numbers on each line and three monthly observations must be present for each quarter.

matdat [date]

Brings data into *G7* prepared by a spreadsheet program. There are two major forms in which the series can be arranged. The series can be arranged either in vertical columns with the name of the series at the top of the column, or in horizontal rows with the names appear on the same line of the matdat command. Up to 20 columns of numbers occupying up to 160 characters per line may be given. The two forms of the command are:

Form 1:

```
matdat
           gnp      c       cd      cnd      cs
   81.1    1513    950     146     359     445
   81.2    1512    949     140     361     448
   81.3    1522    956     143     361     450;
```

where dates appear as the first number on each line. Here 1513 is the value of gnp in 1981.1, 1522 is the value in 1981.2, etc.

or

```
matdat 1981.1
   gnp     c       cd      cnd      cs
   1513    950     146     359     445
   1512    949     140     361     448
   1522    956     143     361     450;
```

The date of the first observation appears on the command line, and no dates appear on subsequent lines. Use a semicolon to end the data. (You must have a semi-colon on the last line of data for either form.)

Form 2:

```
matdat gnp out(1-4) 1981.1
   1513    1512     1522
   950     949      956
   359     140      143
   359     361      361
   445     448      450;
```

Here, series names are given immediately after matdat with starting date given as the last argument on the matdat line. The first series reads from the first line after the matdat, the second series reads from the second line, and so on. Note also that group definitions are allowed here.

matup [date]

Uses the same format as the matdat command, but updates series already in the bank.

To use the matdat or matup commands effectively, create an "add" file containing the data or matdat statements necessary to bring in the data. For example, to transfer data from a spreadsheet where the data already exist in columns, use the following steps.

Print the spreadsheet data to a file. (In LOTUS, you would print the data range to a file, creating a PRN file.) Add the necessary *G7* data commands to the file just created using an the editor. (Alternatively, include the commands in the spreadsheet before you print the range.) If the column data do not include dates, use Form 2 of the matdat command.

From the *G7* console, use the add command to bring in the data stored in the file created by STEP 1.

```
add filename_created_in_STEP_1
```

Speeding Processing of Long Data Files

When adding long data files to *G7*, two commands can accelerate considerably the processing of the data. Include these commands as part of your add file to introduce data, but only after you are sure the add file is trouble free.

(addt)ype n

Normally, *G7* writes to the screen all of the input file when an add is underway. Turning screen writing off with this command speeds up the operation greatly.

Turn screen writing back on again at the end with

```
addt y
```

(wri)ndex <y | n>

Normally *G7* writes the entire index file of the workspace as each variable is added. When many series are added, this feature lengthens processing time. To defer writing the index until the entire add file is processed, "wri n" turns off writing and "wri y" turns it on again and writes the index. A 'q' with writing off will cause the index to be written before quitting, so there is no danger of losing everything by forgetting "wri y".

2.3.10 Writing ASCII Data From *G7*

Here are some *G7* commands commonly used for writing data to ASCII output files. These complement specialized routines for printing data in other formats, such as Excel spreadsheets.

tdates <date1> <date2>

Sets the dates for subsequent typing commands without actually printing out any data.

(ty)pe <series_name> [<date1>] [<date2>]
(ty)pe (<expression>) [<date1>] [date2>]

Displays values for the named series from <date1> to <date2> on the screen. The values either may be a data series or an expression. If a legal expression is presented between parentheses, then the value is calculated and displayed. The first number on each line is the date of the first observation on the line. Dates may be omitted if they are unchanged from the previous *type* command. If the *save* command (see below) is on, the displayed values are transferred to the save file where they are preceeded with an *(up)date* command.

(sty)pe <series_name> [date1] [date2]

"Silent" type writes the series to the currently open "save file" without displaying data on the screen. This can speed processing a long add file.

matty [file <filename>] [<date1> [<date2>]] [<dump>]
<var1> [var2] . . . [var100]

The command's name abbreviates "matrix type." It displays time series data for up to 100 variables (var1, var2, . . .) in vertical columns, with time running down the page. Data are shown for the range of dates specified by the preceding *tdates* commands.

For example, with the Quip data base assigned, the commands

```
tdates 1990.1 1991.4
matty gdp c v fe fi g
```

shows the results:

```
Date         gdp        c          v         fe         fi         g
1990.100    5660.6    3759.2    822.7     541.6     615.9     1153.0
1990.200    5750.8    3811.8    835.0     554.8     615.1     1164.3
1990.300    5782.2    3879.2    804.7     555.5     634.1     1176.9
1990.400    5781.7    3907.0    736.3     577.3     649.2     1210.4
1991.100    5821.9    3910.7    723.5     577.4     610.3     1220.6
1991.200    5892.5    3961.0    716.4     602.7     615.0     1227.4
1991.300    5950.2    4001.6    744.1     602.6     624.5     1226.5
1991.400    6002.1    4027.1    760.7     624.4     639.3     1229.2
```

If the first word after the command is "file", then the next word is the name of a file to which the results will also be written. Example:

```
matty file nipa.dat gdp 1 c 2 cd  v 1 vfnre 2 vfnrs
```

The results are shown on the blue Results screen. If saving is on (as a result of a "save <filename>" command explained below), the results also go into the save file. If the optional "dump" keyword is provided, then the output is more compact; this is useful for creating spreadsheet-like text files.

gridty [file <filename>] [<date1> [<date2>]] <var1> [var2] . . . [var100]

The grid display presents data in a spreadsheet window. This is convenient for scrolling about to look at the data. One can also copy data from the grid display to the Windows clipboard. This feature is very similar to the *show* command for displaying vectors and matrices; see that documentation for more details.

save <filename> [<type>]
save off

Opens a file of the given name to receive output produced by *type*, *stype*, *matty*, *regression*, and other commands. Most commands executed by *G7* are stored in the "save" file. A save file is closed and saving terminated with the "off" option.

The optional <type> argument is the command that will be printed when using *ty* or *sty* commands. The default is "update", but it may also be "data", "vdata", "ovr", "ind", "cta", "gro", "vupdate", or "dump".

ic [<text>]

The "InterDyme Comment" routine prints text to the *G7* output window. If "save" is on, then the text is stored in the text file. In the file, a "#" is inserted before the string is printed. These lines are recognized by *G7*, *Build*, and *IdBuild* as comments.

text [<text>]

The text command nearly is identical to the *ic* command. It prints text to the *G7* output window. If save is on, then the text is stored in the text file. In contrast to the *ic* command, no "#" is inserted before the string is printed.

ts

The "time stamp" command prints the time and date to the *G7* output window. If save is on, then the time and date is stored in the text file as a comment.

format <width> [<decs> [<obsPerLine>]]
format off

This command is used to override the default settings of the *ty* and *sty* commands for the width of the data, the number of decimal points, and the number of observations printed on each line. The <width> and <decs> arguments also apply to the *matty* or *matpr* command. If you give the "format off" command, it will return *G7* to its default style of formatting. Without using this command, *G7* decides on the format to use based on the absolute value of selected elements of the series. This can sometimes result in a messy .SAV file if all data is printed at different widths and decimal points.

2.3.11 How To Read and Write in Excel Format with *G7*

G7 can read, create, and print spreadsheet files in the Microsoft Excel format. The interface allows data to be read from Excel files and written to the *G7* workspace data bank or to a VAM file. Also, text may be read and then recorded using the *G7 catch* command for later processing or it can be stored as strings. Data can be written from *G7* databanks to new or existing Excel files. Microsoft Excel 2000 or later must be installed to use of these tools. Most routines have been tested with Excel 2000, 2003, 2007, and 2010, both on Windows XP, Vista, and Windows 7 platforms.

File Management

xl open
xl open workbook [<filename>]
xl open worksheet <name>
xl open chart <name>

An *xl open* command with no arguments launches the Excel server. An *open workbook* command may provide the name of an existing Excel file. If no name is provided, a new workbook will be created. The option "workbook" may be abbreviated "wb".

An *xl open worksheet* command also can open a worksheet or chart within the Excel file for reading or writing. The number to be used is the order of the worksheets within the file, where the first tab is '1'. The list of charts is handled separately, and the first chart also begins with '1'. Alternatively, the name of the worksheet or chart sheet may be specified. "worksheet" may be abbreviated "ws" and "chart" may be abbreviated "ch".

Examples:

```
xl open
xl open workbook c:\test\demo1\xltest.xls
xl open worksheet 1
```

xl create
xl create workbook [<filename> [<filetype>]]
xl create worksheet [before|after] [<name>]
xl create chart [before|after] [<name>]

An *xl create* command with no arguments launches the Excel server and opens a new workbook with one worksheet.

An *xl create workbook* command may provide the name for a new Excel filename. If no name is provided, a new workbook will be created with the default name. The option "workbook" may be abbreviated "wb".

The default filetype is XLS. Available file types include

File Types	
AddIn (.xlam)	WorkbookNormal (.xls)
CSV (.csv)	SYLK (.slk)
CSVMac (.csv)	Template (.xltx)
CSVMSDOS (.csv)	TextMac (.txt)
CSVWindows (.csv)	TextMSDOS (.txt)
DBF2 (.dbf)	TextPrinter (.txt)
DBF3 (.dbf)	TextWindows (.txt)
DBF4 (.dbf)	WK1 (.wk1)
DIF (.dif)	WK1ALL (.wk1)
Excel2 (.xls)	WK1FMT (.wk1)
Excel2FarEast (.xls)	WK3 (.wk3)
Excel3 (.xls)	WK4 (.wk4)
Excel4 (.xls)	WK3FM3 (.wk3)
Excel5 (.xls)	WKS (.wks)
Excel7 (.xls)	WQ1 (.wq1)
Excel9795 (.xls)	UnicodeText (.txt)
Excel4Workbook (.xls)	Html (.html)
IntlAddIn (.xla)	XLS (.xls)
IntlMacro (.xlsm)	

If no extension is provided with the filename, then the appropriate extension will be determined according to the file type and will be appended to the file name.

An *xl create worksheet* command may provide a name for a new worksheet to be added to the open workbook. The instruction "worksheet" may be replaced with "ws". An *xl create chart* command may provide a name for a new chart sheet to be added to the open workbook. The instruction "chart" may be replaced with "ch". If "before" or "after" is specified, then the new sheet will be inserted to the left or right of the currently-active sheet. See also the *xl name* command.

Examples:

```
xl create
xl create workbook NewData.xls
xl create worksheet "Data"
```

xl save [<namefile> [filetype]]

A *save* command must provide a name for the open workbook, if the workbook has not already been named.

If a file type is specified that is different than the current setting, then the spreadsheet will be saved as the new file type. The default file type is XLS. Other available file types include

File Types	
AddIn (.xlam)	WorkbookNormal (.xls)
CSV (.csv)	SYLK (.slk)
CSVMac (.csv)	Template (.xltx)
CSVMSDOS (.csv)	TextMac (.txt)
CSVWindows (.csv)	TextMSDOS (.txt)
DBF2 (.dbf)	TextPrinter (.txt)
DBF3 (.dbf)	TextWindows (.txt)
DBF4 (.dbf)	WK1 (.wk1)
DIF (.dif)	WK1ALL (.wk1)
Excel2 (.xls)	WK1FMT (.wk1)
Excel2FarEast (.xls)	WK3 (.wk3)
Excel3 (.xls)	WK4 (.wk4)
Excel4 (.xls)	WK3FM3 (.wk3)
Excel5 (.xls)	WKS (.wks)
Excel7 (.xls)	WQ1 (.wq1)
Excel9795 (.xls)	UnicodeText (.txt)
Excel4Workbook (.xls)	Html (.html)
IntlAddIn (.xla)	XLS (.xls)
IntlMacro (.xlsm)	

If no extension is provided with the filename, then the appropriate extension will be determined according to the file type and will be appended to the file name.

Examples:

xl “Data.xls”

xl close

This command closes an open workbook. It does not shut down the Excel session that is running in the background.

Examples:

```
xl close
```

xl exit

This command closes an open workbook and disconnects *G7* from the Excel server that is running in the background. If the Excel server was started by *G7*, then it will be stopped. Otherwise, the Excel server may continue running in the background; it can be closed by opening the Task Manager, selecting “Excel,” and terminating the process.

Examples:

```
xl exit
```

Reading Routines

xl read < column > < row > < value >

xl read < column > < row > < direction > < series > [< start > <end>]

This command reads text or data from an open Excel worksheet, where:

Column

May be given as a number or as letters (e.g. B or 2, AF or 32).

Row
The spreadsheet row number.

Value
To read a text entry, provide a set of quotes (“”). The value is displayed on the *G7* output window. If catch is on, then the value also is stored as text.

Direction
Either d(own), r(ight), l(eft), or u(p), starting with the cell specified with the row and column entry. NOTE: this is the direction indexed by time, so that if if data for a particular series are written across a row, then “right” should be specified.

Series
A *G7* data series. Bank letters are allowed, so the series may be written to an element of a vector or matrix in the default Vam bank. If no letter is provided, then the data is recorded in the workspace bank.

Start
The starting date, where the date is in *G7* format. If dates are not provided, the desired dates are assumed to be those of the current *fdates* setting.

End
The ending date, where the date is in *G7* format.

Examples:

```
xl read  E 3 ""
xl read AF 1 right a.y
xl read  1 1 down  gdp 1976 2010
```

xl mkseries <frequency> [“text1”] <column> <row> [“text2”]

The *mkseries* command forms a series in the workspace with the name in the specified column and row position. The desired frequency must be given; typically, this will be 1 for annual, 4 for quarterly, or 12 for monthly data.

Optional additional text may be entered, such as xl mks 12 “frs” A 1 “old”. Note that the text must be in quotes. Suppose that cell A1 contains the sector number 10. Then the command will create a variable with monthly frequency called “frs10old”. Note that the *G7* string routines, together with the *%xls* keyword, provide far more flexibility than is allowed by the *xl mkseries* routine.

The created name is stored in the *G7* workspace and may be used in the next *xl read* command; simply give the *xl read* command without specifying a series name.

Examples:

```
xl mkseries 12 A 1
xl mks 12 "frs" A 1 "old"
```

xl vecread <c (cols)> <r(rows)> <direction> <vector> <v (index)> [<start> [end]]

This command reads vector data from an open Excel worksheet, where:

Cols
Columns in Excel file for first year of data. May be given as letters or as numbers (e.g. B or 2, AF or 32).

Rows
Rows in Excel file for first year of data.

Direction
Either d(own), r(ight), l(eft), or u(p), starting with the cell specified with the row and

column entry. Note: this is the direction indexed by time, so that if if data for a particular vector element are written across a row, then “right” should be specified.

Vector
A *G7* vector. Bank letters are allowed to specify the vam bank.

Index
A list of vector elements.

Start
The starting date, where the date is in *G7* format. If dates are not provided, the desired dates are assumed to be those of the current fdates setting.

End
The ending date, where the date is in *G7* format. If neither a start date nor an end date is given, the end date is set according to the current *fdates*. If only a start date is given, then the end date is set equal to the start date so that only one period of data is read.

Examples:

```
xl vecread c(c-d f-g) r(35) down c.gdpN v(1-4) 1996 2000
```

xl matread <XL cols><XL rows><matrix><Vam cols><Vam rows><Period>
This command reads data from an open Excel worksheet and records it as a Vam matrix, where:

XL Columns
A listing of spreadsheet columns to be read. May be given as letters or numbers e.g. (1-2, 5, 26-28) or (A-B, E, Z-AB).

XL Rows
A listing of spreadsheet rows to be read. Must be given as numbers, e.g. (1-2, 5, 8).

Matrix
The Vam matrix that will be used to store the data.

Vam Columns
A listing of columns to be written in the Vam file matrix. Must be given as numbers, e.g. (1-2, 5, 8). Must have the same number of elements as are read.

Vam Rows
A listing of rows to be written in the Vam file matrix. Must be given as numbers, e.g. (1-2, 5, 8). Must have the same number of elements as are read.

Period
The date in the *G7* date format.

Examples:

```
xl matread c(A-D, F) r(1-20) mat c(1-5) r(10-30) 2000
xl matread c(1-4, 6) r(1-20) AM c(1-5) r(10-30) 2000
```

Writing Routines

G7 can create workbooks and worksheets and fill them with text and numerical data. Note that the display of numerical data is controlled by the precision set by the *G7* format format command. The syntax for the writing routines is presented below. Fonts and other features of the document maybe controlled; these capabilities are described in the last section.

xl write < column > < row > “< value >”

xl write < column > < row > < direction > [< series >] [< start > [end]] This command writes text or data to an open Excel worksheet, where

Column
The spreadsheet column position. May be given as a number or as letters (e.g. B or 2, AF or 32).

Row
The spreadsheet row number.

Value
A string of text provided on a single line. Must be surrounded with quotes.

Direction
Either d(own), r(ight), l(eft), or u(p), starting with the cell specified with the row and column entry. NOTE: this is the direction indexed by time, so that if data for a particular series are to be written across a row, then "right" should be specified.

Series
A *G7* data series. Bank letters are allowed, so the series may be taken from the workspace bank, a macro bank, or an element of a vector or matrix. If no series is specified, then a series of dates will be printed using the Excel date format.

Start
The starting date, where the date is in *G7* format. If dates are not provided, the desired dates are assumed to be those of the current *tdates* setting.

End
The ending date, where the date is in *G7* format.

Examples:

```
xl write  E 3 "Year"
xl write AF 1 right a.y
xl write  1 1 down  gdp 1976 2010
```

xl vecwrite <vector> < v(index) > < c(cols) > < r(rows) > < direction > [< start > [end]]
G7 can print entire vectors of data for multiple years using a single command. The *vecwrite* command must be used with a VAM bank. This command writes text or data to an open Excel worksheet, where:

Vector
The root name of a vector stored in a vam bank. Bank letters are allowed.

Index
The range of vector elements that are to be printed.

Cols
The spreadsheet columns for the first period of data, given as a group of letters or numbers.

Rows
The spreadsheet rows for the first period of data, given as a group of numbers. Either cols or rows must have a single element, and the other must have the same number of elements as contained in index.

direction
Either d(own) or r(ight), starting with the row or column specified with the cols and row entries. NOTE: this is the direction indexed by time, so that if data for a particular series are to be written across a row, then "right" should be specified.

Start

The starting date, where the date is in *G7* format. If dates are not provided, the desired dates are assumed to be those of the current *tdates* setting.

End

The ending date, where the date is in *G7* format.

Examples:

```
xl vecwrite a.output v(1-10) c(B) r(5-14) right 2008 2010
```

xl formula <column> <row> "< formula >"

Insert an Excel formula in the specified cell.

Examples:

```
xl formula A 5 "=sum(A1:A4)"
```

xl graph <row 1><column1>...<row N><column N> <direction> <start date><end date> <graph style>

xl graph title ["<title>" [font <style>]]

Create a graph sheet in the current workbook. The graph will appear in a sheet to the left of the active worksheet. Specify the first cell of each series to be graphed. Each series must extend in the same direction, either to the right or down the sheet. Each series should have the same number of observations. The first series will appear on the horizontal axis. Current *G7* settings for title, subtitle, and vertical axis title will be employed.

The command also may be used to recover the title of an existing graph; if no arguments are given, then the graph title subsequently is available using the %xls keyword. If a title is given, then the title will be added to the active chart; the title must be surrounded by quotation marks. If the title is followed by the "font" keyword, then font options may be specified including color, typeface, size, single or double underline, bold, and italic.

Available graph styles include

Graph Styles	
ColumnClustered	Bubble3DEffect
ColumnStacked	StockHLC
ColumnStacked100	StockOHLC
3DColumnClustered	StockVHLC
3DColumnStacked	StockVOHLC
3DColumnStacked100	CylinderColClustered
BarClustered	CylinderColStacked
BarStacked	CylinderColStacked100
BarStacked100	CylinderBarClustered
3DBarClustered	CylinderBarStacked
3DBarStacked	CylinderBarStacked100
3DBarStacked100	CylinderCol
LineStacked	ConeColClustered
LineStacked100	ConeColStacked
LineMarkers	ConeColStacked100
LineMarkersStacked	ConeBarClustered
LineMarkersStacked100	ConeBarStacked

continues on next page

Table 1 – continued from previous page

PieOfPie	ConeBarStacked100
PieExploded	ConeCol
3DPieExploded	PyramidColClustered
BarOfPie	PyramidColStacked
XYScatterSmooth	PyramidColStacked100
XYScatterSmoothNoMarkers	PyramidBarClustered
XYScatterLines	PyramidBarStacked
XYScatterLinesNoMarkers	PyramidBarStacked100
AreaStacked	PyramidCol
AreaStacked100	3DColumn
3DAreaStacked	Line
3DAreaStacked100	3DLine
DoughnutExploded	3DPie
RadarMarkers	Pie
RadarFilled	XYScatter
Surface	3DArea
SurfaceWireframe	Area
SurfaceTopView	Doughnut
SurfaceTopViewWireframe	Radar
Bubble	

xl border <FC> <FR> <LC> <LR> <option 1> [<option2>[. . .]]

Set the cell border, where the cell range is given by <first column>, <first row>, <last column>, and <last row>, and options are

off

remove border.

color

border color, chosen from the list of Excel colors.

weight

border weight, chosen from hairline, thin, medium, or thick.

position

border position, chosen from border[default], left, right, top, bottom, horizontal, vertical, diagonalup, or diagonaldown.

linestyle

border linestyle, chosen from continuous, dash, dashdot, dashdotdot, dot, double, or slantdashdot.

xl printer <option 1> [<option2>[. . .]]

Set print options, where the options are

orientation

set the page orientation to landscape or portrait.

area

set the print area with "area <fr><fc><lr><lc>", or turn off the print area with "off".

print

print the current page.

xl cf <FC> <FR> <LC> <LR> <number of conditions>

<type1> <operator> <condition1> [<operator> <condition2>] [font <options>] [border <options>] [background <options>]

[<type2> <operator> <condition1> [<operator> <condition2>] [font <options>] [border <options>] [background <options>]]

[<type3> <operator> <condition1> [<operator> <condition2>] [font <options>] [border <options>] [background <options>]]

Set conditional formatting for the specified range of cells. Up to three conditions may be specified, and so the number of conditions and the number of specification lines must be between 1 and 3.

Each row of conditions must begin with "value"; "formula" will be offered later. Operators must be <=, <, ==, !=, >, or >=. If a second condition is specified and the first operator is < or <=, then the second operator must be > or >= (not between), or if the first operator is > or >=, then the second operator must be < or <= (between); otherwise, any second operator and condition will be ignored. A condition may be a number, string, or cell address. Available format settings include font and background; the "font" or "background" keyword must preceed the format specification, where specifications may be chosen from the relevant list. Font options include color, bold, italic, and single or double underline; border is not implemented yet; background options include color. The number of conditions must match the number of rows.

Miscellaneous Routines

xl gridlines [<on|off>] [<color>]

Turn on or off the display of gridlines on the selected worksheet. A color may be specified from the list of Excel colors.

xl missing [<symbol>]

This sets missing value symbols for the spreadsheet. When *G7* is reading the spreadsheet file, a "missing value" entry is recorded in the *G7* data bank for any spreadsheet cell containing this symbol. The symbol may be a word, number, or a string, where strings must be specified in quotes in the *xl missing* command. Up to 10 missing value codes may be stored, but each must be entered separately. If the command is given without arguments, then previous entries are reported. The following example allows *G7* to recognize 0.0, NA, and _N/A_ as missing value codes when they are read from a spreadsheet file. See also the replace command.

Examples:

```
xl missing 0.0
xl missing NA
xl missing _N/A_
```

xl clear missing

This command clears missing value codes specified in *xl missing*. It also resets the replacement value to the default setting, where replacement values are specified with the *xl replace* command.

Examples:

```
xl clear missing
```

xl replace <value>

This command sets a replacement value for missing values in the spreadsheet file. If *G7* reads a cell that matches an entry set with *xl missing*, then the value is replaced by <value>. By default, this replacement value is the *G7* missing value code (-0.0000001). <value> must be a number. The following example replaces missing value codes read from a spreadsheet with zeros in the *G7* data bank. See also the *xl missing* command.

Examples:

```
xl replace 0.0
```

xl print missing “<value>”

This command provides a character or string to indicate a missing value when *G7* writes data to a spreadsheet.

Examples:

```
xl print missing "N/A"
```

xl visible

This command makes visible the Excel program, provided that Excel is running. Making Excel visible may decrease the execution speed of your script. By default, the Excel window is not visible when *G7* launches the Excel server and opens a spreadsheet file.

Examples:

```
xl visible
```

xl invisible

This command makes invisible the Excel program, provided that Excel is running. Making the display invisible might increase the execution speed of your script.

Examples:

```
xl invisible
```

xl name worksheet <sheetname>

This command names the worksheet that currently is open.

Examples:

```
xl name ws "Data"
```

xl column width <column> <width>

This command sets the column column in the selected worksheet to width width.

Examples:

```
xl column width A 30
```

xl row height <row> <height>

This command controls the specified row. Current controls include specification of the row height, given as an scalar.

Formatting Controls for Printing Data

G7 can create formatted workbooks and worksheets. Note that the display of numerical data is controlled by the precision set by the *G7* *format* command. Other control over fonts, column widths, and so on are described here.

xl setfont <settings>
xl font <column1> <row1> <column2> <row2> <settings>

The *setfont* routine sets the font for all subsequent printing. The *font* command sets the font for the specified range of cells. The <settings> are a listing of one or more font settings; these may be provided in any order.

clear
Clear the fonts set by the *setfont* command.

bold
Set text to bold face.

italic
Set text to italic.

underline
See the list of underlining options.

justify|center|right|left
Horizontal alignment.

top|vcenter|bottom|vjustify"
Vertical alignment.

color
See the list of available text colors.

typeface
See the list of available type faces.

X or size<X>
Set the font size to integer X.

Examples:

```
xl setfont bold italic ~ right top "red" "arial" size14
xl font A 1 D 10 "Courier New" 10 "blue"
```

textwrap [off]
Turn text wrapping on or off for the specified cells.

format(<format>|off)
This setting allows specification of general, number, currency, accounting, date, short date, long date, time, percentage, fraction, scientific, text, off, or custom settings. Complex and multiple word settings must be wrapped in "". The "off" and custom settings must use the "format(<>)" syntax; other settings may be stated simply as "general", "number", or otherwise.

separator("<separator>"|off]
Turn on or off the use of a separator for numeric cells. A custom separator may be specified, or the "("<separator>")" argument may be omitted to employ the default separator. The setting will be stored, but it only will be applied to cells if a format specification has been or will be given. Precision still is controlled by the G7 format command.

Underline

Underlining may be specified in several ways.

underline [-]
UnderlineSingle
-

Single underlining.

underline =
UnderlineDouble
=

Double underlining.

underline _
UnderlineSingleAccounting
_

Single accounting underlining.

underline [~]
UnderlineDoubleAccounting
~

Double accounting underlining.

underline n
UnderlineNone

No underlining.

Available font colors include:

Font Colors	
"None"	"Medium Gray"
"Aqua"	"Mint green"
"Black"	"Navy blue"
"Blue"	"Olive green"
"Cream"	"Purple"
"Dark Gray"	"Red"
"Fuchsia"	"Silver"
"Gray"	"Sky blue"
"Green"	"Teal"
"Lime green"	"White"
"Light Gray"	"Yellow"
"Maroon"	

System fonts may be specified by employing the "TF(<system_font)" function (or "typeface(<system_font>)") in <settings>. Multiple-word font names must be surrounded by quotation marks. For example, if the Adobe Garamond Pro font is installed, then it may be specified in the *xl font* command as "TF("Adobe Garamond Pro")". Available standard font types include:

Font Types	
"Agency FB"	"Gill Sans Ultra Bold Condensed"
"Agency FB Bold"	"Gloucester MT Extra Condensed"

continues on next page

Table 2 – continued from previous page

“Algerian”	“Goudy Old Style”
“Arial”	“Goudy Old Style Bold”
“Arial Black”	“Goudy Old Style Italic”
“Arial Black Italic”	“Goudy Stout”
“Arial Bold”	“Haettenschweiler”
“Arial Bold Italic”	“Harlow Solid Italic”
“Arial Italic”	“Harrington”
“Arial Narrow”	“High Tower Text”
“Arial Narrow Bold”	“High Tower Text Italic”
“Arial Narrow Bold Italic”	“Impact”
“Arial Narrow Italic”	“Imprint MT Shadow”
“Arial Rounded MT Bold”	“Informal Roman”
“Arial Unicode MS”	“Jokerman”
“Baskerville Old Face”	“Juice ITC”
“Batang”	“Kristen ITC”
“Bauhaus 93”	“Kunstler Script”
“Bell MT”	“Lucida Bright”
“Bell MT Bold”	“Lucida Bright Demibold”
“Bell MT Italic”	“Lucida Bright Demibold Italic”
“Berlin Sans FB”	“Lucida Bright Italic”
“Berlin Sans FB Bold”	“Lucida Calligraphy Italic”
“Berlin Sans FB Demi Bold”	“Lucida Fax Demibold”
“Bernard MT Condensed”	“Lucida Fax Demibold Italic”
“Blackadder ITC”	“Lucida Fax Italic”
“Bodoni MT”	“Lucida Fax Regular”
“Bodoni MT Black”	“Lucida Handwriting Italic”
“Bodoni MT Black Italic”	“Lucida Sans Demibold Italic”
“Bodoni MT Bold”	“Lucida Sans Demibold Roman”
“Bodoni MT Bold Italic”	“Lucida Sans Italic”
“Bodoni MT Condensed”	“Lucida Sans Regular”
“Bodoni MT Condensed Bold”	“Lucida Sans Typewriter Bold”
“Bodoni MT Condensed Bold Italic”	“Lucida Sans Typewriter Bold Oblique”
“Bodoni MT Condensed Italic”	“Lucida Sans Typewriter Oblique”
“Bodoni MT Italic”	“Lucida Sans Typewriter Regular”
“Bodoni MT Poster Compressed”	“Magneto Bold”
“Book Antiqua”	“Maiandra GD”
“Book Antiqua Bold”	“Map Symbols”
“Book Antiqua Bold Italic”	“Matura MT Script Capitals”
“Book Antiqua Italic”	“Mistral”
“Bookman Old Style”	“Modern No. 20”
“Bookman Old Style Bold”	“Monotype Corsiva”
“Bookman Old Style Bold Italic”	“MS Mincho”
“Bookman Old Style Italic”	“MS Outlook”
“Bradley Hand ITC”	“MT Extra”
“Britannic Bold”	“Niagara Engraved”
“Broadway”	“Niagara Solid”
“Brush Script MT Italic”	“OCR A Extended”
“Californian FB”	“Old English Text MT”
“Californian FB Bold”	“Onyx”
“Californian FB Italic”	“Palace Script MT”
“Calisto MT”	“Palatino Linotype”
“Calisto MT Bold”	“Palatino Linotype Bold”

continues on next page

Table 2 – continued from previous page

“Calisto MT Bold Italic”	“Palatino Linotype Bold Italic”
“Calisto MT Italic”	“Palatino Linotype Italic”
“Castellar”	“Papyrus”
“Centaur”	“Parchment”
“Century”	“Perpetua”
“Century Gothic”	“Perpetua Bold”
“Century Gothic Bold”	“Perpetua Bold Italic”
“Century Gothic Bold Italic”	“Perpetua Italic”
“Century Gothic Italic”	“Perpetua Titling MT Bold”
“Century Schoolbook”	“Perpetua Titling MT Light”
“Century Schoolbook Bold”	“Playbill”
“Century Schoolbook Bold Italic”	“PMingLiU”
“Century Schoolbook Italic”	“Poor Richard”
“Chiller”	“Pristina”
“Colonna MT”	“Rage Italic”
“Comic Sans MS”	“Ravie”
“Comic Sans MS Bold”	“Rockwell”
“Cooper Black”	“Rockwell Bold”
“Copperplate Gothic Bold”	“Rockwell Bold Italic”
“Copperplate Gothic Light”	“Rockwell Condensed”
“Courier New”	“Rockwell Condensed Bold”
“Courier New Bold”	“Rockwell Extra Bold”
“Courier New Bold Italic”	“Rockwell Italic”
“Courier New Italic”	“Script MT Bold”
“Curlz MT”	“Showcard Gothic”
“Edwardian Script ITC”	“SimSun”
“Elephant”	“Snap ITC”
“Elephant Italic”	“Stencil”
“Engravers MT”	“Symbol”
“Eras Bold ITC”	“Tahoma”
“Eras Demi ITC”	“Tahoma Bold”
“Eras Light ITC”	“Tempus Sans ITC”
“Eras Medium ITC”	“Times”
“Felix Titling”	“Times New Roman”
“Footlight MT Light”	“Times New Roman Bold”
“Forte”	“Times New Roman Bold Italic”
“Franklin Gothic Book”	“Times New Roman Italic”
“Franklin Gothic Book Italic”	“Trebuchet MS”
“Franklin Gothic Demi”	“Trebuchet MS Bold”
“Franklin Gothic Demi Cond”	“Trebuchet MS Bold Italic”
“Franklin Gothic Demi Italic”	“Trebuchet MS Italic”
“Franklin Gothic Heavy”	“Tw Cen MT”
“Franklin Gothic Heavy Italic”	“Tw Cen MT Bold”
“Franklin Gothic Medium”	“Tw Cen MT Bold Italic”
“Franklin Gothic Medium Cond”	“Tw Cen MT Condensed”
“Franklin Gothic Medium Italic”	“Tw Cen MT Condensed Bold”
“Freestyle Script”	“Tw Cen MT Condensed Extra Bold”
“French Script MT”	“Tw Cen MT Italic”
“Garamond”	“Verdana”
“Garamond Bold”	“Verdana Bold”
“Garamond Italic”	“Verdana Bold Italic”
“Gigi”	“Verdana Italic”

continues on next page

Table 2 – continued from previous page

“Gill Sans MT”	“Viner Hand ITC”
“Gill Sans MT Bold”	“Vivaldi Italic”
“Gill Sans MT Bold Italic”	“Vladimir Script”
“Gill Sans MT Condensed”	“Wide Latin”
“Gill Sans MT Ext Condensed Bold”	“Wingdings”
“Gill Sans MT Italic”	“Wingdings 2”
“Gill Sans Ultra Bold”	“Wingdings 3”

2.3.12 Examples For Using the XL Commands

Example 1

```
vamcr vam.cfg hist
vam hist b
dvam b

# MAKE SAMPLE DATA
fdate 1975 2010
f t = 1974 + @cum(t, 1.0, 0.0)
f y = 1975 / (t - 1975)
f y{2010} = -0.0000001            # Set missing value in 2010

xl open xltest.xls               # Start Excel  server, open
                          # the xltest.xls workbook.
xl open worksheet 1             # Open worksheet 1.
xl write  A 1 "Writing text to file:"
                          # Write a string to A1.
xl open worksheet 2             # Open worksheet 2.
xl write A 1 "Record data columns"
xl write A 3 "Year"             # Record label for the date
xl write A 4 down t 1976 2010      # Record the date
xl write B 3 "y"               # Record series name
xl write B 4 down y 1976 2010      # Write series 'y' from the
                          # workspace to the worksheet,
                          # starting in cell B4 and
                          # moving down the sheet
xl close                     # Close the workbook.


## READ EXCEL FILE
xl open xltest.xls               # Open the workbook.
xl replace 0.0                  # Set replacement value for
                          # missing values codes to 0
XL missing "N/A"                   # Specify the code for
                          # missing values in the
                          # Excel file.
xl open worksheet 2             # Open worksheet 3.
xl read B 4 down b.y1 1976 2010     # Read data into vam bank b.
xl read F 4 right  y2 1976 2010     # Read data into workspace.
xl exit                      # Close workbook, close
                          # connection to Excel server.
```

Example 2

```
XL open BEA\section2all_xls.xls       # Start or attach to Excel
                                  # server, open workbook.
XL missing "....."                   # Replace missing values
do{
  XL open worksheet 2               # Open worksheet 2.
  XL read D %2 right a.inv_cst%1 1987 2004
                                  # Read data.
                                  # Store data in vam bank
  XL open worksheet 3               # Switch to worksheet 3.
  XL read D %2 right a.inv_qst%1 1987 2004
  }(1-66)(10 12 14 16-46 48 50 52 54-74 76 78-84 ) m
xl exit
```

Example 3 A demonstration of the ability of *G7* to read a matrix from an Excel file.

```
xl open C0319e.xls
xl invisible
xl open worksheet 1
xl replace 0.0
ic Read a full matrix from Excel, store in Vam matrix.
xl matread c(2-18) r(14-17, 19-20, 22-24,26-29,31-32, 34-35) c.AM c(1-17) r(1-17) 2000
xl exit
```

Do Loops in xl Commands The *xl* commands can be used in combination with the *do* loop command. The *do* command permits us to use loops, which saves time when writing code. These *do* loops, as in other programming languages, can be nested. Here is an example of reading data into *G7* using this technique:

Code without the do loop:

```
xl read C 3 right va1 1998 2007
xl read C 4 right va2 1998 2007
xl read C 5 right va3 1998 2007
```

Using the do loop:

```
do{
 xl read C %1 right va%2 1998 2007
 }(3-5)(1-3)m
```

In this case, *G7* starts to read the spreadsheet in cell C 3 and maps it into a series of workspace variables with root names “va”. It does this sequentially, matching the spreadsheet row number (%1) with the index number (%2). This is a powerful and versatile tool that can be used with many spreadsheet and other commands. For more information on this function, check the Do Loop section of this help file.

2.3.13 Writing Data To Lotus WK1

There are two related commands for writing series of data to Lotus WK1 command, either in row or in column format. A Lotus WK1 file can hold up to 256 columns and 4096 rows. Therefore, if you want to write many series, each with less than 256 observations, use *rp123*, which writes the series by row. If you have a few series with many (>256) observations, use *p123*, which writes by column.

These routines largely have been supplanted by the commands for printing Excel spreadsheets. Note that Microsoft Excel no longer supports spreadsheets in WK1 format. However, alternative spreadsheet software like OpenOffice continues to provide support. Note also that the WK1 standard severely limits the number of rows and columns that may be stored, and these restrictions have been relaxed in modern spreadsheet standards. Still, the following WK1 routines have several advantages over the *xl* commands. First, they do not require auxiliary software to be installed; recall that the *xl* commands require that Microsoft Excel is installed. Second, the routines print data considerably faster than the *xl* commands.

p123 <file_name> <date1> <date2> [width] [decs]
<ser1> [<ser2> <ser3> ... <sern>];
p123 <file_name> <date1> <date2> [width] [decs] [<ser1> <ser2> ... <sern>];

This command is similar to the *matty* command, except that the data are written directly to a worksheet. Do not include the WK1 extension with the filename; *G7* automatically appends the extension WK1. <date1> and <date2> are the desired starting and ending dates. Up to 239 series names can be entered in free format. End the list with a ';'. <width> and <decs> are optional; <width> specifies the display width within "Lotus" for all series transferred, and <decs> specifies the number of decimals to be displayed. Defaults settings are 9 and 3, respectively. *p123* worksheets are compatible with "Lotus", "Excel", "Quattro Pro", and many other programs.

Example 1:

```
p123 natacct 60.1 91.1 8 1
gnp c v vf vi fe fi g
```

Example 2:

```
p123 natacct 60.1 91.1 8 1 out(1-85)
```

rp123 <file_name> <date1> <date2> [width] [decs]
<ser1> [<ser2> <ser3> ... <sern>];
rp123 <file_name> <date1> <date2> [width] [decs] [<ser1> <ser2> ... <sern>];

This command works like *p123* to write series into a Lotus worksheet but, whereas *p123* writes the series as columns, *rp123* writes them as rows. With *p123*, one can have up to 255 series, each with up to 4096 observations. With *rp123*, one can have up to 4096 series, each with up to 255 observations. For many annual and quarterly data banks, *rp123* may prove the more efficient format.

2.3.14 Making Tables

The *Compare* program is used to make tables with the results of running the model or to present other data. As the name suggests, it is particularly adapted to comparing results of several runs of a model, but it can also list the contents of a data bank or the results of a single run of a model. When being used to show a base case and several alternatives, it can show the alternatives as actual values, as deviations from the base, or as percentage deviations from the base. The results can be written to a file that can be sent directly to a printer, or results can be printed as plain text, or spreadsheets can be created in Excel format. To use *Compare*, one first must prepare a stub file. Details are given below.

Compare is a companion program that is not part of the *G7* program. However, *Compare* can be run from the *G7* menu by selecting Model | Tables. Selection of this menu item opens a form that asks for the information necessary for *Compare* to run. Clicking OK then executes *Compare*. If you know that you want to run *Compare* with exactly the same entries on the form as the last time that *Compare* was run, you can skip the form by clicking Model | Express Tables.

Details on the Stub File

Here is an example of a stub file:

```
\xls
\dates 1995.0 1996.0 1997.0 1996.1 1996.2 1996.3 1996.4 1995-1996 1996-1997
\*
;
;THE AMI MODEL
;
&
gdpR        ;Gross national product
cR          ;Personal consumption
pibgR       ;Personal income w/o gov't
pidisR      ;Personal disposable income
taxrate\*100 ;taxrate\*100
gtnisR      ;Gov trans, net int, sub.
cpcR        ;Consumption per capita
pop\*1000    ;Population (in millions)
emp\*0.001   ;Employment
```

The first line indicates that the results should be printed in a spreadsheet document. The next line gives the dates of the periods which are to be printed. In a quarterly data bank or model, an annual date such as "1998.0" will print the annual average for the year. Dates shown with a hyphen will display the growth rate between the two periods shown. For instance, the above stub file will make a table with the annual growth rates between 1995 and 1996 and then 1996 and 1997. Alternatively, the first line may say "Ask dates"; *Compare* then will ask the user to provide the dates from the keyboard.

Compare has several editing features that are controlled by special characters at the beginning of a line. They are:

*****

Go to the top of a new page and print the titles of the alternatives runs as given in the .FIX file.

***m**

If there is not room for <m> more items on the page, go to a new page and print the dates across the top.

***m n**

If there is not room on the page for <m> more items plus <n> lines, go to a new page and print the names of the base and alternative runs.

;
: Print the line just as it stands.

&
: Print the dates across the page above the appropriate columns.

\fw dp pl tm bm tw
: where <fw> <dp> <pl> <tm> <bm> and <tw> all are numbers. This is an optional command to set the field width of each printed number to <fw>, the number of decimal places to be printed to <dp>, the page length to <pl> lines, the top margin to <tm> lines, the bottom margin to <bm> lines, and the width of the titles to <tw>. The last three may be omitted. The default values are "7 1 60 3 9 32".

\g
: where 'g' is the letter 'g'. Show the growth rates in percent for the variables named on the following lines. The growth rate will be for the time period between adjacent columns in the table.

\dates
: Do not use this if you have specified growth rates in your dates command using the hyphenated dates.

\n
: where 'n' is the letter 'n'. Cancel a previous \g.

\ar
: use if monthly or quarterly data is given at annual rates (the default). (ar = annual rates)

\pr
: use if monthly or quarterly data is given at monthly or quarterly rates (pr = period rates).

\xls
: Use to direct output to a spreadsheet document.

Many other commands that begin with the '\' character are available. A line beginning in any other way will be presumed to begin with a variable name, such as gnp$, or an expression. For the expressions, all the functions available in *G7* are also available in *Compare*. After the name or expression comes a ';' and after the ";" come up to thirty characters which will be printed at the left side of the line. Leading blanks – blanks between the ";" and the first visible letter on the line – will be printed and can be used to indent the titles.

Please see the Compare documentation for more details on using the software. Additional details are available elsewhere in these Help files.

2.3.15 Drawing Graphs

G7 can make a variety of graphs, which can then be printed, saved into Windows metafiles to be included in documents, or copied to the windows clipboard to be pasted into other applications. Here are the basic graph commands and related subsidiary commands. Detailed explanations follow.

title
: Provides the title for the graph.

subti
: Provides the subtitle.

vaxti
: Provides the vertical axis title.

vaxl
: Puts vertical axis labels inside or outside the rectangle of the graph.

graph
Graphs up to seven series on one graph with one scale.

mgr
Multi-scale graph. This is most useful with two series, since the left axis scale will be for the first series and the right axis scale for the second.

sgraph
Scatter graph. Graphs one series against another.

lgraph
Graphs variables that are in logarithms, while displaying the y axis in natural units

vrange
Controls the vertical range of graphs (the y-axis scale).

hrange
Controls the horizontal range for scatter graphs or the right-vertical range for multi-scale graphs.

legend
Controls printing of legend on graphs.

The characteristics of each line on the graph are set by selecting Graph|Settings from the main menu.

Graph Titles

(ti)tle <title of graph or regression>
Provides a main title that appears on subsequents graphs. Use *ti* with no following text string to remove the title. Note that the font for the title, subtitle, and other text may be set in the *G7* Graph | Font menu.

(subt)itle <text string>
Provides a subtitle on the subsequent graphs. Use *subti* with no following text string to remove the subtitle.

(vaxt)itle <text string>
Provides a vertically printed title for the y (left) axis. Use *vaxti* with no following text to remove the left axis title.

vaxl <in | out>
Causes the vertical axes numbers to be printed inside or outside the axes. The default is “vaxl in”.

Displaying Graphs

(gr)aph <name1> [<name2>] [<name3>] ... [<name7>] <date1> <date2> [date3]
(gr)aph (<name1>) [(<name2>)] [(<name3>)] ... [(<name7>)] <date1> <date2> [date3]
Constructs a standard Line Graph. Graph the named series from <date1> to <date2> or <date3>, if present. If <date3> is present, a vertical line (to separate history from forecast) will appear at <date2>. If dates have not changed since the last *graph* command, they need not be repeated. After a regression, the actual and predicted values may be plotted by *gr **. The actual data will be marked by squares and the predicted by plusses, unless the marks have been changed by a previous *line* command. Algebraic expressions may be provided in place of some or all of the series names, so long as the expressions are enclosed in parentheses.

Example:

```
title RTB -- The Treasury Bill Rate
gr rtb 70.1 85.1
```

(mg)raph <name1> [<name2>] [<name3>] . . . [<name7>] <date1> <date2> [date3]

(mg)raph (<name1>) [(<name2>)] [(<name3>)] . . . [(<name7>)] <date1> <date2> [date3]

Consruct a Multi-scale Graph. This is exactly like the *graph* command except that the series has its own vertical scale chosen so that its graph extends from the top to the bottom of the screen. The scale shown on the left is for the first series; the scale shown on the right is for the second. Algebraic expressions may be provided in place of some or all of the series names, so long as the expressions are enclosed in parentheses.

(lgr)aph <name1> [<name2>] [<name3>] . . . [<name7>] <date1> <date2> [date3]

(lgr)aph (<name1>) [(<name2>)] [(<name3>)] . . . [(<name7>)] <date1> <date2> [date3]

Construct a Semi-Logarithmic Graph. This is exactly like the *graph* command except that the series are expected to be in logarithms. The vertical scale will be marked in the natural units, not the units of the logarithms. If vertical range control is in effect (see below), the vertical ranges will be presumed to be in natural units. Algebraic expressions may be provided in place of some or all of the series names, so long as the expressions are enclosed in parentheses.

(sgr)aph <series 1> <series 2>

(sgr)aph (<series 1>) (<series 2>)

Construct a Scatter Graph. This will display a scatter diagram of <series 1> and <series 2>. The graph is created with <line 1> characteristics. Use the "0 width" option on the *line* command to plot only scatter points; otherwise the points will be connected. Each point on the two-dimensional graph measures the value of the first variable on the vertical axis, and the value of the second variable on the horizontal axis. For use only with the *sgraph* command, there is an *hrange* command that works just like the *vrange* command but it controls the horizontal axis. Algebraic expressions may be provided in place of either or both of the series names, so long as the expressions are enclosed in parentheses.

Style of Graph – Line, Range, Legend Control

line <number> <color> <width> <style> <mark> <fill> <left> <right>

The easy way to use the *line* command is interactively. Use Graph | Settings from the main menu. For use in command files, however, the following reference information is provided. The options are:

Number

The line number – 0 for the frame and annotation, 1-4 for the series graphed

Color

The following colors are available when graphing in *G7* – black, maroon, green, olive, navy, purple, teal, gray, red, lime, yellow, blue, fuchsia, aqua, ltgray, dkgray, white, lavender, brown, sienna, orange, tomato, coral, hotpink, azure, skyblue, gold, dkred, blood, ltred, dkblue, ltblue, dkgreen, ltgreen, seagreen, chartreuse, midnight, chocolate, slategray, forest, turquoise, and pumpkin. Alternatively, the user can enter a numbered color code from 1 to 63 for EGA and VGA monitors.

Width

The width of the line in pixels – 0 for no line (only marks or bars are printed). With a width above 1, all lines will be solid.

Mark
: Point markers – + for plus signs, x for x's, s for squares, d for diamonds, ^ for up-pointing triangles, v for down pointing triangles, > for arrows in direction of line, b for bars, f for bars without a rectangle around them. (Use *f* with 0 fill for the "low" line in "high-low" graphs.)

Style
: The line style – 0 for a solid line, 1 for a dashed line, 2 for a dotted line, 3 for a dashed line, 4 for dash dot, 5 for dash dot dot, 6 for none.

Fill
: Fill pattern for bars: 0 for Solid, 1 for Clear, 2 horizontal hatching, 3 for vertical hatching, 4 for forward diagonal, 5 for backward diagonal, 6 for cross hatching, and 7 for diagonal cross hatching.

Left
Right
: These values (between 0 and 1) specify where the left and right edges of the bar fall in the available space for each observation. The first value (left) must be smaller than the second value (right). Entering values of 0.2 and 0.8 for left and right, respectively, would yield a bar that started 20% across the column and end at 80%. Parallel bars are drawn by assigning non-overlapping intervals to different series. Stacked bars or high-low graphs are produced with overlapping intervals.

Vertical Range Control

(vr)ange <bottom> [top]
vr <bottom> [<line1> [<line2> [. . . [<line8>]]]] <top>
vr off

The graphing program normally sets the vertical range so the series extend from the bottom to the top of the graph. Sometimes, it is desirable to set the range independently. This is done with the *(vr)ange* command. For example "vr 0 100" will make all subsequent graphs have 0 at the bottom and 100 at the top until the "vr off" command is encountered, restoring *G7* to its normal mode. If the <top> is omitted, then only the bottom of the graph is set and the program finds the top so that the graph fits on the screen.

If one or more of the optional <line> entries are present, horizontal lines will be marked at those levels. The lighted pixels which mark these lines are located above the long marks on the horizontal axis, so theyalso serve as meaningful vertical lines.

Legend Control

(le)gend <a> [b]

Controls printing a legend at the bottom of a graph. 'a' equals

y
: for yes, prints the legend (default)

n
: for no, do not print the legend

s
: leave space for legend but do not print. Useful for allowing the user to annotate the legend.

y
for yes, mark the dates (default)

n
for no, do not mark dates. Useful for making bar graphs of data occurring at arbitrary intervals.

Background color

color <background>
The background color can be set interactively with Graph | Settings menu item or by the following .

Example:

```
color 4
```

Sets colors for graphs. The numbers may range from 0 to 7.

Graphing Distributed Lags For graphing distributed lags, it is helpful to specify the "dates" as coefficient numbers. Thus, to plot the coefficients a3 through a16 do "gr rcoef :3 16". The ":" is the signal to take the following numbers as regression coefficient numbers.

Setting Dates Without Actually Graphing

gdates <date1> <date2> [date3]
Sets the dates used by subsequent *graph* commands. With two dates provided, the series will be graphed from <date1> to <date2>. If a third date is given, the series will be graphed from <date1> to <date3>, with a vertical line drawn at <date2>.

gdates a
Selects "automatic" dates for *graph* commands. The automatic dates are the first and last date of the series actually present. The default setting in look is for automatic dates, unless specific dates have been previously specified. Automatic dates also adjust automatically to the frequency of the series. Not to be trusted when more than one series is being placed on the same graph.

Printing and Saving Graphs

gprint
This command prints the current graph to the printer. It is equivalent to picking Graph | Print from the main menu.

autoprint < y | n>
This command turns the autoprint flag on or off. It is off by default. When autoprint is set to 'y', then every graph that is displayed also is printed automatically.

gsave <filename>
This command is equivalent to picking Graph | Save from the main menu. It will create a Windows Metafile with the name given, followed by a .WMF extension. Do not supply the file extension on the command line.

Annotating Graphs The following commands allow text and arrows to be drawn on a graph using *G7* commands. Alternatively, the Graph | Annotation utility may be used to add these features interactively to a graph.

anntext <x-coordinate> <y-coordinate> <text to be inserted>
This command adds text to a graph. The x and y are values 0.000 to 1.000. The 'x' values start from the left side, where 0.250 would be 1/4 of the way from the left. The 'y' values start from the top of the graph, where 0.25 would be 1/4 of the way down from the top.

Example:

```
anntext 0.549 0.128 Hurricane Katrina
```

annline <starting x-coordinate> <starting y-coordinate> <ending x-coordinate> <ending y-coordinate> <symbol>

This command adds an arrow to a graph. 'x' and 'y' are values 0.000 to 1.000. The 'x' values start from the left side, where 0.250 would be 1/4 of the way from the left. The 'y' values start from the top of the graph, where 0.25 would be 1/4 of the way down from the top. <symbol> is a character added to the end of the line. Typically, the symbol is the '>' character that forms an arrow point.

Example:

```
annline  0.260  0.145  0.63  0.79 >
```

zip [off]

After the "zip" command has been given, the program will not pause after a graph has been created. It also does not calculate the "lever" variable after each regression. It particularly is useful for rapid re-estimation of an entire model when data has been updated. The command to turn off zip is "zip off".

Use of the Group Titles Feature It often is the case that you know the sector number of a series and you need to know the corresponding title. This is where the group titles features comes in handy. There are two commands that work together, *gtfile* and *gtitle*.

gtfile < titles file | "off" >

This command scans the specified titles file and stores the starting position of each line in the file. This makes it possible to use the *gtitles* command to supply a title from the titles file when given a number. To close the titles file or to load an alternative titles file, issue the command "gtitle off".

Example titles file:

```
1 "Agriculture, forestry, and fisheries"
2 "Metal mining"
3 "Coal mining"
4 "Natural gas extraction"
5 "Crude petroleum"
6 "Non-metallic mining"
```

gtitle <number>

Use of this command requires that a *gtfile* command has been given with a valid titles file as an an argument. The *gtitle* command then will use the n'th line of the titles file to put a title in the following *graph* command. These commands particularly are handy in combination with the *do* command or with an add file with group arguments. Since in many cases only sector numbers are being passed to such add files, the *gtitle* command allows you to specify only a sector number but still find the appropriate title for that sector.

2.3.16 Miscellaneous Commands and Other Information

Recovery From Catastrophe

If by some misfortune you are thrown out of *G7* against your will, you can continue right from where you were. Start *G7* again and recover the workspace of the previous run of *G7* by typing "wsb ws". If you then issue a "lis w" command you should see that you still have all of the variables that you created before *G7* died. Repeat any necessary *limits*, *mode*, *dates*, or other settings.

(q)uit

Quit, that is, close up shop in *G7*. This is equivalent to picking File | Exit from the menu.

time

Display the date and time.

dos
dos [options] <arguments>
dos [options] [batch file arguments] { ... }

The *dos* command passes instructions to the operating system. Three versions are offered, along with an important option.

To open a DOS window, simply enter dos with no arguments. *G7* pauses until the window is closed. The window may be closed by clicking the button in the upper right-hand corner or by typing “exit”.

To pass single commands to the system, type the desired system commands after *dos*. For example, to save the workspace bank as “newws”, enter the following:

```
dos copy ws.* newws.*
```

To pass a group of commands to the operating system, use the *dos{}* command. The brackets tell *G7* to create a temporary batch file containing the system commands between the brackets. The temporary file is destroyed automatically upon completion. Arguments may be passed to the batch file by listing them after dos and before the open bracket. Here is an example that again copies the workspace.

```
dos ws newws{
   rem Copying bank %%1 to %%2.
   copy %%1.ind %%2.ind
   copy %%1.bnk %%2.bnk
   }
```

Note that the syntax required for text within the *dos{}* command is the same syntax required elsewhere in *G7*. In particular, in order to pass a ‘%’ character to the system, “%%” must be specified in the *G7* script as is shown in the example above.

When the *dos* command is given with arguments (i.e. as in the second and third cases listed above), the output is captured that otherwise would appear in the DOS window. When the window closes, this output is displayed in the *G7* window. No output is displayed on the DOS window. While this often is acceptable and desirable, it is not suitable for running programs or commands that require user input. For this reason, an option is given to execute the *dos* command in interactive mode. Specify the option by entering “-i” (for interactive) immediately after *dos*. For example,

```
dos –i idbuild master
```

opens a command window and executes the program *IdBuild* with “master” as a command-line argument. This command is displayed in the DOS window, together with output from *IdBuild*. If *IdBuild* requires user input (for example if no configuration file is available), the user will see the prompt in the DOS window and can enter a response. The same option is available in batch mode.

```
dos -i ws newws{
   rem Copying bank %1 to %2.  Type <ctrl> c to cancel.
   pause
```

(continues on next page)

(continued from previous page)

```
   copy %1.ind %2.ind
   copy %1.bnk %2.bnk
   pause
   }
```

Multiple commands may be entered on a single line. The commands must be separated by a semicolon and enclosed with brackets. This capability works when there is only one line of *G7* input; everything, including the brackets, must be entered on the same line. In particular, this is useful for entering multiple DOS commands in the *G7* command box. The following command opens a DOS window, displays the contents of the directory one page at a time, and pauses at the end:

```
dos –i {dir /p; pause}
```

Note that the command interpreter also may offer the ability to enter more than one command per line. In Windows 2000, multiple commands may be separated with an ampersand (&). The above command may be entered as

```
dos –i dir /p & pause
```

findmode < m|a>

This command controls the search for a series when more than one databank is assigned. Normally, the mode is set to 'm', or manual. By default, the bank at position 'a' and the default vam file are searched. Series in any other banks may be searched by including the series name with the bank letter and a period. For example, if three banks are assigned, 'a', 'b', and 'c', and a series called "gnp" is in bank 'c', then you must specify "c.gnp". If *findmode* is set to 'a' (automatic), then *G7* will search sequentially through each bank in turn until it finds a series named "gdp".

vammode <s|a>

The vam mode may be set either to 's' (simple) or 'a' (advanced). In the simple mode, when a Vam file is assigned, only that file is assigned. The 'a' option is used by builders and users of *InterDyme* models, for which there is a G bank containing the macrovariables of the model in addition to the Vam bank that stores the vectors and matrices. Each time a new simulation is made, both a Vam file and a G bank are created with the same root name. When vammode is 'a' and a Vam file is assigned, *G7* looks to see if there is a G bank with the same name. If the G bank exists, then *G7* loads both banks as a single unit.

For example, if there is a vam file named HIOIL.VAM, with associated macro bank HIOIL.BNK, the following code will assign the two files as a unit:

```
vammode a
vam hioil a   # Load both hioil.bnk and hioil.vam
```

If the series "gnp" is in the G bank, you can view the macro series "gnp" with "ty a.gnp". If the Vam bank contains a vector output, then the vector can be shown with "show a.output". Note that both commands refer to bank 'a'. The user alternatively might want to load the banks separately. The following code will load the banks separately.

```
vammode s
vam  hioil a   # Load hioil.vam only
bank hioil b   # Load hioil.bnk
```

seed <integer>

The *seed* command allows the user to initialize the random number generator. The random number generator is used with the *@rand()* and *@normal()* functions. The *@rand()* function provides draws from the uniform (0,1) distribution and the *@normal()* function provides draws from the standard normal (0,1) distribution. If the *seed* command with identical arguments is employed between calls to *@rand()* or *@normal()*, then these random number generators should return identical series.

For example, the *G7* script

```
seed(20707)
f uni1 = @rand()
seed(20707)
f uni2 = @rand()
```

will produce series uni1 and uni2 that contain identical elements.

dir [arguments]

The *dir* command prints the contents of the working directory. <arguments> are legal DOS arguments for the *dir* command. Note, however, that any '#' characters are interpreted as the *G7* comment flag, and so any text after # are ignored.

For example, to display the listing of files beginning with 't', enter

```
dir t*.*  # This command displays file beginning with 't'
```

Note that the *dir* command does not accept the options permitted in the DOS *dir* command, such as "/w" or "/p". If such options are necessary, the following *G7* entries will allow them:

```
dos dir t*.* /Os
dos –i { dir t*.* /w & pause }
```

cd [path]

The *cd* command allows users to change the working directory. By default, *G7* looks to the current directory to read and write files.

pwd

The *pwd* command prints the path of the present working directory.

break

The *break* command terminates execution of an add file. If an add file calls a second add file, and the second contains the *break* command, then upon processing the *break* command *G7* will return to the first add file.

beep

This command typically is used to signal the end of a long script.

2.4 Regression in *G7*

2.4.1 Ordinary Regression

Here are some commands related to the running of an ordinary regression:

(ti)tle <the title you want for your regression or graph>

Supplies the title for the display of your regression results.

(lim)its <date1> <date2> [date3]

Specifies the starting date for the fit, <date1>, the ending date for the fit, <date2>, and the ending date for the test or forecast period, <date3>. If <date3> is omitted, then it is assumed to be the same as <date2>.

mode <test | forecast>

mode <t | f>

Determines what is to be done between <date2> and <date3>. The default mode is "test." In that case, the values predicted by the equation for the period from <date2> to <date3> are compared to the actual data and a SEE and MAPE are computed for the test period. For "forecast" mode to work, the series for all independent variables except lagged values of the independent variable must extend to <date3>. Then, in forecast mode, two forecasts are made for the period from <date2> to <date3>. The first simply is the predicted value; the second is the predicted value plus the "rho adjustment" to take account of the error in the last period of fit.

cmtype <yes | no>

If "yes", the simple correlation matrix and the standard deviation of each variable will be displayed in the results window before the regression results. After the regression, the variance-covariance matrix of the regression coefficients is displayed, as is the matrix of derivatives of the regression coefficients with respect to one another. For each coefficient, this matrix shows how all of the other coefficients would change if the given coefficient forcibly were changed by a *(con)straint* command. The default value for *cmtype* is "no."

See also the Regression Tests topic.

OLS Regression Command

r <y> = <x1>, <x2>, <x3>, . . . , <xn>

r <y> = ! <x1>, <x2>, <x3>, . . . , <xn>

Both forms regress the dependent variable <y> on the independent variables <x1>, . . . , <xn>. The first form automatically supplies a constant term, but the second does not. The total number of variables, including the constant, must not exceed 500. Independent variables may be expressions, as on the right side of an *f* command. Dependent variable must be a single variable. If the line ends with a ',' the command will continue on the next line.

If lagged values of the dependent variable occur among the explanatory variables, they automatically will be supplied from previously calculated values when forecasting. Regression coefficients and other values are stored in the "rcoef" variable in the workspace bank, the dependant variable is stored as series "depvar", predicted values are stored as the series "predic", and residuals are stored in series "resid."

The series "rcoef" is created by the regression command. The series begins in the first period of the workspace bank; this period is set as the base period in the G.CFG file. It is displayed with the wsinfo command. The following table displays the data stored in the "rcoef" series, for a regression on *k* independent variables.

ORDER OF THE RCOEF TIME SERIES
Number of coeficients (k)
Coefficient 1
….
Coeficient k
Mexval 1
….
Mexval k
Std. Error of Estimate (SEE)
R-Squared
Rho [(2-DW)/2]
Adjusted R-Squared
Mean Absolute Percentage Error (MAPE)/100
Number of observations (N)

Use *gr ** to graph regression results. When you have the graph of the regression and want to look back at the regression results, just move to the regression results window by navigating through the open windows of *G7*.

Use “gr rcoef :3 8” to graph regression coefficients 3 through 8. This especially is useful with distributed lags.

Use “gr resid” to graph the residuals. Use “gr lever” to graph the “leverage” series, which shows the derivative of the predicted value of each observation with respect to the observed value of that observation. It is useful for spotting outlying observations.

Terms Appearing on the Regression Display

These are characteristics of the entire regression. These appear at the top.

SEE
Standard error of estimate, or the square root of the average of the squares of the misses or “residuals” of the equation.

RSQ
(pronounced r-square) Coefficient of multiple determination.

RHO
(Greek rho) Autocorrelation coefficient of the residuals.

Obser
Number of observations.

SEE+1
The *SEE* for forecasts one period ahead using rho adjustment.

RBSQ
Coefficient of multiple determination adjusted for degrees of freedom.

DW
Durbin-Watson statistic; contains same information as does *RHO*.

DH
Durbin *H* statistic; replaces *DW* if a lagged value of the dependent variable is present.

DoFree
(Degrees of freedom) = *Obser* - number of independent variables.

From
To
The dates covered by the regression.

MAPE
Mean Absolute Percentage Error.

JarqBer
A test of normality due to Jarque and Bera.

Next are values relating to individual variables. In the Regression | Settings you may specify which of these are to be displayed. The default display is:

Name
Code name of the variable.

Reg-coef
Regression coefficient for this variable.

Mexval
(Marginal explanatory value) The percentage increase in *SEE* if this variable is left out of the regression. A factual alternative to the t statistic.

T-value
Student's t values.

Elas
Elasticity of the dependant variable with respect to this variable, evaluated at the means of both.

NorRes
Normalized residuals. The ratio of the sum of squares residuals after the introduction of this variable to the sum of squared residuals after all variables have been introduced. A factual alternative to F statistics.

Mean
The mean value of this variable.

Beta
What the regression coefficient would be if both the independent and dependent variables were scaled so that they had unit standard deviations.

Fstat
The Fisher F statistic for testing the significance of this and the following independent variables.

Additionally, if you give the command *cmtype y*, *G7* will display the correlation matrix of the original variables and their standard deviations, the matrix of variances and covariance of the regression coefficients, and a matrix of the derivatives of one regression coefficient with respect to another. The derivatives with respect to variable 3, for example, show the rate of change of each other coefficient if the coefficient on variable 3 is changed by a constraint. They show how the estimate of one coefficient depends on the estimate of another. You can turn the display of the t-statistic on or off with *showt y* or *showt n*.

Recursive OLS Regression

limit <date1> <date2> <date3>
recur <y> = <x1>, <x2>, <x3>, . . . , <xn>
recur <y> = ! <x1>, <x2>, <x3>, . . . , <xn>

The indicated regression will be done from <date2> to <date3>, then from <date2>-1 (the period before date2) to <date3>, then from <date2>-2 to <date3>, and so on back to <date1> to <date3>. The regression coefficients are made into time series and stored in the workspace. "b1" is the series for the first regression coefficient; "b2", for the second; and so on. The regression coefficients are stored in the date corresponding to the first date in their regression. Thus, "b1{date1}" would be "b1" for the regression over the period <date1> to <date3>. Similarly, the standard errors of the regression coefficients are stored in "s1", "s2", "s3", etc.

If a *zip* command preceeds the *recur* command, no regressions will be displayed, but the *zip* command will be turned off at the end of the *recur* command for you surely will want to view graphs after the recursive regression and they cannot be done with *zip* on.

Example:

```
zip
limit 1980.1 2005.1 2010.2
recur gnp$ = g$, v$, fe$, fi$
gdates 1980.1 2005.1
fadd BOUNDS.ADD (1 - 5)
```

where "BOUNDS.ADD" is the file

```
ti Estimate and 2-Sigma Limits for Coefficient %1
f upper = b%1 + 2*s%1
f lower = b%1 - 2*s%1
gr b%1 upper lower
```

intvar <prefix> <date1> <date2> [<date3> <date4> ... <dateN>]

The *intvar* command makes up linear interpolation functions between the given dates. Each linear interpolation function begins at 0 and remains at 0 until its particular time interval comes; then it rises by 1 each period until the end of its interval. After that, it remains constant at whatever value it has reached. For example, if the prefix is "tax", and we have 6 dates, 6 linear interpolation functions will be created, with names "tax1", "tax2", . . . , "tax6". Then, if we regress a policy variable, such as the federal tax rate, on these functions, the predicted values from the regression take the shape of a set of spliced line segments, approximating the curve of the left hand side variable.

For example, to approximate the federal personal tax rate in the Quest model, use:

```
r pitfBR = tax1, tax2, tax3, tax4, tax5, tax6
```

See also the following topics:

- Distributed lags
- Soft Constraints
- Seemingly Unrelated Regression (SUR)
- Homogeneity and normality tests
- Non-linear regression

2.4.2 Soft Constraints

Soft constraints are a way of combining *a priori* theoretical information or opinions about the desired value of estimated parameters with what the data suggests the value of those parameters should be. The commands available for applying soft constraints are the *con* command, which applies a constraint directly to a single parameter or to a linear function of a number of parameters; and the *sma* command, which softly constrains a number of distributed lag weights to lie close to a certain degree of polynomial. The *con* command is explained below. See the distributed lags topic for details of the *sma* command.

con <top> <c> = <linear function of ai's>

This command imposes "softly" the given constraint on the regression. The <top> (trade-off paramater) determines how "soft" the constraint with the convention: "the higher, the harder." Specifically, the constraint counts as $top \times T$ observations, where T is the number of regular observations in the regression. Soft constraints also are known as "Theil's mixed estimation," as "stochastic constraints," and also are a type of Bayesian regression.

Examples:

```
con 100 1 = a3 + a4 + a5
con 200 0 = a7 - 3a6 + 3a7 - a8
```

Ridge Regression

This technique amounts precisely to imposing a soft constraint that one or more of the coefficients should each be zero. For example,

```
con 100 0 = a2
con 100 0 = a3
```

will put <a2> and <a3> on a "ridge." Such constraints may increase the t statistic for a variable, but they almost certainly distort the regression coefficients.

Quadratic Programming Regression

G7 also can estimate regression parameters subject to "hard" constraints. These may be combined with soft constraints.

Suppose matrix A specifies linear combinations of the regression parameters β that is less than or equal to vector b. Then we have the conditions that

$$\begin{aligned} A \times \beta &\leq b \\ \beta &\geq 0 \end{aligned}$$

Constraints A and b are imposed with the *qpcon* command, and the parameter vector β is estimated with the *r* command. Soft constraints also may be added with the *con* and *sma* commands. These constraints are imposed on the next linear regression, which is specified by the *r* command. If no constraints are imposed with the *qpcon* command, then *G7* finds parameters using OLS. If constraints are imposed with *qpcon*, then *G7* finds parameters using a quadratic programming algorithm.

The syntax for the *qpcon* command is as follows:

qpcon b <sign> [constant][*]ai [<±> [constant][*] aj . . .]

where <sign> is <, =, or > (<= and >= also are accepted but are recorded as strict inequality constraints). <constant> are optional scalar multiples for the respective parameters. An asterisk may be included to clarify that the parameters are multiplied by the corresponding constant. The parameters are denoted by ai, where i is the position in the regression equation of the corresponding variable. If a constant is included in the regression, then the parameters are denoted as a1, a2, ... Right-hand-side terms are separated by either a + or -.

Examples:

```
qpcon 25.0 > a1          # constrain the constant
qpcon 13.0 = 1*a2 + 1*a3 # constrain the sum of slope parameters
```

Additional details on soft constraints and quadratic programming are available in the paper presented at the 2004 Inforum World Conference by Ronald Horst.

2.4.3 Distributed Lags

To easily estimate a polynomial distributed lag, in which the lag coefficients are constrained softly to lie along a certain degree of polynomial, use the *sma* command described here.

sma <top> <first> <last> <order> [f]
The *sma* command imposes a series of constraints which "softly" require the regression coefficients between <first> and <last> to lie on a polynomial of degree order. If the 'f' is present, the polynomial is "free" at the end; without the 'f', the polynomial will be assumed to be zero for the coefficient after <last>.

Examples:

```
sma 100 a4 a9 3 f
sma 100 a8 a16 1
```

2.4.4 Regression Tests

Tests of Homogeneity

chow <n>
where <n> is the number of regressions involved in the test. A Chow test is a special case of the Fisher *F* test used to test the homogeneity of a sample.

Typically, one runs two or more regressions covering parts of a sample and compares the sum of squared errors with that of a single regression covering the whole sample. If we were going to break up the sample into two parts, the command would be

```
chow 3
```

Then we run first the two regressions over the separate halves of the sample, and finally the single regression over the whole sample. After this last regression, one will be greeted by the results of the Chow test.

There is a special case in which the second "half" of the sample is too small to run the regression. In that case, give the command *chow 2* and then run the regression over the reduced sample and then again over the whole sample. After the second regression, the Chow tests appear.

Test of Normality of Errors

(norm)ality <y | n | f>
This command turns on (or off) tests of normality of the error terms. If the option chosen is 'y', then the Jarque-Bera test appears on the standard regression result screen labeled "JarqBer." Under the hypothesis of normality, it has a chi-square distribution with two degrees of freedom.

If the option 'f' (for full) is chosen, then before the regression results are shown, a table appears with moments and measures of symmetry and peakedness, as well as the Jarque-Bera statistics. If a catch file is active, this table will go into it.

2.4.5 Hildreth-Lu Technique for Autocorrelation Correction

hl <rho1> <rho2> <incr> <y> = <x1>, [<x2> [, <x3> [, . . . [, <xn>]]]]

In the Hildreth-Lu procedure, a guess of the autocorrelation coefficient of the errors, *rho*, is chosen, multiplied by the equation lagged once, and subtracted from the unlagged equation. The resulting equation then is estimated by ordinary regression. Another value of the guess of *rho* then is chosen and the process repeated. In this command, <rho1> is the starting guess of *rho*, <incr> is the amount by which it is incremented on each iteration, and <rho2> is an upper limit on the guess. The <y> and <x1>, . . . , <xn> values are as in the *r* command.

At the end of the process, you will get a table with this heading:

```
RHO-HL    SEE 1 AHEAD  RHO-EST    SEE LONG-RUN
```

RHO-HL shows the assumed *rho*, the SEE 1 AHEAD shows the standard error of estimate (SEE) of the estimated equation, RHO-EST shows the *rho* of the estimated equation, and SEE LONG-RUN shows the standard error using the fitted equation on the original, undifferenced data, without a knowledge of the true lagged value of the dependent variable, as must be done in forecasts of more than a few periods ahead. If the *save* command is on, all of the estimated equations will placed in the .SAV file as undifferenced equations suitable for going into a model. You must choose which one you want.

For graphing the HL output, one should run the equation one last time with the chosen value of *rho* as both the starting and ending *rho*. Then also run the equation with just the *r* command (no autocorrelation). Finally, construct two graphs:

```
gr depvar predp1 hlshort
gr depvar predic hllong
```

The first will compare the actual with two one-period ahead forecasts, the one from ordinary least squares (+'s) and one from Hildreth-Lu (x's). The second compares the actual with two forecasts not relying on the lagged value of the dependent variable. Again, the uncorrected least squares fit is shown by +'s and the fit using the H-L correction is shown by x's.

The pauses at the end of each fit in the HL procedure can be eliminated by the *zip* command. If you use *zip*, however, you must remember to do "zip off" before using the *(gr)aph* command.

G7 does not provide the Corchrane-Orcutt correction, since Hildreth-Lu is better.

2.4.6 ARIMA Techniques

ac <series> [n]

Calculate the autocorrelation function for the named series over the period specified by the latest *limits* command. The optional 'n' is the maximum number of terms to be calculated; its default value is 20. The output shows the autocorrelation function and the autoregression coefficients determined by the Yule-Walker equations. Taking the rightmost elements of each line of the autoregression coefficients gives the partial autocorrelation function. The function also is placed into the workspace with a name derived by adding "_ac" to the variable's name. This variable then can be used to graph the function. If the command was "ac vi 20", then the graphing command would be "gr vi_ac :0 20".

Example:

```
ac vin$ 11
```

bj <q> <y> = <x1>, [x2,] . . . , [xn]

Perform a Box-Jenkins estimation for autoregressive moving average models. Here <q> is the number of lags in the moving average error terms; e.g., with q = 2, u[t] = e[t] + b1×e[t-1] + b2×e[t-2]. This

<q> must be no more than 4. The program only will check for the stability of the solution if q <= 2. No more than 10 independent variables, please. At each iteration of the "Newtonian" non-linear regression algorithm, you will be allowed to intervene to try new values for the theta's, the coefficients for the moving average of error terms. If *save* is on, the first and the final equation will be in the .SAV file with the theta values appearing as coeffients of "theta". These must be removed before building a model with *Build*.

Example:

```
bj 2 vif$ = vif$[1], vif$[2]
```

2.4.7 Seemingly Unrelated Regressions

The format for the sur command is shown by this example:

```
sur
r y1 = x11, x12, ...
...
r yn = xn1, xn2, ...
con 10 1 = a2 + 4b5 + etc
sma 100 a6 a12 1
do
```

After the *sur* command come all of the individual regressions. Any *constraint* or *sma* commands must come after the regressions. In the *con* and *sma* lines, a's denote coefficients in the first regression; b's coefficients in the second, and so on. Thus, "a2" denotes the second coefficient of the first regression and "b5" denotes the fifth coefficient of the second regression.

The predicted values, dependent variables, and regression coefficients of the successive equations appear in the workspace file as "predic1", "depvar1", "rcoef1", "predic2", "depvar2", "rcoef2", etc. The results of the individual equations can be plotted by "gr *1", "gr *2", etc. The *graph* commands should follow the *do* command and each should be preceeded by an appropriate *title* command.

stack

The format for *stack* is exactly like that for *sur* except that the word "stack" replaces "sur". With *stack*, no attention is paid to contemporaneous covariances. The point of *stack* is solely to impose soft constraints across regressions.

Limitation: Earlier editions of *G7* imposed a limit of 30 regression variables, counting dependent, intercepts, and independent variables, that quickly could become binding in *sur* and *stack*, since it applied to the total variables in all equations involved in the *sur* or *stack*. In later versions, this constraint was relaxed so that *G7* now can employ up to 500 regression variables.

2.4.8 Common-Coefficient, Panel-Data, or Pooled Regressions

It fairly frequently happens that we have similar data sets collected at different times or in different places. We then may wish to estimate regressions that combine the data sets in various ways. The *comcoef* command in *G7* is designed for this purpose. It allows one to specify a number "common coeficients" in the regressions that follow. The method may be explained best with an example. In the following example, "cps78" and "cps85" are data banks of from the Current Population Surveys of 1978 and 1985, respectively. The first has data on 550 individuals, the second on 534 individuals. We want to regress the logarithm of an individual's wage, lnwage, on the individuals education, ed. We want the coefficient of lnwage to be the same in both samples, but the interecepts may be different in the two years. Here is what we do.

```
f one = 1
comcoef 1 4
hbk cps78
lim 1 550
r lnwage = ! ed, one
hbk cps85
lim 1 534
r lnwage = ! ed, one
do
```

In the *comcoef* command, the “1” tells the program to use the same coefficient for the first 1 variable(s) in all the following regressions. The “4” is the total number of variables, counting the dependent variable, in the combined or “pooled” regression. Once the *comcoef* command has been received, *G7* accumulates regressions until the *do* command is reached.

Only a limited list of commands is permitted between the *comcoef* command and the *do* command. They are the *bank*, *hbk*, *lim*, *f*, *add*, *pause*, *quit*, and *r* commands. The *quit* command only quits *comcoef*, not *G7*.

The *comcoef* command can be used under the *chow* command and under the *catch* command. It also puts the estimated coefficients into the *rcoef* series in the workspace bank, as usual. It does not enter a series of predicted values into the workspace bank, so *gr ** will not work following a *comcoef* regression. Nor are the variable names shown on the display of the results, for in general the variables with a common coefficient may have different names while variables with different coefficients may have the same name, such as “one” in the above example.

2.4.9 Non-linear Regression

G7 offers two algorithms for non-linear regression. The *nl* command uses the simplex method (sketched below and not to be confused with the simplex method of linear programming) and the Powell method. The form for the two is exactly the same except that the first is invoked by the *nl* command, while the second is invoked by the *nlp* command. We illustrate the with *nl* here.

nl <y> = <non-linear function involving n parameters, a0, a1, . . . an-1>
n <starting values of the parameters> <initial variations>

Example:

```
nl y = @exp(a0*@log(a1 + a2*x))
3
1.5  25.0  2.00
0.1   1.0  0.05
```

Since the non-linear function must be evaluated repeatedly, anything the user can do to speed the evaluation is helpful. For example, put all of the variables involved into the workspace; make the workspace no larger than necessary; take any functions of variables, such as logs or squares, which do not change through the calculations and put the results in the workspace and use them instead of, say, taking the log over and over.

In the simplex algorithm, ‘S’, the sum of squared errors, initially is calculated at the initial value of each parameter. Then, one-by-one, the parameters are changed by adding the initial variations and ‘S’ is recalculated at each point, thus yielding values at ‘n+1’ points (a simplex). By the algorithm sketched below, points in the simplex are replaced by better points or the simplex is shrunk towards its best point. The process continues until no point differs from the best point by more than one-tenth of the initial variation in any parameter. Each time a replacement is done, the simplex and the operation that yield it is reported on screen. Like all non-linear algorithms, this one is not guaranteed to work on all problems. It has, however, certain advantages. It easily is understood, no derivatives are required, the programming is easy, the process never forgets the best point it has found so far, and the process either converges or goes on improving forever. The display shows the regression coefficients, their standard errors, *t*-values, and variance-covariance matrix. The Powell method also requires no derivatives and has the property of "quadratic convergence." That is, applied to a quadratic form, it will find the minimum in a finite number of steps.

Following the *nl* command, there can be *f* commands, *r* commands, and *con* commands before the non-linear equation itself. These may contain the parameters a1, a2, etc.; they each should be terminated by ';'. The non-linear search then includes the execution of these lines. E.g.:

```
nl f x1 = @cum(s,v,a0);
y = a1 + a2*x1
```

If the name of the left side variable is "zero", no squaring of the difference between the left and right side will be done. Instead, the squaring must be done by the @sq() function. This feature allows soft constraints to be built into the objective function. For example,

```
nl zero = @sq(y - ( a0 + a1*x1 + a2*x2)) + 100*@sq(@pos(-a2))
```

will "softly" require a2 to be positive in the otherwise linear regression of y on x1 and x2.

If the name of the left-side variable is "last", no summing will be performed. Instead, the value of the function on the right at the "last" period will be minimized. This "last" period is the one specified by the second date on the preceding *limits* command. In order to combine summing of some components of the function on the right with the use of the "last" option, the *@sum()* function is used. It places the sum of the elements from the first date to the second date in the second date position of the output series, which is otherwise zero. This device is useful in some maximum liklihood calculations.

Both the simplex method and Powell's method are described in the book Numerical Recipes in C, by Wm. H. Press, et al., Cambridge University Press. The Simplex Method is sketched below.

Sketch of Simplex Method of Non-linear Regression

The method begins with a point in the parameter space of, say, 'n' dimensions and step sizes for each parameter. Initially, new points are found by taking one step in each direction. Actually, each step is made both forwards and backwards, and the better of the two positions is chosen. This initial operation gives 'n+1' points, the initial point and n others. These 'n+1' points in 'n' dimensions are known as a simplex. The algorithm then goes as follows.

```
Reflect the old worst point, W, through mid-point, M, of the other points to R (reflected).
(R is on the straight line from W to M, the same distance as W from M, but on the other side.)

if R is better than old best, B {
   expand to E by taking another step of length MR in the same direction.
   if E is better than R,
      replace W by E in the simplex.
   else
      replace W by R. (reflected)
   }
else if R is better than W {
   replace W by R in the simplex.  (reflected)
   }
else{
   contract W half way to mid-point of other points, to C. (contracted)
```

(continues on next page)

(continued from previous page)

```
  if C is better than W,
     replace W by C.
  else
     Shrink all points except B half way towards B. (shrunk)
  }
```

2.4.10 2SLS and 3SLS

Two-stage Least Squares

G7 needs and has no special command for ordinary two-stage least squares. Consider the following system of equations:

```
y1 = f(y2, x1, x3)
y2 = g(y1, x2, x3)
```

To obtain the 2SLS estimates, first do:

```
r y1 = x1, x2, x3
rename predic z1
r y2 = x1, x2, x3
rename predic z2
```

The z's now are the second stage regressors, so we do:

```
r y1 = z2, x1, x3
r y2 = z1, x2, x3
```

or, if we are building a model and want to have the correct names on the independent variables in the second stage regression, we can replace the last two lines by

```
rename z1 y1
rename z2 y2
r y1 = y2, x1, x3
r y2 = y1, x2, x3
```

Systemic Two-Stage Least Squares

The bane of two-stage least squares has been that there usually are so many exogenous variables and lagged values of endogenous variables in the system that the first-stage regressions fit so closely that there is no substantial difference between the OLS and the 2SLS estimates, especially if nonlinearities in the model are considered by including powers and cross products of the exogenous variables. *G7* and *Build* offer a new way to deal with this problem. The procedure is simple. Estimate the equations of the model by OLS or other single-equation method. Build the model with these equations and simulate it over the historical period. The simulated values of the endogenous variables are not influenced by the errors in the structural equations. They therefore may be used as regressors in the second stage to obtain estimates free of simultaneous equation bias. There are two variants of this procedure. In the first, the simulation is done using the simulated values for lagged values of endogenous variables; the other uses the actual values of these variables.

To perform the first variant, perform the simulation without any special command, copy the bank of simulation results to the *G7* workspace, start *G7*, select the 'b' option on the opening menu, do "bank bws" to put the actual values in the assigned bank, give the command "second on" so that *f* commands are ignored, and re-estimate the equations with the same add files as originally used. With "second on", the dependent variable of the regression will be taken from the assigned bank; all others come from the workspace, which contains the simulated values. To turn off the *second* feature, the command is "second off".

For the second variant, give the command first to the Run program before giving it the run command. Run will now store the simulated values in the workspace as before, but it will use actual values of all lagged variables. You may now proceed as before to start *G7* and give the second command. You will then be using simulated values of all independent variables. Alternatively, you can choose to use actual values of all lagged variables. To do so, proceed as before, but place an "a." in front of the name of every lagged endogenous variable in an *r* command. The "a." in front of a variable name causes *G7* to look for the variable only in the assigned bank, where it will find the actual value.

The command *second off* will restore *G7* to its normal condition in which it processes *f* commands.

Three Stage Least Squares

G7 performs three stage least squares by combining the two stage procedure explained above with the seemingly unrelated regression procedure described in the help topic for SUR. Consider again the system of equations:

```
y1 = f(y2, x1, x3)
y2 = g(y1, x2, x3)
```

Obtain the 2SLS instrumental variables by:

```
r y1 = x1, x2, x3
rename predic z1
r y2 = x1, x2, x3
rename predic z2
```

Insert the second stage regressors (z's) into the system and estimate with SUR:

```
title Two-stage and Three-stage Least Squares Estimates
sur
r y1 = z2, x1, x3
r y2 = z1, x2, x3
do
```

The first set of regression results are the 2SLS estimates, the second set are the 3SLS estimates. To graph the results use the same statements as for *sur*.

2.5 Modeling with *G7*

2.5.1 Building a Model

The "Model" item on the *G7* main menu will guide you through the steps of building and using a model. The Craft of Economic Modeling, Volumes 1-3, by Clopper Almon are available online in PDF form and provide additional guidance. References to this book are made in the text below.

The first step is to create a Master file that specifies the regression result files – the .SAV files – which will be used in the model together with other identities necessary for the model. Selecting Model | Master will show the current contents of the master file in the editor. Each regression equation is brought into the model with a command of the form "add XXX.SAV", where xxx is the usually the name of the dependent variable in the regression. This must be the same name that appeared on the save line in the .REG file by which the equation was estimated. When you are adding an equation to the model, the first thing you should do is to modify the master file to include the new equation. Generally, it is not crucially important where the *add* command appears, but it is natural to place it after the determination of the explanatory (right-hand-side) variables that it uses.

Besides the *add* commands to include regression equations, the Master file may also have identities such as *f* or *id* commands and definitions of exogenous variables with *fex* commands.

At the bottom of the master file there should be one or more *check* commands, for example:

```
check cR 0.2
```

The *check* commands control the iterative solution of simultaneous models as explained relative to Model 5 in Chapter 1.

Note carefully that you conclude the editing of a Master file by clicking Save on the editor menu – not Run. You want to use the Master file as input into the *Build* program, not the *G7* program. Clicking Run will accomplish nothing useful and may consume a good bit of time as you respond to various "bad command" messages.

The second step is to estimate all of the equations and save the results in .SAV files. This step is not mentioned under the Model menu items because it involves all of the other capabilities of *G7*. We already have seen several examples and the following chapters have many more.

The third step is to build the model. This step is accomplished by selecting Model | Build. This command opens a command prompt window and executes the MODEL.BAT file that should be in the folder containing your master file. This batch file contains the following lines:

```
build master
make -fc:\pdg\run.bpr
pause
```

For the first line of MODEL.BAT to work correctly two things are necessary:

- The *Build* program, BUILD.EXE, must be in the C:\pdg directory and this directory must be on the system's path. This directory should also contain the files RPROTO.H, COMMON.INC, and GCOMMON.INC. All of these files will have been placed in C:\pdg in the process of installing *G7*. To put this directory onto the system path system path, see the Installation page of the *G7* Help Files or see the README documents that accompany the software installation files.
- A BUILD.CFG file must be in the directory (or folder) in which you are building the model. Here is the BUILD.CFG file for the Quest model. It is very similar to the G.CFG file and even has two irrelevant lines to maintain that similarity. You can copy this file from the AMI demonstration program and modify it to fit your needs, or you can use this example.

  ```
  Name of workspace; bws.bnk
  Name of workspace index; bws.ind
  Name of default bank; quip.bnk
  Name of default bank index; quip.ind
  Path to files (like RPROTO.H) for inclusion; c:\pdg
  Reserved for future use; x
  Default regression limits (not actually used); 1980.1 1998.3 1998.3
  Default base year of workspace file; 1960
  First month covered; 1
  Default maximun number of observations per series in workspace; 240
  Do you wish to supply convergence test and subdivision manually?; n
  Is there a global declaration file "global.inc" to be included?; y
  ```

For the second line of MODEL.BAT to work correctly, the Borland C++ compiler must have been installed and be on the system's path. Installation normally puts it on the path.

The *Build* program, run by this step, reads the master file and all the .SAV files, writes a C program, HEARTA.C, to execute the model, and compiles and links this program to produce an executable file, RUN.EXE, which is the model in the form of a program ready to run. This step also creates

bwsw
: is a data bank containing all the variables used in the model

RUN.GR
: a command file for *G7* to graph all the variables in the model

RUN.ERR
: a record of all the diagnostic information provided by the Build program. Be sure to check it by looking at it with the editor. If you have included the command *addtype y* in your Master file, then this file also will contain everything in the Master file and in the files which are "added." If you include at the top of the Master file the command *checkdup y*, then *Build* will check every *f or *fex* command to see if it is calculating a variable that had been calculated earlier and, if so, whether the values are identical. This check is a powerful means of identifying common errors in model building. Check your RUN.ERR file frequently.

EXOGALL.REG
: a command file for *G7* to create mechanical projections of all exogenous variables in the model by regressing them on time. Look at EXOGALL.REG to see what it does. The projections will be put into files with the variable name as the name of the file and .XOG as the extension.

RUN.XOG
: a file to use all the .XOG files made by running EXOGALL.REG.

SKIPALL.FIX
: a file to "skip" the calculation of all the variables calculated by regression equations. When all the regression equations are skipped, these variables become essentially exogenous and the model becomes tautological. Hence, its historical simulation should completely reproduce history. If it fails to do so, then there is something wrong with the identities. Since it is very easy to make mistakes in the identities, it always is important to run a model with all regressions skipped. This file makes it easy to do so.

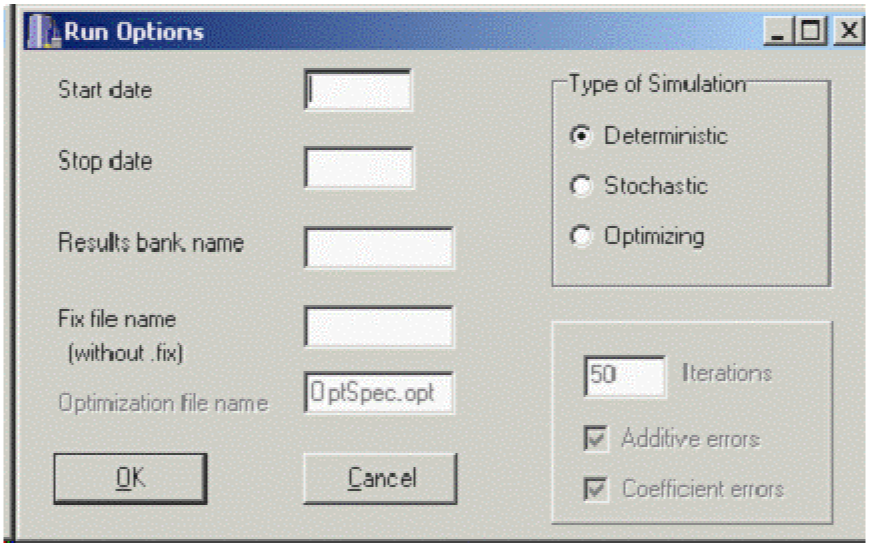
RUN.NAM
: a file with the names of all variables in the model and their numbers necessary to read the hearta.c program should you have to do so. With luck, you won't need to ever look at heart.c or run.nam.

HEART.DAT
HEART.DIM
RUN.LAG
RUN.MAP
: these files are internal to the system. You should not need to look at them.

With the model built, you are ready to run it. See the next topic for details.

2.5.2 Running a Model - Deterministic Simulation

When a model has been built, the next step naturally is to run it, that is, to make simulations or forecasts with it. This is accomplished by selecting Model | Run. You will be presented with the form shown below.



Run Options

Start date

Stop date

Results bank name

Fix file name (without .fix)

Optimization file name OptSpec.opt

Type of Simulation

● Deterministic

○ Stochastic

○ Optimizing

50 Iterations

☑ Additive errors

☑ Coefficient errors

Fig. 1: **Macro Model Run Window**

You first should select the type of simulation you wish to make in the panel in the top right. Most simulations or runs are deterministic, and we will initially assume that to be the case. The other two types of simulation are covered in the following two help topics.

Fill in the starting and stopping dates of the run. They should be in the usual form for dates in *G7*: 1980 for the year 1980, 1980.1 for the first quarter of 1980, and 1980.001 for January 1980.

Then fill in the name of the output bank. If you are running a historical simulation, HISTSIM would be a natural name. For a base forecast, BASE would be a good name; for high monetary growth scenario, HIMONEY could be a good name. This will be the root name of the .BNK and .IND files that compose the *G7* data bank containing the results of the simulation.

Next, fill in the root name of the "fix" file. This is the file (which should have a file name without spaces and ending in .FIX) that provides values of exogenous variables with the *(up)date* commands and may in various ways to override or modify the results of the regression equations. The commands for doing so are *rho*, *cta*, *mul*, *ovr*, or *skip*.

To specify the complete course of the exogenous variable gR from 1995.1 to 2000.4, the command would be

```
update gR
1995.1 1411 1415 1413 1401
1996.1 1421 1429 1416 1419
1997.1 1435 1457 1462 1465
1998.1 1458 1483 1486 1494
1999.1 1512 1522 1544 1579
2000.1 1589 1610 1601 1617
```

Each line of the data begins with a date followed by the value of the variable in that quarter and as many successive value of the variable as you care to put on the line. Thus, this data could equally well have been supplied in the form

```
update gR
1995.1 1411 1415 1413 1401 1421 1429 1416 1419
1997.1 1435 1457 1997.3 1462 1465 1458 1483 1486 1494 1512 1522 1544
1999.9 1579 1589 1610 1601 1617
```

For specifying future values of an exogenous variable, it often is adequate to use linear interpolation between a small number of given values. A 0 in the data for an *(up)date* command means that the value is to be computed by linear interpolation. For example, the lines

```
update gR
2001.1 1630 0 0 0
2002.1 0 0 0 1700 0 0 0 0 0 0 0 0 0 0 0 1800
```

will make gR rise in a straight line from 1630 in 2001.1 to 1700 in 2002.4 and then on up by another straight line to 1800 in 2005.4.

The *update* command may be abbreviated as *up*. In counterhistorical simulations, that is, simulations over the historical period with some exogenous variables having values different than their historical values, it often is convenient to use the capacity of the *update* command to add a constant to all values of the variable or to multiply all values by a constant. Thus,

```
update gR +100
```

would add 100 to all values of gR, while

```
update gR -100
```

would subtract 100 from all values and

```
update gR *1.05
```

would multiply all values by 1.05.

You also can modify the values produced by the equations of the model. You could ask, for example, how the economy would have behaved if investment had always been 20 percent above what the equation predicted. Such modifications of the predicted values are referred to, somewhat disparagingly as "fixes." There are five types of fixes available to you:

cta

Constant Term Adjustment fix (also called "adds"). Add a specified amount to the value calculated by the equation. The amount added can be different in different periods.

mul

Multiplicative fix. Multiply the value calculated from the equation by a specified factor, which may be different in different periods.

ovr

Override fix. Over-ride the equation. Discard the value calculated and use the one specified.

rho

Rho-adjustment fix. Add a rho-adjustment to the variable

skip

Skip fix. Ignore the value predicted by the equation and uses the historical value in its place.

The format for the first three is similar that of the *(up)date* command:

fixtype <variable_name>
    <data lines>

For example,

```
cta cpcR
  1998.3 -20 -20 -25 -30 -30 -25 -15 -12 -10 -8 -4 -2 ;
```

The format for the rho-adjustment fix is

rho <variable_name> <rho_value>

For example,

```
rho vfR 0.9
```

The error of the regression equation in the starting year of the run will be calculated and then *rho* times this error will be added to the predicted value one period ahead, *rho* squared times the value added to the predicted value two periods ahead, and so on.

The format for skip is just

skip <variable_name>

For example,

```
skip cpcR
```

These fixes work only when applied to variables that are the dependent variable in a regression. When applied to other variables, the forecasting program does not reject them; it just never uses them. Furthermore, only one can be applied to any one variable. If more than one is applied, only the last takes effect. For example, if somewhere in the fix file after the above "cta cpcR" there should appear "rho CPCR 0.9", the *cta* will be forgotten completely by the program without any warning.

When you click the OK button, a command prompt screen opens and you get a] prompt. You may give any of the above commands at this point. Generally, however, you will have all of these commands in the .FIX file and will only wish to give a full title to your run, something like

```
ti Historical Simulation
```

You get the] prompt again, and you type "run", and the model runs. More precisely, when the OK button is clicked, a file by the name of _GO.BAT is created, a command prompt session is started, and this file executed. The folder that contains the model should also contain the file RUN.CFG, which may be copied from that for the AMI demonstration model.

2.5.3 Optimizing a Model

The optimization procedure will vary specified regression coefficients to minimize the value in the last period of the simulation of a specified objective function variable.

For example, to optimize the fit of the model as measured by the sum of the squares of percentage errors in the simulated value of real GDP (gdpR) and nominal GDP (gdp), we can define in the model the objective function as the last value of misses defined by

```
fex gdpRX = gdpR
fex gdpDX = gdpD
f miss = @sq((gdpR-gdpRX)/gdpRX)+@sq((gdpD-gdpDX)/gdpDX)
f misses = @cum(misses, miss, 0.0)
```

We then need to convey to the simulation program which of the many equations in the model is the objective function and which of the many regression coefficients can be varied to minimize the last value of this variable. Both of these tasks are done by the optimization specification file. By convention, its name should have the extension .OPT and it should have the form illustrated below.

```
misses : Name of objective function
20 : Maximum number of coefficients to be varied.
vfnreR
# 1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25
0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.1
cRpc
# 1     2    3    4    5    6    7    8    9    10   11   12   13   14    15   16
0.10 0.001 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.005 0.00 1.00
```

The first line of the file gives the name of variable whose last value is to be minimized. The second line specifies the maximum number of parameters which will be varied in the course of the optimization. It does not hurt if it is larger than the number actually used. Here we have set the maximum at 20 but will only use 6.

The next line says that some parameters will come from the equation for vfnreR. The third line begins with a # which marks it as simply a comment to be ignored by the program. For us, however, it is very useful since it numbers the 25 regression coefficients which occur in the equation for vfnreR. The line below it gives the step sizes for each of these 25 coefficients. A coefficient given a step size of 0 is not involved in the optimization. Thus we see that coefficient 1, the intercept, is given a step size of 0.1 and that the coefficient for the 25th variable also is given a step size of 0.1. The next triplet of lines does the same for three coefficients in the cRpc equation.

The step sizes determine an initial simplex of points which are then varied by the downhill simplex downhill simplex method to find the optimum. (See Wm. Press et al., Numerical Recipies in C.) Choice of these step sizes can matter. If they are too large, some steps may take the model into regions where it does not converge. If they are too small, the algorithm may conclude that it has converged before it really gets started.

To run an optimizing simulation, click Model | Run and in the "Type of Simulation" panel, click on the button for "Optimizing." Fill in the dates, fix file name, and results file name as usual. Also give the name of the optimization specification file. Then click OK. As usual, you will get a command prompt window and a] prompt. You may add any sort of fix command and give a title to the run. Then give the command *run*.

Running the optimizing simulation will create a file with the optimized values of all regression coefficients in the model. It will have the same root name as the Optimization file but with the extension .DAT. If the optimization file was OPTSPEC.OPT, this file will be OPTSPEC.DAT. This file has exactly the same format as the HEART.DAT file created when you originally built the model. If you now wish to run the optimized model, you need only copy the OPTSPEC.DAT file to HEART.DAT and run the model. You probably first should save the original HEART.DAT file, so after optimizing the model, in *G7* you should do

```
dos copy heart.dat original.dat
dos copy optspec.dat heart.dat
```

and then run the model again.

2.5.4 Stochastic Simulation

G7 allows models to be run with stochastic simulation. To do stochastic simulation with a model, one or more of its regression equations must be estimated after giving *G7* the command

```
stochastic y
```

In the save (.SAV) files, the equation then will be followed by the letter "s", the Standard Error of Estimate of the equation (SEE), the value of Rho, and the variance-covariance matrix of the regression coefficients. Running *Build* with such save (.SAV) files automatically will create a model that can be run with stochastic simulation. To run stochastic simulations, one must select the stochastic option in the run-model window and specify the following additional options:

Number of iterations
This is the number of model solutions for random simulation; the default is 50.

Additive errors
Check this box to run the simulation with random additive errors.

Random Coefficients
Check this box to run the simulation with random coefficients.

In each iteration, values of the additive errors, or of the random coefficients, or both are picked and a deterministic simulation is run with those values. The additive errors will have the standard errors and autocorrelation coefficient found from the residuals in the regression. The random coefficients will have the variance-covariance matrix variance-covariance matrix found for the regression coefficients in the regression.

A run with stochastic simulation creates two banks. One is the usual output bank specified on the form which appeared when you clicked Model | Run. In the case of stochastic simulation, this bank contains the means of the various simulations. The second bank created by the stochastic simulation always is called SIGMA. It contains the standard errors of the simulations of the various variables in the model.

2.5.5 Graphing Model Results

To look at the results of a model graphically, it is convenient to create special files to be executed by an *add* command. These files may be called "show" files and denoted with the extension .SHW to make them easy to find and identify.

To make a show for a historical simulation, we need to assign the bank with the actual data, usually called the BWS bank, and another with the results of the simulation, which we may assume was called HISTSIM. We do so with the commands:

```
bank bws c
bank histsim b
```

Then we have a number of *(ti)tle*, *gname*, and *(gr)aph* commands to make the graphs. We can put all of these in an add file, such as HIST.SHW, when we execute the file by

```
add hist.shw
```

A simple HIST.SHW file is may look like this:

```
bank histsim b
bank bws c
gdates 1980.1 2001.1
title GDPR -- REAL GROSS DOMESTIC PRODUCT
```

(continues on next page)

(continued from previous page)

```
gname gdpR
gr b.gdpR c.gdpR
title cR -- CONSUMPTION
gname gcR
gr b.cR c.cR
title vfR -- GROSS PRIVATE FIXED INVESTMENT
gname vfR
gr b.vfR c.vfR
```

and so on for as many series as we want to graph. The *gname* lines give the name of the file (to which .WMF will be added) if the user clicks save in the panel that pops up to control the execution of the add command. In the *gr* lines, the "b." and "c." tell *G7* to look for these series in and only in bank 'b' or bank 'c', respectively. The effect of this command, therefore, is to graph the gdpR series from histsim as the first line, the red line, and then to graph the gdpR series from bws as the second, the blue line. The result is a graph with the actual historical course of gdpR shown in blue (true is blue) and the model's calculated gdpR shown in red.

2.5.6 Tabulating Model Results or Data

Tables of data from model runs are prepared by the powerful *Compare* program, so called for its ability to compare data from various runs of a model. However, it also can show data from a single run or from a data bank.

Compare is activated through the Model | Tables on the *G7* main menu. Selecting this item brings up a form for preparing input data to *Compare*.

At the top left of the form, you are asked for the name of the stub file. The stub file fundamentally gives the names of the series to be put into the table and the dates to be listed. However, it may contain many commands to *Compare*, as will be described further below.

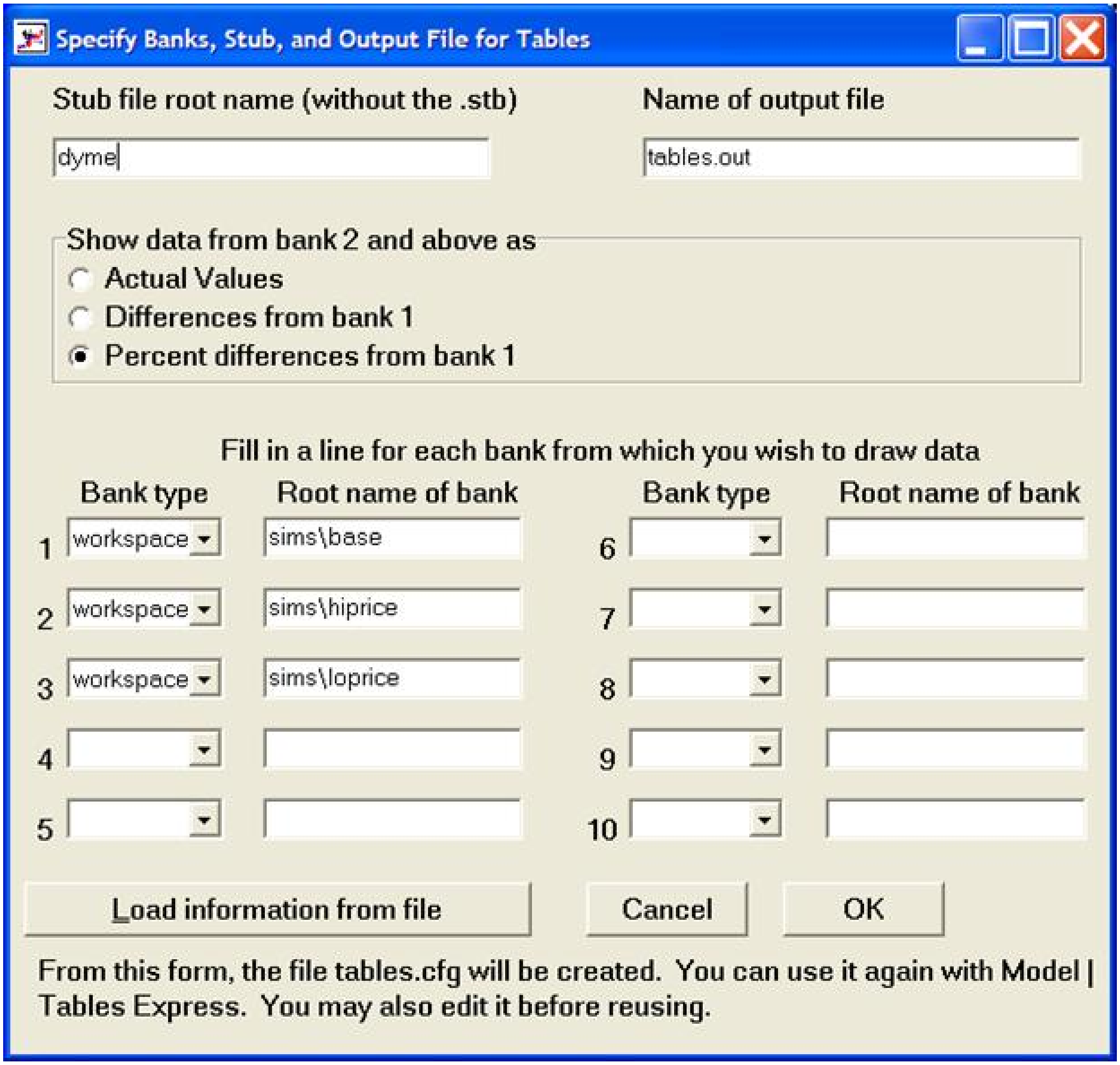
At the top right of the form is space for the name of the "output file," the text file that will be produced as the output of the comparison. An acceptable entry would be HISTSIM.OUT.

Next, you must specify how you want to show data from the alternative banks. The base bank, the first, will always be shown in actual values. The others, however, may be shown as deviations from this base, as percentage deviations from the base, or as actual values. Usually – but not always – the best choice is deviations, so that is the default value.

Then we must enter the types and names of the banks we want to compare. Unless you are using the Interdyme system, the type almost always is "workspace." *InterDyme* users also might have Vam banks. To compare a historical simulation bank, "histsim", against actual values in a bws bank, put "bws" in the name field for the first bank and "histsim" in the name field for the second bank. Then click OK.

Compare is a console application program; it will run in a command prompt window. Normally, there is nothing to do except tap any key when the program has finished. However, if *Compare* finds fault with your commands, it will let you know in this window, so look for messages before blithely tapping a key.

Note that the configuration that you specify in the Tables window will be saved to file as COMPARE.CFG. This configuration will load automatically the next time that you open the Tables window. You may load an alternative configuration from file by selecting the "Load information from file" button. Note too that *G7* will create a file called TABLEX.BAT, if it does not exist, when you click OK. This batch file is executed by *G7*, and the *Compare* program is run by way of a command in the batch file. If the TABLEX.BAT file already exists, *G7* simply will run it. You thus may edit the TABLEX.BAT file to add additional DOS commands or to modify the path to the COMPARE.EXE program.



Specify Banks, Stub, and Output File for Tables

Stub file root name (without the .stb): dyme

Name of output file: tables.out

Show data from bank 2 and above as

○ Actual Values

○ Differences from bank 1

● Percent differences from bank 1

Fill in a line for each bank from which you wish to draw data

	Bank type	Root name of bank		Bank type	Root name of bank
1	workspace	sims\base	6		
2	workspace	sims\hiprice	7		
3	workspace	sims\loprice	8		
4			9		
5			10		

From this form, the file tables.cfg will be created. You can use it again with Model | Tables Express. You may also edit it before reusing.

Fig. 2: **Compare Window**

When *Compare* has finished and the window has closed, look at the output file, HISTSIM.OUT in our example, with the editor. Be sure that the editor is using a monospaced font, such as Courier, Line Printer, or Fixedsys; otherwise, the columns of numbers might not line up vertically. Now let us look at a stub file. Here is AMI.STB:

```
\\dates 1980 1982 1986 1988 1990 1992 1994 1996 1998
\\under = \\7 1 65 2 2 45
;
&
gdpR ;Real Gross domestic product
cR ; Personal consumption
vR ; Gross private domestic investment
vfR ; Fixed investment
viR ; Inventory change
feR ; Exports
fiR ; Imports
gR ; Government purchaes
;
gdpD ;GDP Deflator
gdp ;Gross domestic product
pibg ; Personal Income before gov
indtax ; - Indirect taxes ptax ; - Personal taxes
ngtpp ; + Gov transfers to persons
pidis ; = Personal disposable income
piipcb ; - Interest paid by consumers
pipttf ; - Transfers to foreigners
c ; - Personal consumption expenditure
pisav ; = Personal saving
```

The commands beginning with a “\” are commands to *Compare*. The first one sets the dates for the data to be shown. A date like 1990 is an annual date, and *Compare* will take an annual average of the quarterly data in our model. We also could give a quarterly date like 1990.2. If we include intervals in the dates, like 1990-1995, *Compare* will show the growth rate over that period. The growth rates will be calculated with continuous compounding. (Other compounding specifications are available, but they generally are inferior to the default.) The “\under” command just sets the character that will be used for underling in the table.

The “\” command by itself, as in the third line, sets the following parameters:

fw
field width of the numbers in the table

dp
decimal places to be show in the table

pl
page length, in lines

tm
top margin, in lines

bm
bottom margin, in lines

tw
width of the titles on the left side of the page.

An “&” at the beginning of a line causes the dates to be written across the page and underlined with the underline character. A line with a “*” in the first position forces a new page. The other lines, that is to say, most lines, have variable or expression on the left followed by a ';' followed by a title. The title will appear at the left of the page and the value of the expression will be shown in the columns under the appropriate dates.

There are many other *Compare* commands. You also may wish to read the documentation in COMPARE.PDF. Additional details are available elsewhere in the Help files.

2.5.7 Model Building

This help topic discusses some of the *G7* commands that are useful for model building. See The Craft of Economic Modeling by Clopper Almon for more on the use of *G7* and *Build* to build macroeconomic models.

save <filename>
Opens the named file which then receives all *(ti)tle*, *f*, *fex*, *id*, and *cc* commands exactly as they appear on the screen. It also catches the output of *(ty)pe* commands but with the *(ty)pe* changed to update. Most important, the file receives the results of regressions in the form of equations with the variable names and values of the regression coefficients. Use *save off* to close the saved file and stop saving. By convention, not necessity, all "save" files are given the extension ".SAV".

f, *fex*, and *fdup* all do the same thing in *G7*, but when passed to *Build* through a *G7* "save" file:

f
Computes the left-side variable and puts it and all of the right-side variables into the workspace, and also puts an identity into the model.

fex
Computes the left-left side variable and puts it into the workspace. It does NOT put the right-side variables into the workspace and does NOT put an identity into the model. It is used to define historical values of exogenous variables, such as tax rates, and historical values of dependent variables of behavioral equations.

fdup
This has no effect whatsoever in *Build*.

id <identity>
This has no effect in *G7* but, when passed to *Build*, puts all variables (without computing the left-side one) into the workspace and puts an identity into the model.

check <variable name> <tolerance>
This command can only be placed in a "master" file for input into *Build*. It causes the named variable to be checked for convergence to within the specified tolerance. The values of the checked variables will be printed on the screen as the model runs. It generally is necessary to check only a few of the variables in a model.

cc <C language statement>
This has no effect in *G7* or *Build*, but the C-language statement is passed through to the file HEART.C, which contains the model written in the C language and is ready for compilation. Such passed-through statements may be used to do special tasks for which *G7* and *Build* have no convenient mechanism – such as scaling one group of variables to sum to another variable. For many models, no *cc* statements are needed.

sty <variable name>
Silent type. Prints the variable to the file being saved, but does not show it on the screen. See also the *type* command.

Matrix Operations in Models

For models involving matrices that vary over time, the *InterDyme* system should be used. However, constant matrices can be built into a model as described here. The following matrix commands can be placed only in a master file for input into *Build*. When they are used, the value of *dimmax* in COMMON.INC must be set equal to the largest dimension of the largest matrix.

vec <y> = <n1>, <n2>, <x1>, <x2>, . . . , <xn>
vec <y> = <n1>, <n2>

Defines the vector <y> with <n1> rows and <n2> columns. The two-dimensional specification is used to distinguish between row vectors and column vectors. The *vec* command forms the vector <y> by putting the variables <x1>, <x2>, . . . , <xn> into the corresponding components of <y>. When the variables <x1>, <x2>,. . . , <xn> are not specified, the *vec* command defines a zero vector. In the following example, fd is a column vector and nulv is a row vector. ::

Example:

```
vec fd = 2,1,auto,food
vec nulv = 1,4
```

mat <y> = <n1>, <n2>, <filename>
mat <y> = <n1>, <n2>

Defines the matrix <y> with <n1> rows and <n2> columns. The entries of <y> are fetched from the given file <filename> row by row. When the file name is not specified, the *mat* command defines a zero matrix.

Example:

```
mat iomat = 78, 78, io77.usa
mat cmat  = 78, 34
```

mop <y> = <x1> + <x2> (matrix addition)
mop <y> = <x1> - <x2> (matrix subtraction)
mop <y> = <x1> * <x2> (matrix multiplication)
mop <y> = <x1>' (transposition)
mop <y> = <x1> ^ <x2> (vector component-wise multiplication)

Performs matrix operations. When addition, subtraction, multiplication, or transpose is done, the resulting matrix is stored in matrix <y>. With the ^ operation, the i-th component of the vector <y> will be the product of the i-th components of vectors <x1> and <x2>. WARNING! All matrices and vectors should be defined by the *mat* command and *vec* command before using the *mop* command.

2.5.8 Running an *InterDyme* Model in *G7*

The InterDyme software is distributed as a very simple model based on Chinese data. (A newer version, called Tiny, is built with U.S. data.) This model contains only the output and price solution and a simple imports equation. This simple model, called *Slimdyme*, should be installed and compiled first to make sure that everything is working correctly. The *Slimdyme* model file contains comments pointing out where you need to add the code for the functions of a typical interindustry model, such as final demand equations, employment equations, value added equations, etc. To install, create a directory called SLIMDYME and unzip the *Slimdyme* files into that directory. The SLIMDYME ZIP file will be named DYMEVxxx.ZIP, where the "xxx" indicates the current version of *InterDyme*. You also should have installed Borland Builder, which includes a 32-bit Borland C++ compiler called by "bcc32", or alternatively install the Borland free compiler.

Here is a sequence of steps to operate the model. It also tests that everything is installed properly.

From the *G7* command line, do

```
add initial
```

The file "initial" has the following contents.

```
# INITIAL for SLIMDYME
# Create vam file
vamcreate vam.cfg nohist
# Assign as bank b
vam hist b
# Make b the default vam file
dvam b
```

Set up a constant A-matrix, using the lint command:

```
add am.dat
fdates 1980 2000
f mover = 1
index 1980 mover am
```

Bring into the VAM file from the Mudan

```
# G7 bank data for some vectors.
ba mudan
fadd getout.add   sec33.fad
fadd getfd.add    sec33.fad
fadd getim.add    sec33.fad
fadd getprice.add sec33.fad
fadd getva.add    sec33.fad

# Move 1990 final demand vector, fd,
# forward by 2% per year.

add indexfor.add 1990 .02 fd

# Store the loaded vector, fd.  This is
# necessary only at the end of the work
# with the vam file.
store
```

(continues on next page)

(continued from previous page)

```
# Before you can do Model | VecFixer, you
# must do "close b", but you may first want
# to "show b.fd" or show other vectors or
# matrices.
```

When the program finishes and the blinking prompt returns to the *G7* command line, you may, as indicated by the comments at the end of the "initial" file, either first look at some of the vectors and matrices or proceed to build the rest of the model. Before proceeding, however, you first must give the command

```
close b
```

Build the macro part of the model by selecting from the *G7* main menu

```
Model | IdBuild
```

This step also compiles and links the simulation model. Next, prepare the macro fixes:

```
Model | MacFixer
```

From the information on the form opened by this command, *G7* will prepare the file MACFIXER.CFG and then execute the *MacFixer* program. For *Slimdyme*, you always should accept the default information in the form that opens in this and the following menus.

Prepare the vector fixes:

```
Model | VecFixer
```

From the information on the form opened by this command, *G7* will prepare the file FIXER.CFG and then execute the *Fixer* program.

Run the model:

```
Model | Run Dyme
```

From the information on the form which this command opens, *G7* will prepare the file xxx.CFG, where xxx is the name of the model as you indicate on the form. The model then will be executed with this control file.

Execute the *Compare* program with the menu command

```
Model | Tables
```

The data in the form is used to make a control file for the *Compare* program. If you are running Compare again with exactly the same input control, you can use

```
Model | Express Tables
```

Once you have verified that *Slimdyme* is working correctly, you can create a directory for your own model. To copy only the necessary files into that directory, first copy the file GETDYME.BAT from the SLIMDYME directory to your new directory, and then type "getdyme slimdyme" from that directory. You also should examine the GETDYME.BAT file to see what it is copying. Before proceeding much further, you should study the *InterDyme* manual to learn how to build an interindustry model. Also, see The Craft of Economic Modeling on the Inforum web site.

2.6 Vectors And Matrices

2.6.1 Working with Vam Files in *G7*

G7 is part of the Interdyme software for building dynamic interindustry models. These models store their vector and matrix data in files called Vam (for Vectors And Matrices) files. Usually a model has a single Vam file that contains all vectors and matrices used in that model. To speed reading and writing during the execution of the model, these files are organized quite differently than the usual *G7* data bank. The usual bank places the data for successive observations on a single variable in adjacent memory positions. The Vam file, on the contrary, organizes all of the cells of a vector for one year before going on to the next year. This difference largely is transparent to the user of *G7*, but it explains why some ways of working are faster than others.

VAM files can be created and viewed in *G7* and used as a source of data for regression or graphing. They are just another type of data bank as far as *G7* is concerned. Several Vam files can be attached at the same time and use as sources of data. Data input and manipulation, however, is possible in only one Vam file at a time; this one is called the default Vam file. The next sections explain how to create a Vam file, how to make it the default Vam file, how to load data into it, and how to transform it in several ways.

G7 can create and manage Vam banks at annual, quarterly, and monthly frequencies. At this time, the other model building programs support only annual frequency for Vam files. These other programs include *IdBuild*, *Fixer*, *MacFixer*, *Compare*, and the *InterDyme* C++ library.

2.6.2 The Vam Configuration File

Every Vam file begins from a Vam configuration file, often called VAM.CFG, which sets out the data content of the model in so far as the data is best thought of as vectors and matrices. This configuration file contains some vital information – namely, the starting date and ending dates for all data in the vam file – the range of years over which the model can be run or graphs and tables made. Then, for each vector or matrix in the model, there is a line containing :

- its name,
- its number of rows,
- its number of columns,
- if a vector, the number of lagged values with which a vector can be used in the model, or if a matrix, whether or not it is "packed" so that only non-zero elements are stored,
- the file name of the titles for the rows of a vector or matrix, and if a matrix, followed by the file name of the titles for the columns of the matrix.
- a '#' followed by a comment usually explaining the economic nature of the variable and possibly the name of the file with historical data.

The format is free. Here is part of the VAM.CFG for the Mudan model of China:

```
#         Matrices and Vectors of the Mudan Model
#
1980 2020 # Starting and ending years
#
#Name |Number of  |Files of titles of|    Description
#     |row col lag|  rows    cols    |
#
```

(continues on next page)

(continued from previous page)

```
# Matrices
am   33 33 0 sectors.ttl sectors.ttl # the input output matrix
bmcr 33 11 0 sectors.ttl hhrural.ttl # the bridge matrix for cr
bmcu 33 19 0 sectors.ttl hhurban.ttl # the bridge matrix for cu
bmv  33 40 p sectors.ttl bmv.ttl     # the bridge matrix for investment

# final demand vectors
hcr  11  1 0 hhrural.ttl             # Consumption of Rural Hh, Hh sectors
hcu  19  1 0 hhurban.ttl              # Consumption of Urban Hh, Hh sectors
...
out  33  1 3 sectors.ttl             # Output
...
#
# Vectors related to fixes
#
dump   33 1 0 sectors.ttl        # Discrepancy of Fixed Outputs
pdump  33 1 0 sectors.ttl         # Discrepancy in prices
fix   100 1 0 fix.ttl            # Fixes, to be filled in by Fixer
```

Note that the starting and ending dates do not control when a particular run of the model starts or stops, but rather they define the range of the Vam file. The model in the example cannot start earlier than 1980 or run past 2020, but it certainly does not need to run over the whole span on each simulation. In this example, the model will have a vector named "hcr" which will have 11 rows and 1 column, as indicated by the first two numbers on the line; only current year values of this vector are needed in computing the results of the model in the present year, so the number of lags used, the third number on the line, is 0. In the example above, only the "out" vector requires lagged values. Note the 'p' in this position of the "bmv" matrix. This 'p' marks a matrix that is to be "packed" so that only non-zero items are stored. (This presently is not the actual case in the Mudan model.) The next item on the line is the name of the file with the sector names to be used for the spreadsheet-like displays of the vector. Everything following the '#' is a comment.

Names of vectors may contain up to 16 letters or numbers and may contain the underscore mark, "_". They must not, however, end in a number. This restriction is necessary because sometimes it is necessary to use the sector number as a suffix to the vector name and to convert between the suffix and subscript forms of the name. For example, we must be able to recognize that pce[23] and pce23 are the same series. If we had a vector named g2, then g2[3] would convert to g23, and g23 would convert back to g[23], which is wrong. So no numbers at the end of vector names, please!

G7 is case sensitive; Q is not the same variable as q.

When using a Vam file, you may need a reminder of the names of the various vectors and matrices. You can use *G7*'s editor to look back to the VAM.CFG file or, if the names alone are sufficient reminder, you can use the *listvecs* command.

listvecs
lv [-sr] [<bank_location> [<wildcard>]]
lvc [-sr] [<bank_location> [<wildcard>]]

> The *listvecs* or *lv* command works just like the *lis* command, but all vector and matrix names in the default vam file or specified bank are printed. The *lnc* command is the same but prints names in one column. Option 's' sorts the series in alphabetical order, and 'r' reverses the order. By default, the command will operate on the default vam bank, but other bank locations may be specified. Use of wildcard is optional, and like in DOS, '*' will match any number of characters. For any single character, however, you need to use '^', instead of '?' as in DOS.

2.6.3 Creating, Assigning, Setting Defaults, and Closing a Vam File

To create a Vam file from a Vam configuration file, the command in *G7* is

vamcreate <vam configuration filename> <vam filename>

For example, to create the Vam file hist.vam from the configuration file vam.cfg, the command is

```
vamcreate vam.cfg  hist
```

The vamcreate command may be abbreviated to *vamcr*, or:

```
vamcr vam.cfg hist
```

At this point, the newly created Vam file has zeroes for all of its data. The following help pages deal with how to put data into it.

The first step is to assign it as a bank. The command is

vam <filename> <letter name of bank>

For example:

```
vam hist b
```

will assign hist.vam as bank b. Letters 'a' through 'z' (except 'w') may be used to designate banks. However, it generally is a good practice to leave 'a' as the *G7* bank which initially was assigned.

In order not to have to continually repeat the bank letter, most commands for working with Vam files use the default Vam file. It is specified by the dvam command

dvam <letter name of bank>

For example:

```
dvam b
```

A Vam file already must be assigned as a bank before it can be made the default. However, if several Vam files are assigned, the default can be switched from one to another as often as needed.

To review, the commands

```
vamcr vam.cfg hist
vam hist b
dvam b
```

will create an all-zero Vam file as specified by the VAM.CFG file, assign this file as bank b, and make it the default Vam file.

Assigning a Vam file to *G7* opens the file for writing as well as reading, though most commands will modify the Vam bank only if it is assigned as the default bank. The operating system ensures that no two programs can have the same file open for writing. Hence, if after looking at model results in the file dyme.vam, we decide to run the model again, we must first close DYME.VAM with the *G7 close* command. Thus the sequence might be

```
...
vam dyme c
gr c.out1
...
close c
```

and we are then free to run the model again and write to the DYME.VAM file.

2.6.4 Input of Time Series into Vam Files

There are a number of ways of loading data into Vam files. This help page shows ways to introduce a single time series for one element of a vector or matrix. Following pages show how to introduce whole vectors or matrices. Many of these tools are designed to read data from ASCII files. Additional capabilities are offered for reading data from Excel spreadsheets.

vdata <series_name>

This command works just like *data* except that the series introduced goes to the default Vam file. Recall that if "fd" is a vector in the Vam file, "fd23" will be the 23rd element of it. If "am" is a matrix, "am12.14" is the element in row 12, column 14.

(vup)date

This command works just like *update* except that the introduced series goes into the default Vam file.

Loading the Vam File from G7 banks

It also is possible to introduce data from *G7* banks into the Vam file. Just as the *f* command will form a variable and store it in *G7*'s workspace, the *vf* command will load a single series into a vector or matrix element in the default Vam file. A significant difference is that the vectors and matrices in the Vam file must be specified in advance, though without data, while new series may be introduced to the workspace at any time. The format is

vf <name> = <expression>
vf <name[{<date1> [- <date2>]}] > = <expression>
vf <name> <operator> <expression>

Examples:

```
vf out1 = out1
vf out1{2000} = out1
vf out1{2000-2010} = out1
```

This seemingly tautological example actually does something useful. It will look in *G7's* workspace bank for out1 and if it fails to find it, it will look in the bank assigned as 'a'. Let us suppose that it finds out1 there. It then will copy this series to the default Vam file. Specifically, the series will be copied to element 1 of the out vector in the default Vam file. The right side of the *vf* command can be any valid *G7* expression, including the various @ functions, such as *@log*, *@exp*, *@pos*, *@ifpos*, and *@csum*. To specify that a variable on the right should come from the bank assigned as b, put "b." in front of the variable name. The *vf* command works over the range of dates specified by the *fdates* command. The command can be carried out over an alternative set of periods by specifying the dates within brackets after the left-hand-side variable name is specified.

In addition to the assignment operator (i.e. "="), the *vf* command can add <expression> to the left-hand series using the "+=" operator, or it can subtract <expression> from the left-hand series with the "-=" operator, or multiply with "*=", or divide with "/=". These operators follow the syntax for the C++ programming language, though here they operate over a range of values.

fdates <fdate1> <fdate2>

This command specifies dates that will be used as the range of action of the *f*, *vf*, *vc*, *index*, *ctrl*, and other commands, and also of the *@cum()* function. The default values of the *fdates* are the beginning and ending dates of the Vam file, respectively.

Example:

```
fdates 1980 1999
```

2.6.5 Input of Vectors into Vam Files

vmatdata

The *vmatdata* command puts whole vectors into the default Vam file. It can be used to bring in data in a variety of formats commonly used by statistical agencies in releasing input-output tables. It can read a file that shows any one of the following:

- one vector in columns for several years
- one vector in rows for several years
- several vectors in columns for one year
- several vectors in rows for one year

The *vmatdat* command requires at least two lines and three if a format for reading is specified. The form of the first line is always the same, as is the form of the third line, if present. The second line has two different forms, one if several vectors are being read for one year and another if data for one vector is being read for several years. The form of the command is

vmatdat <r|c> <nv> <nyrs> <first> <last> [skip]
<year> <vec1> <vec2> . . . <vecnv>
vmatdat <r|c> <nv> <nyrs> <first> <last> [skip]
<vec> <year1> <year2> . . . <yearnyrs>
[form]

where

r|c
is 'r' or 'c' according to whether the vectors are rows or columns.

nv
is the number of vectors.

nyrs
is the number of years.

first
is the position in the vector in the Vam file of the first item in the data which follows.

last
is the position in the vector in the Vam file of the last item in the data which follows.

skip
is the number of characters on each line that should be skipped before beginning to read data. If no value for skip is provided, then a form line (explained below) will be expected as the third line. If no characters are to be skipped and no form line provided, then give skip a value of 0.

year
: is the year of the vectors or the first year if *nyrs* > 1.

vec1
: is the name of the first vector, "vec2" is the name of the second vector, etc.

form
: If no value for skip is given, then this form line must be provided. It should have the letter 'r' in every position that is to be read in the following table. This device allows direct reading of tables that have columns of data separated by '|' or other symbols or that have subtotals that are not to be read.

A typical form line and a matching data line are

```
form:                 rrrrrrrrrrrrrrrrrrr        rrrrr     rrrr
data: Agriculture     | 3450  3718  249   0  | 6780|  8102 |**| 1598
```

Columns that do not have an 'r' simply are ignored. Consequently, they cannot be used to separate data fields. Thus, reading the following

```
rrrrrrrr      rrrr
    851       2971
```

would produce one number, 8512971, not two numbers, 851 and 2971.

The first version of the second line is used if there are several vectors for one year; the second is used when giving data for one vector in several years. In both cases, the item of which there is only one is specified first and then the items of which there are several.

Any line beginning with a '#' is skipped completely.

For example, the lines

```
# Example of several vectors in columns for one year:
vmatdat c 6 1 1 57 3
1986 cons gov inv stk exp imp
1      21        0        2       -3         6          7
...    ...       ...      ...     ...        ...        ...
57     38       401        0        0         17          15
```

would read the 6 columns of data into positions 1 through 57 of the named vectors (cons, gov, . . .) for the year 1986. Positions 1 - 3 of each line, which contain the sector number for easy visual reference, are skipped.

Here is another example that reads the vectors as rows, as often may be the case when reading printed tables of primary inputs.

```
# Example of several vectors in rows for one year:
vmatdat  r 6 1 1 6 16
1986 labinc propinc deprec inter profit tax
#                 1     2     3     4     5     6
labor income    7303    21   368  1203  1302   820
proprietor inc  1215     9   100   107   303    92
depreciation     950     3    82   104   121    73
interest income  845     7    83    54    95    52
profits           50     3    25   217   337    38
indirect taxes   121     1    32   178   294    29
```

Note the use of the line beginning with a '#' to provide labels for the columns. The second form of the second line is illustrated by the following example that reads one vector, cons, for five years.

```
# Example of a single vector in columns for several years:
vmatdat c 1 5  1 57 2
cons 1985 1986 1987 1988 1989 1990
 1   31.8 37.2 32.5 38.7 41.2 43.1
....
57   1.2  1.7  1.8  2.0  2.1  2.2
```

Finally, here is an example that reads a single vector in columns for several years.

```
# Example of reading a single vector in columns for many years.
vmatdat r 1 28 1 8  12
demog 1989 1990 1991 1992 1993 1994 1995 1996 1997
# Date  ncent south  west college  twoy  fs1   fs2   fs5   head1 head3
89.000  0.243 0.345 0.208  0.220  0.469 0.243 0.323 0.106  0.274 0.350
90.000  0.241 0.346 0.209  0.222  0.476 0.250 0.320 0.105  0.267 0.350
91.000  0.240 0.347 0.211  0.224  0.482 0.247 0.320 0.105  0.263 0.348
92.000  0.238 0.348 0.212  0.226  0.487 0.245 0.319 0.104  0.260 0.346
93.000  0.237 0.349 0.213  0.227  0.493 0.244 0.319 0.102  0.257 0.345
94.000  0.236 0.349 0.215  0.227  0.495 0.243 0.321 0.106  0.257 0.340
95.000  0.233 0.351 0.218  0.228  0.507 0.252 0.320 0.102  0.257 0.343
96.000  0.233 0.351 0.219  0.229  0.509 0.253 0.320 0.101  0.257 0.342
97.000  0.232 0.352 0.219  0.231  0.511 0.254 0.321 0.099  0.256 0.342
```

It must be emphasized that, precisely because it can read in free format, the *vmatdat* command cannot interpret blanks as zero entries. There must be a numerical value for all cells in the tables, including the 0's.

The flexibility of *vmatdat* often makes it possible to read final demand columns or primary inputs as they appear in printed tables or on the disks provided by statistical offices.

For large models, it is convenient to be able to read "folded" vectors and store them in the default Vam file. This is the function of the *fvread* command. The usage is:

fvread <vector_name> <year> <skip>

> <skip> is the number of columns to skip at the beginning of each line. There must be present as many elements as are in the vector, and they must be in order.

Example:

```
fvread fd 1997 10
finaldem 1    23 36 83 92 32
finaldem 6     8 23 73 80 75
```

This example will put the vector data (23, 36, 83, 92, 32, 8, 23, 73, 80, 75) into the fd vector of the default Vam file for the year 1997.

2.6.6 Input of Matrices to Vam Files

There are four commands to introduce into the default Vam file a matrix from an ASCII file. The first two use different input formats to put matrices into the Vam file. The other two put data into packed matrix files, in which the cells that are zero in all years have been eliminated to conserve space.

The first, and most used, command is *matin*. It has the form

matin <matrix_name> <year> <firstrow> <lastrow> <firstcol> <lastcol> [skip] [form]

A rectangular array of numbers must follow, with values matching the <firstrow>, <lastrow>, <firstcol>, and <lastcol> parameters of the command. As in *vmatdata*, the optional <skip> parameter is the number of spaces to be skipped in reading each line. The <skip> parameter and the <form> line work together exactly as described above for *vmatdat*; the absence of <skip> indicates the presence of a form line. A '#' in the first space of a line means to skip the whole line. Use of *matin* does not affect entries outside of the specified area, so it can be used to update a matrix as well as to introduce it originally.

It also is possible to introduce a matrix that is in a spreadsheet by a "Copy" and "Paste" operation into the window created by the *show* command described below. While this route may be a quick expedient, if the Vam file is changed and the process has to be repeated, it quickly loses its appeal.

The second command for reading matrices is solely for conversion of models previously built with the *Slimforp* program. If you are not converting a model from *Slimforp*, ignore this command. Its form is

matin5 <matrix_name> <year> [<width> <decs> <IndexWidth>]

It is followed by a matrix in the *punch5* format used in *Slimforp*. The end of the matrix is signaled either by a ';' in the first space on a line or by the end of the file. For those not familiar with the *punch5* format, it is a fixed length format with 5 matrix entries to a line, each entry of the form "<row> <col> <number>". One possible format in Fortran would be (5X,5(2I3,F9.5)). In this case, <width> = 9, <decs> = 5, and <IndexWidth> = 3.

Here is a sample from the beginning of an A matrix:

```
matin5  am 1977 9 5 3
# A-matrix for 1977. 83 rows and 83 columns.
AM   1  1 0.245790 1  4 0.000220 1  8 0.000410 1  9 0.281300 1 10 0.058360
AM   1 11 0.002070 1 12 0.005680 1 13 0.000380 1 15 0.001700 1 16 0.003060
AM   1 22 0.089770 1 24 0.000070 1 48 0.001210 1 49 0.000130 1 50 0.000030
```

Packed Matrices

There are two special commands for introducing data for a "packed" matrix. Packed matrices become important in models with many sectors, in which the matrices – especially the various bridge matrices – often are very sparse; that is, they have relatively few non-zero elements. *Interdyme* has the ability to carry the matrices in a packed form in which only the non-zero elements are stored. In most places in *G7*, the packed matrix can be used interchangeably with the "full" matrix. For example, in the *index*, *lint*, and *show* commands described below, either full or packed matrix names can be used without needing to remember which kind of matrix is being used. For the input of data, however, there is a special format for packed matrices. Moreover, packed matrices are not included in the Vam file. Only the file name of the packed matrix is in the Vam file; the actual data are stored in a special binary file for which we use the extension .PMX. This separation of the .PMX from the Vam file was done because (a) the .PMX files are fairly big and (b) they often do not change at all between various alternative runs of the model. Thus, we might have five runs of the model, each with its own Vam file but all with the same PMX files. The savings in disk space over having five copies of the .PMX files could be substantial.

The first command for introducing data into packed matrices assumes that the elimination of the zero elements already has been done, as likely is the case if the matrix comes from Inforum programs for updating a matrix. The second command reads the matrix in full rectangular form and eliminates the zeroes itself. The result is the same in either case. The first command is:

pmatin <matrix_name> <packed_matrix_filename>

where <matrix_name> is the name of a matrix in the VAM.CFG file which must have the letter 'p' in the "lags" position. The <packed_matrix_filename> is name the file name of the .PMX file. (The .PMX should be included in the specified name.)

The format of the data that follow is:

```
<year>
<row> <num_non-zero_elements>
<col1> <element1> <col2> <element2> ...
```

For example

```
pmatin bm bm.pmx
1985
1  5 # row 1 has 5 non-zero elements
1  .656  7 .352  11 .038      23 .019  27 .218
3  2 # row 3 has 2 non-zero elements
7  .098  51 .923
.... ;
```

If the matrix is available for several years, the data for all years should be introduced with only one *pmatin* command, with each year's data preceded by the number of the year. It is essential that the year numbers not be abbreviated; use "1987" please, not "87". If a cell is non-zero in any year for which there is data, the packed matrix will have a place reserved for it in all years. To signify the end of the data, provide a ';'.

pmatin1 <matrix_name> <packed_matrix_filename>

An alternative way to introduce data into packed matrices is the *pmatin1* command. The format is just like that of the *pmatin* command except that the data should be arranged in rectangular rows and columns like the *matin* command.

For example, if you wish to treat as a packed matrix a 3 x 3 matrix known as B in the Vam file, you can introduce it with the *pmatin1* command as follows:

```
pmatin1 B B.pmx
1995
  1  0  0
  5  0  1
  6  2  0
1996
 1.5 0  0
 6   0  2
 7   3  1
```

Note that in the above example, only those cells that are zero in all years will be left out of the packed matrix. Cell (3,3) will actually be stored, since it has a non-zero value in 1996. As with the *pmatin* command, you should put of the data for all years you want to read in the same file, and use only one *pmatin1* command per matrix.

pmfile <matrix_name> <packed_matrix_filename>

Although packed matrix files often do not change between runs, they may. If, for example, we wished to study the effects of changes in the A matrix and the A matrix was packed, then we would need two different .PMX files. If the original was called AM.PMX, then to create the alternative, say, AMALT.PMX, we would go to the DOS prompt via File | DOS and do

```
copy am.pmx amalt.pmx
copy hist.vam vamalt.vam
```

Then from *G7*'s command box type

```
vam vamalt b
dvam b
pmfile am amalt.pmx
```

This *pmfile* command both will assign AMALT.PMX to be used whenever the am matrix is referenced and will put “AMALT.PMX” into AMALT.VAM as the name of the file to be used for the packed am matrix.

pmmode { “simple”| “advanced” }

Note that the <packed_matrix_filename> setting is limited to 30 characters, including any path specification. Sometimes more than one vam bank is loaded in *G7*, where the vam banks include packed matrix files with the same name but that are stored in different locations. Sometimes it is not convenient to reset the link for each vam bank using the *pmfile* command, or perhaps inclusion of a full path specification would be needed to distinguish between the files but the full specification exceeds the limit of 30 characters. In such cases, it might be useful to specify a link to each packed matrix file relative to the placement of the corresponding vam bank. By default, *G7* will assume that the packed matrix file is in the current working directory, or, if a relative path is specified, that the specification is relative to the current location. This behavior can be controlled with the *pmmode* command. The *pmmode simple* command corresponds to the default behavior. The “advanced” mode instructs *G7* to attempt to find PMX files in the same location as the corresponding vam bank, or if a relative path is specified for the PMX file, the root location should be the location of the parent vam bank.

2.6.7 Display of Vam File Data

All of the *G7* commands for the display of data all work for data in any assigned Vam file; it simply is necessary to prefix the variable name with the bank's letter designation followed by a period. Thus

```
vam dyme b
ti Output, Export, and Import of Agriculture
gr b.out1 b.ex1 b.im1
type b.out1
matty b.out1 b.ex1 b.im1
```

will graph or type out the time series of the first element of the out, ex, and im vectors from the vam file assigned as bank b.

For vam files, however, there is another command that shows vectors and matrices as if one were in a spreadsheet program. This is the *show* command. For vectors, the format is

show <bank_letter>.<vector_name> [<first_row> [<first_year>]] [<-rt <filename>>] [<-ct <filename>>]

The values of the vector in successive years will appear as columns; dates run across the top and sector numbers down the side. Providing the <first_row> and <first_column> positions will scroll the window accordingly. Row and column titles are read from the files that are specified in the Vam configuration file for the bank loaded in the <bank_letter> slot. If the title files that are specified in the Vam file are not found, then generic row and column titles will be created and displayed. If an alternative set of titles should be displayed, and those titles are stored in text files, then the file names for these alternative row and column titles may be specified at the end of the show command; “-rt” or “rowtitle” (or “-ct” or “columntitle”) and the filename indicate the appropriate file.

The Options menu item allows the user to control the number of decimal places, fonts, and colors of the display. The display can be copied to the clipboard in the usual way for Windows programs. The contents of the clipboard then can be copied into a spreadsheet such as Excel or into a table in a word processor. Select the bottom right cell of the rectangle that you wish to copy. Then select Copy from the menu, and you get to specify how much you want copied – just the numbers or also the frame. It also is possible to go in the other direction, from the spreadsheet into the show window and thus into the Vam file.

For matrices, the show command has a slightly different format:

show <bank_letter>.<matrix_name> <view> <line> [<first_row> [<first_column>]] [-rt <filename>] [-ct <filename>]

The “view” argument must be one of the following:

r

<line> is the number of the row to be displayed. (A row is displayed as if it were a column.)

c

<line> is the number of the column to be displayed.

y

<line> is the year number.

Examples:

```
show b.am r 5
show b.am c 7
show b.am y 1997
```

Cutting and pasting works as with vectors. If <first_row> or <first_column> parameters are specified, then the display will scroll to the given row and column position.

Note that this feature is very similar to the *gridtype* command for displaying time series.

2.6.8 Vector Calculations

vc <vector_name>[{<date1> [- <date2>]}] = <expression>

The *vc*, or *vector calculation*, command evaluates the expression on the right and puts the results into the named vector. Only a few matrix and vector operations are implemented with *vc*. It can add or subtract any number of vectors and multiply a matrix times a vector. It cannot add or subtract matrices. Parentheses are not supported. However, scalar values (either constants or G bank variables) can be used to premultiply or postmultiply a vector or matrix. Also, a scalar value can appear all alone on the right hand side of a *vc* equation to indicate that the entire vector will be set equal to that scalar value. Between a matrix and a vector, an ‘*’ indicates matrix-vector multiplication. Between two vectors, the ‘*’ means element-by-element multiplication. Between a vector and a matrix, the ‘/’ means multiplication by the transpose of the matrix or vector on the left. Between two vectors, the ‘/’ means element-by-element division. Setting a vector equal to a time series stores the time series in each vector element. Limited capability exists for setting values for all periods equal to the value in a single year and to normalize a vector by its values in a single year; see the examples below. The *vc* command is performed only over the interval defined by the current *fdates* command.

Example: If am and cm are matrices and out, pdm, qcu, cprices, and index are vectors, we could write

```
vc out = am*out + out    # Matrix multiplication, vector addition
vc qcu = out*pdm         # Element-by-element vector multiplication
vc out = qcu/pdm         # Element-by-element vector division
vc cprices = pdm/cm      # This actually is pdm'cm.
vc pdm = pdm / pdm{2000} # Normalize prices to 1.0 in 2000.
vc index = index{2000}   # Set equal to values in 2000 for all periods.
```

The <vector_name> must be a valid vector name in the default Vam file; matrices and packed matrices are not allowed. All right-hand-side vectors and matrices must be in the default Vam file; bank letters are not allowed.

The *vc* command is the only tool available for vector calculations that allows general (though limited) operations on the right-hand-side. Note that the *vf* command allows far more flexibility, but it operates on just one element at a time. While *vc* operates only on vectors, several standard matrix operations are offered. The following technique provides another means of copying vectors and matrices, in whole or in part, and allows addition and subtraction.

<vmake|vplus|vminus> [startyr [endyr]] <target> <source> [source . . .]

The *vmake*, *vplus*, and *vminus* commands construct a vector out of another vector or part of a matrix, or make a matrix out of one or more vectors and/or other matrices. *vmake* replaces selected values in the target with the values from the source; *vplus* adds the values in the source to the values is the target; and *vminus* subtracts the values in the source from the values is the target. Up to 200 sources may be taken from one or more assigned Vam files.

startyr
endyr

Optional parameters. If not given, then the process will work over the range of dates common to the target Vam file and all the source vam files.

target

A matrix or a vector.

source

Sources are one or more matrices and/or vectors.

Each matrix, whether a target or a source, is represented by

MatrixName [(<r|c> [lines])[(<c|r> elements)]]

r|c indicates whether the data will be taken by row or by column, lines the selected rows or columns, and elements the items selected within the specified rows or columns. The total number of lines selected from the sources must equal the number of lines selected from the target, and the number of elements selected from each source must equal the number of elements selected in the target. If lines or elements are omitted, the default is all rows/columns or all elements.

Each vector, whether a target or a source, is represented by

VectorName [(v elements)]

where elements are the items selected within the vector. If elements are omitted, the default is all elements. The number of elements selected from each source must equal the number of elements selected in the target.

Here are some examples:

The first example replaces all elements in rows 1, 2, and 3 of matrix ‘A’ with all elements in vectors ‘a’, ‘b’, and ‘c’.

```
vmake A(r 1-3) a b c
```

The second example adds the values of all elements in vectors ‘a’, ‘b’, and ‘c’ to the first five elements of columns 1, 2 and 3 in matrix ‘A’ (assuming that vectors ‘a’, ‘b’, and ‘c’ each have five elements.

```
vplus A(c 1-3)(r 1-5) a b c
```

If vectors ‘a’, ‘b’, and ‘c’ had more than five elements, the command would have to be

```
vplus A(c 1-3)(r 1-5) a(v 1-5) b(v 1-5) c(v 1-5)
```

The next example subtracts column 3 of matrix ‘A’ from the vector ‘c’.

```
vminus c A(c 3)
```

The final example is more complicated. It puts all rows of matrix ‘B’ (starting from row 1) followed by the vector ‘c’ into columns 2-100 of the matrix ‘A’, thus transposing the data in the sources.

```
vmake A(c 2-100) B(r) c
```

Related Topics: *vmake*

2.6.9 Matrix Calculations in VAM files

G7 offers basic matrix operations on matrices, one or two at a time, but not yet long expressions involving many terms. For other vector operations, see the *vc*, *vmake*, and related commands.

The available commands are

mcopy
: Copy one matrix to another; the two do not need to be in the same VAM file. The routine also can copy vectors.

madd
: Add or subtract two matrices in the default Vam file.

mmult
: Multiply two matrices in the standard matrix algebra way, perform element-by-element multiplication or division, multiply two matrices after transposing the first, select the greater or lesser elements of the two matrices, or multiply or divide matrix elements by a constant.

minv
: Invert the matrix.

linv
: Form the Leontief inverse of a matrix.

mtrans
: Form the transpose of a matrix.

The operations can be limited to a single year or can operate over the range specified by *fdates*. Here are the details of the commands.

mcopy [bank letter.]<destination> [bank letter.]<source> [year]

Examples:

```
mcopy  AM = FM 2000
```

Copies the FM matrix for the year 2000 from the default VAM file into the AM matrix in the same file.

```
mcopy AM = FM
```

Same as above, except that the copy is done for all the years in the *fdates* range.

```
mcopy AM = c.FM
```

Copies the FM matrix in the VAM file assigned as bank c to the AM matrix in the default VAM file for all years in the *fdates* range.

```
mcopy d.AM = c.FM 1995
```

Copies the FM matrix from the VAM file assigned as bank c to the AM matrix in the VAM file assigned as bank d. The copy is performed for 1995 only.

If AM and FM in these examples do not have the same dimensions, the copy is done the smaller number of rows and the smaller number of columns. If the optional year is not provided, then the operation is performed over the range specified by the *fdates*.

All of the other matrix commands operate only on matrices in the default VAM file.

madd <A> = < B> + <C> [year]
madd <A> = <B> - <C> [year]

Matrix addition or subtration of matrices, all in the default VAM file. If the optional year specification is absent, the operation is done over the *fdates* range.

mmult <A> = <B>*<C> [year]

Matrix algebra multiplication of matrices 'B' and 'C'. Matrices must be conformable and dimensions of 'A' appropriate for the product. Alternatively, B or C may be a constant.

mmult <A> = <B> ' <C> [year]

Matrix 'B' is transposed before matrix multiplication.

mmult <A> = <B> & <C> [year]

Element-by-element multiplication of matrix 'B' by matrix 'C'. Alternatively, B or C may be a constant.

mmult <A> = <B> / <C> [year]

Element-by-element division of matrix 'B' by matrix 'C'. Alternatively, B or C may be a constant.

mmult <A> = <B> < <C> [year]
mmult <A> = <B> > <C> [year]

Assign elements of A to the lesser or greater values of elements in 'B' and 'C'.

mtrans <A> <B> [year]

Place the transpose of 'B' into 'A'.

minv <A> [year]

Replace 'A' with its inverse.

linv <A> [year]

Subtract the 'A' matrix from the identity matrix, invert the result, and replace 'A' with the result.

2.6.10 Groups of Sector Numbers

We already have seen how, with the *add* and *do* commands, it is useful to be able to specify a group of integers as arguments. In the *lint*, *index*, and *ctrl* commands which we are about to explain, this concept of a group of integers is carried further. For use in these commands, the group is specified first and then used in the commands. *G7* has a dynamic group that can be specified and respecified over and over as necessary. It also can use the static groups as are defined in the *Fixer* program.

The dynamic group is defined by the command:

group <group definition>

Define the content of the dynamic group. The sector numbers are specified as in the *add* command.

Example:

```
group 19-25 (20 22)
```

The name of this dynamic group is ':'. The names and content of the static groups defined in *Fixer* are preceded by a ':' in the following commands.

Named groups also can be introduced with the *group* command. The format is:

group <group name>

<group definition>

This version of the *group* command adds a named group to the GROUPS.BIN file. This group then can be used in other commands requiring group expressions by prefixing the group name with a colon ':'. The group definition include a sequence of integers, a range of integers, a set of letters to specify spreadsheet columns, and other named groups. To exclude a set of values from the group definition, simply include the set to be excluded in parentheses.

For example:

```
group Manufacturing   # All manufacturing sectors
1-58
f empmfg = @csum(emp, :Manufacturing)
group NonchemicalMfg  # All manufacturing sectors except chemicals
:Manufacturing (20-27)
```

listgroups

List the names of all of the groups currently in the GROUPS.BIN file.

glist <name>

List the sectors in a group. The <name> specifies the group name.

After the above group command, the command

```
glist :
```

would give the answer

```
19 21 23 24 25
```

If the *listgroups* command gives the answer

```
Ag Min Mfg Trans Trade Util Serv
```

Then

```
glist Ag
```

might, for example, give an answer like

```
1  2  3  4
```

2.6.11 The Resector Command

The purpose of these tools is to provide a convenient way to aggregate and disaggregate data from various sectoral levels. The routine begins by reading a file that contains concordances between various sectoral aggregation schemes. The column heading on the first line must specify the number of sectors in that aggregation scheme. This number of sectors is also the "handle" that is used to refer to that column in the resector routines.

Please see the demo section of the Inforum web site for examples of the *resector* capabilities.

rs create <filename> [<Number_of_Columns> <Maximum_Number_of_Sectors>]

The *create* function opens the key file and reads in up to 20 columns of aggregation information (6 is the default). An optional argument can be supplied to limit the reading to more or less columns than the default. In principle, the routines handle aggregation or disaggregation between any two of the schemes read from the file. With six aggregation schemes, this results in 36 possible combinations of aggregation.

The beginning of a sample file is displayed below:

```
575  495  360  432   85  BEA82
  1    1    1    1    1  10100   Dairy farm products
  2    2    2    2    1  10200   poultry and eggs
  3    3    3    3    1  10301   Meat animals
  4    4    3    3    1  10302   Miscellaneous livestock-horses,bees,ho
  5    5    4    4    1  20100   Cotton
  6    6    5    5    1  20201   Food grains: wheat,rye,rice,buckwheat
  7    7    5    6    1  20202   Feed grains: corn,oats,barley,hay,sorg
  8    8    5    6    1  20203   Grass seeds
  9    9    6    7    1  20300   Tobacco
 10   10     7    8    1  20401   Fruits
 11   11     7    8    1  20402   Tree nuts
 12   12     7    9    1  20501   Vegetables
 13   13     7   10    1  20502   Sugar crops
```

rs formagg <Number_From> <Number_To>]

This is the first function that should be called after creation. It performs the most important initialization tasks. It sets up all of the information that is needed to aggregate from the scheme indicated as <Number_From> to the scheme indicated as <Number_To>. It sets up concordance lists and "split lists" that will be used by other functions listed below. In some programs, you may need to issue this command several times, especially if you need to use some of the "cross-aggregation" techniques described below.

One of the functions of initialization is to set up lists of correspondences between sectors, as well as lists of splits, where one sector from one scheme corresponds to one or more sectors from the other scheme. Since this initialization is time and memory consuming, an explicit function called *formagg* performs this task, and this function is called for only needed aggregation relationships. *formagg* takes as its arguments the maximum sector numbers of the source and destination schemes. Until *formagg* has been called with a certain aggregation pair, no other functions using that pair are allowed.

rs aggvector <Number_From> <Number_To> <Input_Vector> <Output_Vector> <Split_Vector>]

This function is designed to aggregate or split a vector from one aggregation scheme to a vector of another aggregation scheme. <Number_From> should be the number of sectors of the source vector, and <Number_To> should be the number of sectors of the destination vector, as shown in the heading in the key file that was read from the create routine. <Split_Vector> must be of the same aggregation level as the destination vector. It is used by the routine where there is a one-to-many relationship going from source to destination sectors. If you purely are aggregating, the <Split_Vector> will not be used and its values may be set to arbitrary levels.

rs ctrlvec <Number From> <Number_To> <Detailed Vector> <Aggregate_Vector>

This function controls a detailed vector to a more aggregate vector.

rs rdctrlvec <Number_From> <Number_To> <Detailed Vector> <Aggregate_Vector>

This function controls a detailed vector to a more aggregate vector, using right-direction scaling.

rs crossaggvector <Number_From> <Number_To> <Number_In_Between> <Input_Vector> <Output_Vector> <Split_Vector>

Sometimes conversion of a sectoral scheme is more complicated than merely a combination of aggregations and splits. For example, there may exist a many-to-many relationship, where a <SplitVector> of either sectoring level would not provide enough information on how the flows should be allocated. The name "cross-aggregation" indicates the method of translation used by this function, whereby the source vector is split to the level of a more detailed intermediary vector, which then can be aggregated directly to the destination vector. It must be possible to aggregate the intermediate sectoring level both to the source and to the destination levels for this function to work.

In the argument list, <Number_From> is the number of sectors of the source vector, <Number_To> is the number of sectors of the destination vector, and <Number_In_Between> is the number of sectors of the intermediary vector. <Input_Vector> is the source vector and <Output_Vector> is the destination vector. In this function, <Split_Vector> is a vector of length <<Number_In_Between>> that is used to split the source vector.

rs aggmatrows <Number_From> <Number_To> <Input_Matrix> <Output_Matrix>> <Split_Vector>

rs aggmatrows <Number_From> <Number_To> <Input Packed Matrix> <Output Matrix> <Split_Vector>

This function is similar to *aggvector* but aggregates the <Input_Matrix> by row to obtain the <Output_Matrix>. The routine also will accept a packed matrix as the input, though the <Output_Matrix> must be a full matrix.

rs aggmatrix <Number_Row_From> <Number_Row_To> <Number_Column_From> <Number_Column_To> <Input Matrix> <Output Matrix>

This function aggregates a matrix both by rows and columns. Earlier editions of the routine required both the source and destination matrix to be square, and the same aggregation mapping was applied to both dimensions. The current routine operates on rectangular matrices, and different aggregation mappings may be applied to the rows and columns.

rs ctrlmatrows <Number_From> <Number_To> <Input_Matrix> <Output_Matrix>

rs ctrlmatrows <Number_From> <Number_To> <Input Packed Matrix> <Output Packed Matrix>

This function controls a more detailed matrix to an aggregate matrix.

rs ctrlmat <Number_From> <Number_To> <Input_Matrix> <Output_Matrix>

rs ctrlmat <Number_From> <Number_To> <Input Packed Matrix> <Output Packed Matrix>

This function controls all cells within a block of the more detailed matrix to a single cell of the less detailed matrix. The function handles all of the different cases: single cell to single cell, row vector to single cell, column vector to single cell, and sub-block to single cell. The function is useful for updating a more detailed matrix such as a benchmark IO table to a more aggregate (and more current) matrix.

rs crossaggmatrows <Number_From> <Number_To> <Number In Between> <Input_Matrix> <Output_Matrix> <Split_Vector>

This function works much like crossaggvector but aggregates rows of a matrix.

2.6.12 Linear Interpolation of Vectors and Matrices

Linear interpolation of missing values in a vector is performed by the *lint* command.

lint <names>

lint replaces zeros and missing values in the time series of the vector or matrix elements with linearly interpolated values. Zeroes before the first or after the last observation are not replaced. This command applies only to series in the Vam file. Groups may be used in the names field to refer to sectors of the currently loaded vector.

Example:

```
lint pce1
```

This example loads the pce vector and then fills in the missing values in sector 1 by linear interpolation. We similarly could fill in the values for all the sectors in the current dynamic group by

```
load pce
lint :
```

or all the values for the static group Ag by

```
load pce
lint :Ag
```

Entire matrices may be interpolated with one command. For example,

```
lint am
```

will interpolate the entire am matrix. This interpolation also works for packed matrices.

The *lint* command works on the entire range of the Vam file without regard to *fdates*.

2.6.13 Moving Vectors and Matrices by Indexing

A quick way to fill in values of a vector so that they all grow at the same rate is to use the *index* command. It is used in conjunction with a guide series, a previously established single-variable time series, whose growth rates will be applied to each element of the vector. It operates over the range currently specified by the *fdates*. The form is

index <base_date> <guide> <vector_name>

For example, if "years" is the name of a series that contains years – 1980, 1981, etc. – then to make all elements of the vector pce grow by 2.0 percent per year from 1999 to 2010, the following commands can be used:

```
f  g02 = @exp(0.02*(years - 1980))
index 1999 g02 pce
```

The use of groups is allowed in the names of series to be affected. For example, to move only the sectors 1,7, and 9 of pce, we could – recalling that ':' is the name of the dynamic group – do

```
group 1 7 9
load pce
index  1999 g02 :
```

To move just the sectors in the static group Mfg, the command would be

```
load pce
index 1999 g02 :Mfg
```

With a specific "vector + sector" name like exp2, the command would be

```
index 1999 g02 exp2
```

to store the currently-loaded vector, load the exp vector, and move the series exp2 by the index of g02.

A vector or matrix name not followed by a sector number means to move all vector or matrix elements by the index. This device can be used to project an entire input-output matrix, say, am, to be constant, as in this example:

```
fdates 1998 2020
f one = 1
index 1998 one am
```

A matrix name followed without space by a sector number means to move that row of the matrix by the guide. Thus, am2 means move the second row of the am matrix by the guide.

If <guide> is the name of a vector and <name> is a matrix, then the rows of the matrix will be indexed by the corresponding elements of the vector. If an element of the vector is zero in the base year, the corresponding rows of the matrix are unchanged. This feature works both for standard and packed matrices. It can be used to apply across-the-row coefficient changes to an input-output matrix. For example, if movers is a vector, we can make each row of the am matrix follow the index of the corresponding element of movers by

```
fdates 1998 2020
index 1998 movers am
```

In applying this sort of across-the-row coefficient change, one usually does not want it to apply to the diagonal elements, for they have little to do with the technological changes affecting other coefficients. To avoid changing them, the following two commands are convenient.

diagextract <matrix_name> <vector_name>
Extract diagonal elements from the matrix and put them into the named vector. The command works both for regular and packed matrices.

diaginsert <matrix_name> <vector_name>
Insert elements in the named vector into the diagonal of the matrix. The diagonal element will remain zero for a packed matrix that has zero coefficients in that position for all years.

This example combines these last three commands

```
fdates 1998 2020
diagextract  am  diags
index 1998 movers am
diaginsert  am diags
```

2.6.14 Methods of Scaling

Scaling is used to force the elements of a set of numbers to add to a given control total. Desirable properties for any scaling algorithm are that:

1. the components of the set to be scaled should preserve their original rank ordering,
2. all elements of the set should be scaled in the same direction, and
3. the elements should be changed in proportion to their absolute value, which represents their weight or relative importance.

Furthermore, these characteristics should hold when the elements of the set to be scaled contain both positive and negative numbers and when the control total for the sum of the elements is zero.

The three following methods of scaling are used in various *G7* commands:

A. Ordinary Scaling

   Ordinary scaling uses a multiplicative scale factor s that is the control total c divided by the sum of the elements in the set, as follows:

$$s = \frac{c}{\sum x_i}$$

   so that the scaled value of each element of the set equals:

$$x_i^* = s \times x_i$$

   When all of the elements of the set and the control total are greater than zero, or when all of the elements of the set and the control total are less than zero, then all of the desirable properties hold true.

However, when the set contains both positive and negative numbers, the positive and negative elements will move in opposite directions with the above procedure and so the second desirable property listed above will not hold. For example, if the control total is greater than the sum of the elements in the set, the scale factor is greater than one. Thus, the scaled values of all elements greater than zero will be larger than their original values, but the scaled values of all negative elements in the set will be less that their original values. Similarly, if the control total is less than the sum of the elements in the set, the scale factor is less than one; positive original values will be made smaller while negative original values will be made larger.

Even worse, when all of the elements are positive and the control total is negative (or vice versa), the rank order of the elements is reversed. In this case, property (a) fails to hold.

This ordinary method of scaling is used in the *ctrl* command and the standard RAS routine.

B. Proportional Scaling

Proportional scaling uses a set of additive scaling terms, a, as follows:

$$a_i = \frac{|x_i|}{\sum |x_i|}\left(c - \sum x_i\right)$$

where $(c - \sum x_i)$ is the difference between the desired total and the actual sum – or the total amount to be allocated – and $(x_i / \sum x_i)$ is the share to be allocated to the ith element of the set. Thus, the scaled value of each element of the set equals:

$$x_i^* = x_i + a_i$$

Note that unlike the ordinary scaling factor (which is the same for each member of the set), the proportional scale term is different for each element.

The proportional scaling algorithm will scale any set of real numbers to any real control total, but the desirable characteristics for a scaling algorithm do not hold for all possible values of the control total. If $(\sum x - \sum |x|) < c < (\sum x + \sum |x|)$, both positive and negative elements in the set retain their sign and their rank order is unchanged. If the control total lies outside this range, the following tabulation gives the results of proportional scaling for various conditions:

- If $c > (\sum x + \sum |x|)$, the negative elements in the set become positive and their rank order is changed.
- If $c = (\sum x + \sum |x|)$, the negative elements in the set become zero and their rank order is changed.
- If $c = (\sum x - \sum |x|)$, the positive elements in the set become zero and their rank order is changed.
- If $c < (\sum x - \sum |x|)$, the positive elements in the set become negative and their rank order in changed.

Note that the proportional scaling algorithm will show an error message if one of these conditions is true.

This method of scaling is used in the *scale* command, the *ctrl* command, the ordinary RAS routine, and the gross value RAS routine.

C. Right-Direction Scaling

Right-direction scaling is a second method of scaling that preserves the relative ordering of the elements and scales them all in the same direction. Assume that A is the sum of all positive elements of the vector X, B is the sum of all negative elements, and C is the control total.

The problem is formulated to find the number S for which the following equation is true:

$$A \times S + B/S = C$$

This can be rewritten as a quadratic equation in S:

$$A \times S \times S - C \times S + B = 0$$

To solve for S, we use the general quadratic formula to obtain:

$$S = \frac{C + \sqrt{C \times C - 4 \times A \times B}}{2 \times A}$$

Note that positive and negative elements of the set are scaled by different values; the positive elements are multiplied by S, and negative element are divided by S.

This method of scaling may be used in the *scale* command and in the ordinary RAS command.

2.6.15 The Scale Command

The *scale* command uses proportional scaling (or optionally, right-direction scaling) to control a vector, or part of a vector, or a row or column of a matrix to a given control total. The command is:

scale <control> <vectorname> [(<group>)] [<year>] [rdscale]
scale <control> <matrixname> <r rnum> [(<colgroup>)] [year] [rdscale]
scale <control> <matrixname> <c cnum> [(<rowgroup>)] [year] [rdscale]

The <control> may be a series from any bank (e.g. a.ctrl), or from any vector (e.g. b.vec2), or from any matrix (e.g. c.mat1.1) that contains the values to which the vector is to be scaled.

vectorname
A vector whose elements are to be scaled.

group
Optionally specifies which elements of the vector are to be scaled.

year
Specifies a single year for which the elements of the vector are to be scaled. If year is not specified, the control is imposed over each period covered by the current *fdates*. :”rdscale”: Specifies that right direction scaling is to be used.

matrixname
The name of a matrix with a rows or columns that is to be scaled.

r rnum
c cnum
Specifies that a row or column is to be scaled and provides the number of the row or column.

colgroup
rowgroup
Optionally specifies which elements of the column or row are to be scaled.

To scale a group of time series that have a common root name (such as exp1, exp2, . . . , expn), us the *ctrl* command.

2.6.16 The Ctrl Command

The *ctrl* command uses proportional scaling to scale a group of series in the workspace or in a lettered bank that have a common root name (such as exp1, exp2, . . . , expn) to the value of a control variable. The command is:

ctrl <x> <rootname> <group>
where

x
is the control variable.

rootname
is the first part of the name of a group of sectors.

control
is imposed over the period specified by the current *fdates*.

For example, the call

```
ctrl tot out (1-10 (4-7) 13 15)
```

would impose the values of series tot as a control total on the group of sectors whose root name is out, i.e. sectors 1 to 10, except for 4 through 7, and then 13 and 15.

Note that all results will be stored in the workspace. For example, if the out vector in bank a is controlled, the resulting scaled series out1, out2, . . . , will be added to the workspace but the vector in bank a will not be modified. To store the results in the vector, use the *vf* command.

To scale a vector, or part of a vector, or a row or column of a matrix to given levels, use the *scale* command.

2.6.17 Methods of Balancing Matrices

In input-output analysis, one frequently needs to balance a table so that a set of initial row values add to known or estimated row sums and columns values add to known or estimated column sums. In *G7*, there are two methods of achieving this balance – one for matrices in general, and one for entire input-output tables.

A. For matrices in general, if the row and column sums are known, or if the estimated totals can be accepted, the ordinary RAS method of balancing a matrix should be used. This method of balancing may be used with any of the methods of scaling discussed above.

   - If all of the elements of the matrix are greater than or equal to zero and if all of the totals are greater than zero, ordinary scaling may be used.
   - If all of the elements of the matrix are less than or equal to zero and all of the totals are less than zero, ordinary scaling may be used.
   - If the matrix contains both positive and negative elements, or if the totals contain both positive and negative numbers, proportional scaling (or optionally, right-direction scaling) should be used.

   However, if the estimated totals are questionable, or if the ordinary RAS method will not balance the matrix, or if the method forces unacceptably large changes on some cells, the special use of proportional scaling RAS outlined in the section Balancing Matrices with Uncertain Totals might be useful. It allows the questionable row or column sums to change.

B. For entire input-output tables, the gross value RAS method of balancing a matrix should be used. It also can balance matrices with known or questionable control totals.

2.6.18 Matrix Balancing by the Ordinary RAS Method

In work with input-output analysis, one frequently needs to balance a table to known row and column totals. Three methods of scaling are offered in *G7*: ordinary RAS, proportional scaling RAS (*psras*), and right-direction RAS (*rdras*). In additional, outside information may be imposed using the "precondition" features described below.

The RAS commands are:

ras|psras|rdras [<-f|c>] <matrix>[(r rgroup)][(c cgroup)]<rowctrl><colctrl> [yr][r|c][-maxiter][-precon] [-tolerance]

where:

-f|c
specifies the flow (default) or the coefficient mode,

ras|psras|rdras
specifies the method of scaling to be used,

matrix
is the name of the matrix whose elements are to be balanced,

(r rgroup)
(c cgroup)
are options that determine which rows and columns are included,

rowctrl
colctrl
are vectors that contain the row and column totals,

yr
optionally specifies the year for which the matrix is to be balanced. If a year is not specified, the matrix is balanced over the period specified by the current *fdates*.

r|c
optionally specifies whether the sum of the row totals or the sum of the column totals controls the calculations if the two are not the same.

-maxiter
optionally determines maximum number of iterations. The default is 100.

-precon
optionally specifies that name of a file containing the preconditions that are to be imposed.

-tolerance
optionally specifies the tolerance for the convergence test. The default value is 0.000002.

If one or more of row and/or column totals are questionable, the matrix still may be balanced by a special use of proportional scaling RAS, which is explained in Balancing Matrices with Uncertain Totals.

Related Topics: *coef*, *flow*, *psras*, *rdras*

2.6.19 Balancing Matrices with Uncertain Totals

With the special use of proportional scaling RAS outlined below, it is possible to balance a matrix for which one or more row and/or column total is not known with reasonable certainty. The method, which allows these uncertain totals to change during the balancing process, (a) opens up a row and/or column of zeros and (b) subtracts the uncertain totals from the final row or column (leaving a zero in the final row or column) and from the corresponding position of the new row or column of zeros. As the uncertain totals now are inside the matrix, they are treated as part of the matrix and will change as the matrix is balanced. Finally, it is possible to use the precondition option in the *psras* command to control the change in any of the uncertain totals. This command depends critically on the ability of the proportional scaling algorithm to scale rows and/or columns that sum to zero.

The example that follows will make the procedure more clear. It uses a 5x4 matrix of factor payments from a three-sector input-output table (in which total value added is in the last row and total factor payments is in the last column) and subsequently published national accounts data for total value added.

First, expand the matrix to five columns and insert a column of zeros as column four; then move the total factor payments (now in column five) to column four and change their sign.

```
           Primary  Secondary   Tertiary   FactPay  Total
EstDep        58.5       563.7      409.0  -1031.2    0.0
EstWages    1297.9      1759.9     1096.2  -4154.0    0.0
EstTaxes      43.3       694.1      287.1  -1024.5    0.0
EstSurplus    74.5       943.2      343.0  -1360.7    0.0
NatActVA    1421.1      3722.3     2302.9  -7446.3    0.0
```

The matrix now is ready to balance. The command is:

psras [<-f|c>] <matrix>[(r rgroup)][(c cgroup)]<rowctrl><colctrl>[yr][r|c][-maxiter][-precon]

or in this example:

```
psras  VA (r 1-4)(c 1-4) VA 5 VA 5 1997 r
```

To see how it works, download and run the script SPECIALPSRAS.ADD available on the *G7* demo page of the Inforum web site.

2.6.20 Balancing Entire Input-Output Tables

The gross value RAS command can balance an entire input-output table by taking advantage of the ability of the proportional scaling algorithm to handle (a) the elements of sets that contain both positive and negative numbers, and (b) the elements of sets whose control total is zero. The *gvras* command can balance matrices with uncertain totals.

The command is:

gvras <matrix> <nsec> <nva> <nfd> <ngv> <yr> [maxiter] [precon]

where:

matrix

is the name of the matrix whose rows and columns are to be balanced.

nsec

is the number of sectors in the loaded vector. For example, use - : for the sectors in the present dynamic group - :Mfg for the sectors in the static Mfg - 1 7 9 for the group composed of sectors 1, 7, 9

nva
is the number of value added sectors.

nfd
is the number of final demand sectors.

ngv
is the number of gross value sectors.

yr
is the year for which the I-O table is to be balanced.

maxiter
optionally is the maximum number of iterations. The default is 100.

precon
is an optional file that contains a list of preconditions.

The algorithm requires, first, that the I-O table is set up in an unusual form:

- Move the detailed value added totals from the intermediate input column to the gross output column.
- Move the detailed final demand totals from the intermediate input row to the gross output row.
- Delete and close the intermediate input column, the intermediate input row, and the total value added row.

Second, the I-O table must meet the standard requirements that the row total for each sector equals the column total for each corresponding sector and that the sum of the value added totals equal the sum of the final demand totals.

Additional information may be imposed using the precondition features described below.

2.6.21 Balancing Matrices with Externally Imposed Preconditions

A researcher who needs to balance a matrix often has data that are not part of the matrix itself, such as the value of – or part of the value of – particular cells, or the range within which the values must fall. The *precondition* function allows the use of these data to constrain the value of cells or groups of cells.

The function is invoked by the last parameter in the *ras*, *psras*, and *gvras* commands. This parameter specifies a file (such as VA.PRE) that contains a list of the cells that are to be constrained, the types of constraint to be applied, and the values to which they are to be constrained. These values can be expressed directly as a number or as any expression that the *G7* software can interpret, such as a series name from any bank or Vam file.

Precondition Commands

There are two types of preconditions. In the first type, a known value is removed from a cell and from the corresponding row and column sums; the matrix is balanced; and finally the value is restored to the cell, row sum, and column sum. In the second type, after each iteration a value or condition is imposed until the matrix can be balanced without changing the value or violating the condition.

A. Precondition commands that apply to single cells

eq <rownum> <colnum> <value|expression>
This command forces the value of the specified cell to equal the value or expression.

For example:

```
eq 7 12 127.5
```

forces the value of the 12th cell in the 7th row to equal 127.5. A second example is:

```
eq 11 8 0.5*c.VA2.3
```

forces the value of the 8th cell in the 11th row to equal half the value of the cell in matrix VA in Vam file c. This is done by (a) setting the specified cell to 0, (b) removing the value from both the row sum and the column sum, (c) balancing the matrix, and (d) restoring the value to the row and column sums and setting the cell equal to the value.

pt <rownum> <colnum> <value|expression>

This command preserves that part of the value of the specified cell to equal the value or expression. For example:

```
pt 7 12 51.3
```

This is done by (a) removing the value from the specified cell, the row sum, and the column sum, (b) balancing the matrix, and (c) restoring the value to the cell, the row sum, and the column sum.

max <rownum> <colnum> <value|expression>

This command specifies the maximum value allowed in the specified cell.

min <rownum> <colnum> <value|expression>

This command specifies the minimum value allowed in the specified cell.

B. Precondition commands that apply to groups of cells

sc <firstrownum> <firstcolnum> <lastrownum> <lastcolnum> <value|expression>

This command—which applies to part of a row or part of a column or a block of cells—scales the values in the specified cells to the value or expression provided. For example:

```
sc 3 1 3 5 c.GV3
```

The values in the 1st through the 5th columns of the 3rd row are scaled to the value in the 3rd row of the vector GV in Vam file c. A second example is:

```
sc 4 2 6 2 d.VA1
```

The values in the 4th through the 6th columns of the 2nd column are scaled to the value in the 1st row of the vector VA in Vam file d. A third example is:

```
sc 2 2 4 6 1000
```

In the block that includes the 2nd through the 6th columns of the 2nd through the 4th rows, the values are scaled to 1000.

scmax <firstrownum> <firstcolnum> <lastrownum> <lastcolnum> <value|expression>

This command scales a partial row, or a partial column, or a block of cells if the sum of their values is greater than the value or expression specified.

scmin <firstrownum> <firstcolnum> <lastrownum> <lastcolnum> <value|expression>

This command scales a partial row, or a partial column, or a block of cells if the sum of their values is less than the value or expression specified.

2.6.22 The Coef, Flow, Getsum, Load, and Store Commands

coef <matrix> <vec> [year]

Convert the named flow matrix to a coefficient matrix by dividing each column of the matrix by the corresponding element of the named vector. If no year is named, the command works over the *fdates* range.

flow <matrix> <vec> [year]

Multiply each column of the matrix by the corresponding element of the named vector. If no year is named, the command works over the *fdates* range. This command is the opposite of *coef* command.

getsum <matrix> <r|c> [<group>] <vector>

Get the sum of the rows or columns of a given matrix for the range of periods specified by the *fdates*. The first argument must be the matrix or vector for which we want to obtain the sums. The matrix is assumed to be stored in the default vam file unless a bank letter is provided. The second argument must be 'r' or 'c' (to obtain the sum by rows or by columns). If only a partial sum is desired, then the columns or rows are specified next. The last argument is the vector where the result is to be stored; this vector must be located in the default vam file. The sum is calculated and stored for all years, so that we cannot have mixed information in the target vector.

load <vector_name>

When *G7* is working on a vector in a Vam file, it pulls the whole time series for the vector into a sort of vector workspace. Usually, this is done automatically simply by referring to the vector or one of its elements. The *load* command enables the user to do this loading explicitly. It is used principally in connection with the *index* command.

For example:

```
load pce
```

store

Store the currently loaded vector back to the Vam file with the modifications that have been made. This *store* is automatic when a new *load* or implicit load is encountered. When a *quit* command is given with an unstored, loaded vector, the program will store the vector automatically.

2.6.23 Writing Vam File Data to ASCII Files

It sometimes is necessary to take data from a Vam file and print it to an ASCII file. For example, we may have used the Windows cut and paste commands to move data from a spreadsheet into the window of the *show* command, but we want an ASCII version of the file so that we will not need to repeat this manual procedure if we rebuild the Vam file from scratch. The following commands provide several ways to print matrix and vector data to files that easily can be read back by *G7* or, in some cases, by other programs.

pmpunch <filename> <matname> [precision]

The non-zero elements of the matrix <matname> are written into the named file <filename> as input for the *pmatin* command. The <precision> argument gives the number of decimal places. The years to be written are determined by the current values of *fdates*. This command especially is useful for converting a rectangular matrix in a Vam file into a packed matrix. One writes it out as an ASCII file with this command and then reads it in with the *pmatin* command.

punchvec <filename> <vecname> <startyear> <endyear> [<width> <precision>]
pv

Print a vector to a file for all sectors, for the years specified. The optional <width> and <precision> arguments again specify the field width and number of decimal places. The output file starts with a number of comment lines, telling the name of the vector, starting and ending years, and format information. Then one comment line gives the years as column headings. Each following line of the file contains the time series for one sector, with the name of the series followed by the time series of data.

punch5 <filename> <vecname> [<width> <precision> <IndexWidth>]
p5

Print a matrix or packed matrix to a file in *punch5* format. The years written are determined by the current value of *fdates*. Each year is written with a header that is a *matin5* command, which tells *matin5* also what field width, precision, and index width to use. (The <IndexWidth> parameter is the width of the row and column numbers in the file.) Thus, a file created with *punch5* can be read back into *G7* using the *matin5* command. This provides a convenient way to convert packed matrices into normal matrices. The *punch5* format prints 5 non-zero matrix cells to a line, with row number, column number, and cell value for each cell.

vp <vector_name> [r|c] [field_width] [decimal_points]
vp <matrix_name> <view><line> [field_width] [decimal_points]

Write the named matrix to the currently open "save" file. As you can see, this command has a format and options like those of the *show* command.

For example:

```
save am.dat
vp am y 2000  9  6
save off
```

Note that the use of save files in conjunction with the *matty* or *type* commands also are useful ways to capture Vam data in ASCII form file. Furthermore, since the *Compare* program can work with Vam files, you can use the *\gdata* command in that program to make neat ASCII file printouts of vector or matrix time series from a Vam file.

2.6.24 Titles for Vam Files

vtitle <title>

This command allows you to specify a title for the default Vam file. This title is displayed when using *Compare* and when the bank is loaded in *G7*. It is useful to record the time and date of creation of Vam banks. This can be achieved easily by using the keywords *%time* and *%date* in the title. *G7* will replace these keywords with the actual time and date, respectively.

2.6.25 Regressions on Industry Data

eqpunch <filename> | "off"

The *eqpunch* command is for writing equation results to a file in a tabular format. We follow the precedent of giving those files an .EQP extension, but of course, any file name will do. The use of the *eqpunch* command is analogous to the *ipch* command, and in fact, both often are used in the same regression .REG file. The *eqpunch* command sets up a file for writing by the *titpch* and *tpch* commands. The *titpch* writes a header for the table and the *tpch* command writes out regression results, one line per sector. See the documentation on the *tpch* command for a complete example.

titpch [-<options>] [<stub_len>] [str1 str2 . . . strn]

The *titpch* command defines a header line that will be provided for a regression table, and it defines the regression output that will be displayed on subsequent *tpch* commands. Before using either the *titpch* or *tpch* commands, an equation table punch file (.EQP) first must be opened using the *eqpunch* command. The <stub_len> is a number that indicates how long the "stub" or title for the line should be. You should ensure that the width you specify in the *tpch* command is the same so that the table will line up properly. See the example below for the *tpch* command. <str1>, <str2>, . . . , <strn> are strings to be printed to explain the coefficients. Therefore, <str1> might be "intercept", <str2> is the second explanatory variable name, and so on. The option string starts with a '-' character, followed by one or more of:

b

rbar-squared

o

rho

r

r-squared

s

see

d

double-line format

f

floating point f format, instead of *G7* format.

tpch [<sur_which>] <sector> [<"label">] [str_len] [(coef numbers)]

The *tpch* command writes a line of parameters to an equation table file (.EQP). In order to use this command, the file first must be opened with the *eqpunch* command and a header definition supplied by the *titpch* command. The arguments for this command are as follows:

sur_which

is used only when an equation has been estimated with the *stack* or *sur* commands so that the coefficients are in *rcoef1*, *rcoef2*, etc.

sector

is the number of the sector or category for which the equation is estimated.

label

is a sector or category title in quotes. If used, you also should specify the length that you want to be printed in the <str_len> argument

str_len

is the length of the sector or category label in the printout. Note that this should be equal to the stub length specified in the *titpch* command.

coef numbers

are used when there is a superset of regression parameters possible and each equation uses some subset of that.

Example:

```
eqpunch ven.eqp
# Note: 30 is stub length.  Be sure to use the same for tpch!
titpch -rsf 30 const use usedif
add vena.reg   1 "Oilseed farming"
```

(continues on next page)

(continued from previous page)

```
...
eqpunch off
```

[contents of vena.reg]

```
ti %1 %2
subti Inventory Change Regression
f usedif = use%1-use%1[1]
r ven%1 = use%1, usedif
#gr *
ipch ven %1 a
tpch %1 "%2"  30
```

punch <punchfile | “off”>

The *punch* command opens an equation file for writing with the *ipch* command. These files usually have the extension .EQN and are used by *InterDyme* models to construct an Equation object that contains regression parameters, values of rho, equation types, and other information for multisectoral regression equations. To close the file, use *punch off*.

The *punch* command usually is given at the beginning of a large regression add file. It often is useful to use the fadd command, along with an argument file that contains sector numbers, titles, equation types, starting dates and other information that changes with each sector. Within the body of the fadd, a regress command will be followed by an *ipch* command. This will estimate the regression and put the equation results into the equation file.

ipch [<which>] <label> <sector> [<type>] [<psn1>] . . . [<psnN>] [extra <var1> .. <varN>]

The *ipch* (*InterDyme punch*) command is used for creating equation files for use in building *InterDyme* models, where:

which
is the suffixed number.

label
is the name of the vector for which the equation is estimated.

sector
is the number of the sector to which the equation applies.

type
is a character (‘a’, ‘b’, etc.) to signal the type or form of the equation. The type should not be a numeral.

psn1
(optional) the column number in the matrix where the first regression coefficient belongs.

psnN
The column number in the matrix where the nth regression coefficient belongs.

At the end of the command, type the word “extra” (without the quotes), and then type several variable names that hold the values of extra parameters you would like to pass to the file. For example, for an investment equation that calculates capital stocks and replacement investment, a physical service life is needed to calculate the spill rate. The following code would write the physical life at the end of the parameter list:

```
f plf1 = 4.9
r eqi1 = out1, out1[1], out1[2]
ipch eqi 1 a extra plf1
```

The *ipch* command is not useful unless an equation file already has been opened with the *punch* command. See the *InterDyme* manual, Chapter 4, for more information on the *punch* and *ipch* commands and on detached-coefficient equations in *InterDyme*.

2.6.26 *Vam2Vam* – Selective Copying From One Vam File to Another

Vam2Vam is a utility program, run at the DOS prompt, that copies selected matrices and vectors from one Vam file to another. It often happens that one has left out of a Vam file some essential matrix or vector. It is easy enough to modify the VAM.CFG file and create a new, all-zero vam file with a place for the new matrix or vector, but how can data be copied from the old Vam file to the new? *Vam2Vam* is an answer. Or one discovers that a matrix or vector must be enlarged. How can the data in the old Vam file be copied into the new? *Vam2Vam* is an answer. Or a Vam file has been used in the preparation of data that has in it various matrices and vectors that were essential for preparing the data but that are not needed in the final model. How can we extract just the final product matrices and vectors into a new Vam file for the model? Use *Vam2Vam*. See also the G7 mcopy and vmake routines that may prove more convenient.

The program is invoked by the command:

vam2vam <source> <target> <list> <startdate> <stopdate>
where

source
is the file name, without the .VAM extension, of the source Vam file;

target
is the filename, without the .VAM extension, of the target Vam file;

list
is the name of a file with a list of the names of vectors or matrices to be copied. For example, to copy "abc" in the source to "def" in the target and "xyz" in the source to "xyz" in the target, the contents of the list file would be

```
abc def
xyz
```

Note that the default value of the name in the target is the name in the source, so we did not need to repeat xyz on the last line.

startdate
is the first year whose data is to be copied

stopdate
is the last year whose data is to be copied.

Note that *G7* now has the ability to copy vectors and rectangular matrices from one Vam file to another. See details on the *mcopy* command in the *G7* Help files or the *Reference Manual*.

2.6.27 *VamToG* – Creating a *G7* bank From Series in a Vam File

Like *Vam2Vam*, *VamToG* is a convenient way for getting data from a Vam file, in this case, into a *G7* "ws"-type bank. The operation of this program is controlled by a configuration file, named VAMTOG.CFG. Note that *G7* easily can copy data from a Vam bank to a "ws"-type bank, and use of *G7* might prove more convenient.

A sample file is shown below.

```
Root name of Vam File; hist
Root name of destination *G7* bank; imp
Starting Date for data transfer; 1972
Ending Date for data transfer; 1994
Base Year of *G7* databank; 1955
First period covered; 1
Maximum number of observations;60
Data requests ; im ex out def fpi fpe
```

In this particular example, the source Vam file is HIST.VAM and the destination *G* bank is IMP.BNK. The data from 1972 to 1994 will be transferred. The *G* bank will have a base year of 1955, starting period of 1, and series vectors of length 60. The vectors im, ex, out, def, fpi, and fpe will be transferred.

2.7 Other *G7* Documentation

This concludes the *G7 Help* files. Please see the G7 page on the Inforum web site for software updates and the latest documentation. In addition to the *Help* files, please also see the *G7 Reference Manual,* the *G7 Tutorial,* and the demonstration programs offered there.

You can find contact information for software support on the Inforum web site. Please send questions and concerns by email to the Inforum webmaster.

CHAPTER

THREE

G7 REFERENCE MANUAL

3.1 *G7* Commands: Symbols

:

A ':' at the beginning of a line begins a comment that extends through the next line beginning with a ':'. Text in comments does not cause any action by *G7,* except that the comments are printed to the screen when an add file runs if *addprint* is on.

Related Topics: Command Files, *add*

#

Any text on a line that follows a hash sign ('#') will be treated as a comment. In addition to providing descriptive text for an add file, the # can be a method of temporarily removing lines from an add file that you may want to try again later.

Related Topics: Command Files, *add*, ", :

"

This also is a comment character, if used as the first character of a line. Its use is deprecated, as the '#' and ':' comments are preferred.

3.2 *G7* Commands: A

ac <series> [<n>]

Calculate the autocorrelation function for the named series over the period specified by the latest *limits* command. The optional <n> is the maximum number of terms to be calculated; its default value is 20. The output shows the autocorrelation function and the autoregression coefficients determined by the Yule-Walker equations. Taking the rightmost elements of each line of the autoregression coefficients gives the partial autocorrelation function. The function also is placed into the workspace with a name derived by adding "_ac" to the variable's name. This facilitates graphing of the function. If the command was "ac vi 20", then the graphing command would be "gr vi_ac :0 20".

Example:

```
ac vin$ 11
```

Related Topics: ARIMA, *bj*

add <addfilename> [Arguments]

The add file is one of the most powerful features of *G7* and is used intensively by those who are familiar with the *G7* command set. Although the graphical user interface is a useful tool for teaching and for remembering features, the add file is necessary when you need to get large jobs done quickly, in unattended operation. Think of the add file as the batch file for *G7*. Only the basic syntax of the *add* command will be described in this section, as a complete discussion of its use, with examples, is provided in the section entitled "Command Files, Groups, and Do Lists."

The commands in the <addfilename> are executed almost as if they were being typed in at the keyboard. *G7* knows whether the commands are coming from an add file or not, so sometimes it may behave slightly differently to help speed up the operation. As indicated by the syntax above, the arguments are optional. There may be 99 arguments, and they are passed as text strings. If a long text string with separators such as spaces or arithmetic operators needs to be passed, it should be enclosed in double quotes (""). The limit of the length of an argument is 90 characters.

Within the add file, the places where the arguments are to be substituted are indicated by "%1" up to "%99". Strings passed in quotes will be inserted without the quotes.

Add files also may have a group on the command line. A group is a certain way of coding a list of numbers that may represent industries, categories, or other numbers. For example, "1-85" the list of numbers from 1 to 85 inclusive. To use a group as an argument, we must enclose those sectors in parentheses like this:

```
(1-85)
```

An *add* command allows its last two arguments to be groups. For example one could write out 85 *add* commands as follows:

```
add agr out 1
add agr out 2
...
add agr out 85
```

or equivalently, write them out in only one *add* command:

```
add agr out (1-85)
```

Up to two group arguments are allowed. However, group arguments only can be the last ones on the argument list. If two group arguments are used, by default, the outer loop is controlled by the first group argument, and the inner loop by the second group argument. For example,

```
add invest.reg (32-34) (52-54)
```

is equivalent to:

```
add invest.reg 32 (52-54)
add invest.reg 33 (52-54)
add invest.reg 34 (52-54)
```

There is an variation to the double loop, parallel matching, with an ‘m’ option after the second group. For example:

```
add invest.reg (32-34) (52-55(53)) m
```

is equivalent to:

```
add invest.reg 32 52
add invest.reg 33 54
add invest.reg 34 55
```

That is, the first members of each group are matched to be the first pair of arguments, the second member of each group to be the second pair, and so on. The two groups must have equal number of members in parallel matching.

Related Topics: Command Files, Groups and Do Lists, *catch*, *do*, *fadd*, *function*, *hesitate*, *title*, *#*

(addp)rint <n|y>
(addt)ype <n|y>
(addp)ype

The *addprint* command is used to control the quantity of output going to the screen. If it is invoked with an ‘n’ or “no”, this command turns off the typing on the screen of the contents of the add files. This significantly speeds up the processing of large data files. If you want to turn printing back on, use “addp y” for “yes”.

Related Topics: *add*, *do*, *fadd*, *print*, *type*

autocomplete [<setting>]

The command box provides an auto-complete feature to speed the typing of repeated commands. Sometimes this feature proves troublesome. It can be turned off by clicking File | Autocomplete, or it can be turned off by typing the *autocomplete* command in the command box. Settings may be <0|1>, <yes|no>, <false|true>, or <off|on>. If no setting is given, then the current setting is displayed. The setting will be saved when *G7* is closed, and the same setting will be restored when *G7* next is run.

(autop)rint <y | n>

This command sets the global autoprint flag in *G7*, which is ‘n’ (no), or false, by default.

Example: When you set the flag to true, with:

```
autoprint y
```

then graphs will be printed as soon as they are drawn.

Related Topics: *graph*, *zip*

3.3 *G7* Commands: B

(ba)nk <bankname> [<location>]

This command assigns a *G7* workspace style bank (.BNK, .IND). The workspace style banks are the simplest type of bank in *G7*, and can be created by copying the files WS.BNK and WS.IND to two files with a different root name. If <location> is not given, it is assumed to be 'a', which is the first slot. Up to 25 data banks of the various types may be assigned, in positions 'a' through 'z' (except 'w').

In *G7*, the equivalent function can also be reached via the Bank, Assign menu item, and choosing Files of type "Workspace bank."

Examples:

```
bank mybank
ba h:\idlift\model\dyme d
```

Related Topics: Assigned Banks, *cbk*, *dfr*, *hbk*, *listbanks*, *vam*, *wsbank*, *wsinfo*

bj <q> <y> = <x1>, [<x2>,] . . . , [<xn>]

The "bj" command performs a Box-Jenkins estimation for autoregressive moving average models. Here q is the number of lags in the moving average error terms, e.g., with q = 2, u[t] = e[t] + b1e[t-1] + b2e[t-2]. This <q> must be no more than 4. The program will only check for the stability of the solution if q <= 2. No more than 10 independent variables are allowed at the current time.

At each iteration of the "Newtonian" non-linear regression algorithm, you will be allowed to intervene to try new values for the theta's, the coefficients for the moving average of error terms. If "save" is on, the first and the final equation will be in the ".SAV" file with the theta values appearing as coeffients of "theta". These must be removed before building with Build.

Example:

```
bj 2 vif$ = vif$[1], vif$[2]
```

Related Topics: ARIMA, *ac*

break

The "break" command terminates execution of an add file. If an add file calls a second add file, and the second contains the *break* command, then upon processing the *break* command *G7* will return to the first add file.

(bt)itle <ws | [a-z]> [<title>]

The *btitle* command will report the title of either the assigned bank or of the bank in location 'a' through 'z'. If you ask for the title of a location that has no bank assigned, *G7* will print an error message.

The *btitle* command also allows you to change the title of the workspace bank. After specifying the bank letter, simply provide the desired title *title*.

Related Topics: Assigned data banks, Workspace Bank

(bu)pdate <x> = <y>

The *bupdate* (bank update) command starts with a series <x> from the workspace bank, updates it with non-zero entries from the series <y> in the assigned bank, and replaces the series <x> in the workspace bank.

If <x> was not in the workspace previously, the *bupdate* will put it there with values from the assigned bank. Note that <y> may be an expression.

3.4 *G7* Commands: C

(cat)ch [-a|-w] <catchfile>

The *catch* command captures most screen output (except graphs) to the named <catchfile>. The default flag "-w" causes a new file to be created. The optional "-a" flag causes *G7* to open an existing file and append text to the end. This command is used to capture the set up of a regression and its results into a file. A regression display is caught as it stands when you proceed to the next command. To turn off the catching type "catch off". The *catch* command can also be used as a clever way to create new add files. Simply catch the results of a session, go in and edit out the extraneous output, and voila, a new add file!

Related Topics: *add*, *regress*, *zip*

(cb)k <bank_name> [<bank_location>]

The *cbk* command assigns a compressed bank (.CBK, .CIN). The <bank_location> is optional, and may be a letter between 'a' and 'z'. If no letter is given, the position will be 'a'. The compressed bank comes in two files. The first file, with extension .CBK, holds the data series in compressed format, and the second file, with extension .CIN, holds an index of names of series. When you give the <bank_name>, give only the "root name", that is, without the file extension.

Related Topics: Assigned Banks, *bank*, *dfr*, *hbk*, *listbanks*, *vam*

cc <C language statement>

This has no effect in *G7* or *Build,* but the C-language statement is passed through to the file HEART.CPP, which contains the model written in the C++ language and is ready for compilation. Such passed-through statements may be used to do special tasks for which *G7* and Build have no convenient mechanism – such as scaling one group of variables to sum to another variable. For many models, no *cc* statements are needed.

Related Topics: Model Building

cd <directory>

Change the current working directory to <path>. This is equivalent to the menu option File | Directory.

Normally when starting *G7*, you are asked to specify the location of the G.CFG you would like to use for identifying the default assigned bank and current working directory. Using the *cd* command, you can change the current working directory. After giving this command, *G7* will look in that directory when searching for .add files, .reg files, and databanks.

Related Topics: G.CFG, *pwd*

(ch)ow <n>

The *chow* command is a test of homogeneity. In the calling format above, <n> is the number of regressions involved in the test.

A Chow test is a special case of the Fisher F test used to test the homogeneity of a sample. Typically, one runs two or more regressions covering parts of a sample and compares the sum of squared errors with that of a single regression covering the whole sample. If we were going to break up the sample into two parts, the command would be

```
chow 3
```

Then we run first the two regressions over the separate halves of the sample, and finally the single regression over the whole sample. After this last regression, one will be greeted by the results of the Chow test.

There is a special case in which the second portion of the sample is too small to run the regression. In that case, give the command:

```
chow 2
```

and then run the regression over the reduced sample and then again over the whole sample. After the second regression, the Chow tests appear.

Related Topics: Regression Tests

clear
cls

The *clear* command can be used to clear the contents of the results window. This window can be useful for viewing the history of a *G7* session, but as it gets larger it may scroll more slowly. This command is equivalent to the File | Clear Results command from the menu.

close <bank letter> [<bank letter> <bank letter> ...]
close all

The close command is used to close a bank, multiple banks, or all banks.

Examples

```
close all    # close all open banks except the workspace bank
close a      # close bank 'a'
close c d e  # close banks 'c', 'd', and 'e'
```

cmtype <yes | no>
vcprint <yes | no>

If "yes", the simple correlation matrix and the standard deviation of each variable will be typed before the regression results. The default is "no."

coef [-c|r] <matrix> <vec> [year]

Convert the named flow matrix to a coefficient matrix by dividing each column of the matrix by the corresponding element of the named vector. If no year is named, the command works over the *fdates* range. If the '-r' option is specified, then each row of the matrix will be divided by the corresponding vector element; otherwise, the default '-c' setting is assumed or may be specified to indicate that column elements should be divided by vector elements.

Related Topics: *fdates*, *flow*, *getsum*, *move*, *ras*, *rowscale*, *scale*

(comc)oef <common_variables> <total_variables>

It happens fairly frequently that we have similar data sets collected at different times or in different places. We then may wish to estimate regressions which combine the data sets in various ways. The *comcoef* command in *G7* is designed for this purpose. It allows one to specify a number of "common coefficients" in the regressions that follow. The method may be explained best with an example. In the following example, "cps78" and "cps85" are data banks of from the Current Population Surveys of 1978 and 1985, respectively. The first has data on 550 individuals, the second on 534 individuals. We want to regress the logarithm of an individual's wage, lnwage, on the individuals education, ed. We want the coefficient of lnwage to be the same in both samples, but the interecepts may be different in the two years.

Here is what we do.

```
f one = 1
comcoef 1 4
hbk cps78
lim 1 550
r lnwage = ! ed, one
hbk cps85
lim 1 534
r lnwage = ! ed, one
do
```

In the *comcoef* command, the "1" tells the program to use the same coefficient for the first 1 variable(s) in all the following regressions. The "4" is the total number of variables, counting the dependent variable, in the combined or "pooled" regression. Once the *comcoef* command has been received, *G7* accumulates regressions until the *do* command is reached. Only a limited set of commands are a permitted between the *comcoef* command and the *do* command. They are the *bank*, *hbk*, *lim*, *f*, *add*, *pause*, *quit*, and *r* commands. The *quit* command only quits *comcoef*, not *G7*.

The *comcoef* command can be used under the *chow* command and under the *catch* command. It also puts the estimated coefficients into the *rcoef* series in the workspace bank, as usual. It does not enter a series of predicted values into the workspace bank, so *gr ** will not work following a *comcoef* regression, nor are the variable names shown on the display of the results, for in general the variables with a common coefficient may have different names while variables with different coefficients may have the same name, such as "one" in the above example.

commandcache <clear>
commandcache <print>
commandcache <replay>

These provide control over the command-line cache, where each command-box entry is recorded. Commands allow printing of the cached commands, clearing of the cache, and execution of the cached commands.

con <top> <c> = <linear function of ai's>

The *con* command imposes a "soft constraint" on a regression. The <top> (trade-off) parameter determines how "soft" is the constraint. The higher is this number, the harder the constraint. The constraint counts as *top*T* observations, where *T* is the number of regular observations in the regression. Soft constraints are also know as "Theil's mixed estimation," as "stochastic constraints," and are also a particular type of Bayesian regression.

Remember to put the value <c> of the constraint on the left hand side. Note that one parameter may appear in one or more constraints, but be careful not to impose constraints that are inconsistent!

We often use soft constraints when theory suggests that a parameter should be near a certain value, but ordinary regression gives a raw parameter that is not consistent with what theory suggests. Often, the theoretically sensible parameters also are those that lend stability to our models. The use of soft constraints also is implicit in the estimation of Almon lags, imposed using the *sma* command.

The *con* commands must be given before the regression to that they apply, and they will be cleared after that regression is estimated.

Examples:

```
con 100 1 = a3 + a4 + a5
con 200 0 = a7 - 3a6 + 3a7 - a8
```

Related Topics: Soft Constraints, *qpcon*, *sma*

(ct)rl <x> <concept> <group>

The *ctrl* command imposes the values of <x> series as a control total on a <group> of sectors of a given <concept> (such as exports). Results are stored in the workspace bank with names formed by <concept> and the given sectors.

For example, the command:

```
ctrl tot out 1-10 (4-7) 13 15
```

imposes the values of tot as a control total on the named sectors out, i.e. sectors 1 to 10, except for 4 through 7, and then 13 and 15.

Related Topics: Forming Variables, Groups, *group*, *scale*

3.5 *G7* Commands: D

data <name>
<date> <observation1> ... <observationN>

The *data* command introduces data into the workspace data bank. There are two forms the command can take.

Form 1:

```
data sales
  83.1   34.0 56.8 44.5 55.6
  84.1   39.3 41.2 43.9 47.0 ;
```

The first number on each line is the date of the first observation on that line. Input is terminated by a ";". The ";" may be omitted in an add file except on the last line of the file.

Form 2:

```
data sales 83.1
  34.0 56.8 44.5 55.6
  39.3 41.2 43.9 47.0 ;
```

The date of the first observation is given on the command line immediately after the series name. No dates appear on subsequent lines.

Form 1 is easiest if the data is being entered by hand. Form 2 is easier if the data is generated by a program.

Input data may contain floating point numbers in exponential form. Thus, the number 3 may be represented as 3.0, 3.0E+00, 0.300E+01, or 30E-01. Only E, not e, is recognized in this context. Any number of spaces between data observations is allowed (the data are free format).

A missing value may be represented by a single question mark (?). Therefore

```
data sales
   83.1   34.0 ? 44.5 55.6
```

indicate that the sales for 83.2 quarter is missing.

Related Topics: *frequency*, *matdata*, *update*, *vdata*

del <series name> [<series name2> [. . .]]

Delete the named series from the workspace data bank. The last series is moved to the place of the deleted series. The workspace bank file is not physically reduced in size by a deletion, but if another variable is added to it after the deletion, it will be put in the space formerly occupied by the deleted variable. One or more series may be specified for deletion.

Related Topics: *rename*

dfr <bank_name> <bank_location>

The *dfr* command is used to assign a Dirfor file as a *G7* assigned bank. Dirfor files are files containing forecast results and historical data from a run of the Inforum LIFT model, or from one of the family of *SlimForp* international models. (*SlimForp* is the predecessor to *Interdyme*.)

These Dirfor files have a format described by a DIRFOR.DAT, or DIRFOR.CRD file. After assigning a Dirfor file with this command, you will be asked to provide the name of the appropriate DIRFOR.DAT file. Note that the DIRFOR.DAT file must point to the appropriate DIRHIS.DHS history file, as well as a correct MACRONAM.BIN list of macrovariable names. The series names for *G7* will be formed by the concept name in the DIRFOR.DAT file, joined with the sector or category number.

Related Topics: Assigned Banks, *bank*, *cbk*, *hbk*, *listbanks*, *vam*

dfreq [0|1|2|4|6|12]

The *dfreq* command sets a default frequency for the left-hand side variable of an *f* command, when the right hand side does not have a frequency. The default is 0, which means no frequency. Otherwise, frequencies are determined by the frequencies of the series on the right hand side of the *f* command.

Related Topics: *frequency*

(diage)xtract <matrix_name> <vector_name)

This command takes the diagonal elements from the matrix and inserts them into the vector. The vector and the matrix must be of the same dimension, and the matrix must be square. One common use of this command is when you want to use the *index* command to move the A-matrix coefficients forward in time, excluding the diagonal. In this case, extract the diagonal to a scratch vector, do the coefficient indexing, and then insert the diagonal back again.

Example:

```
fdates 1998 2020
diagextract am z      # am is the A-matrix; z is a scratch vector
index 1998 mover am  # mover is a vector of coefficient change indexes
diaginsert am z
```

Related Topics: *diaginsert*, *index*

(diagi)nsert <matrix_name> <vector_name>

This command is just the reverse of *diagextract*. It inserts the elements of the vector into the diagonal of the matrix.

Related Topics: *diagextract*, *index*

dir [arguments]

The *dir* command prints the contents of the current working directory. The arguments are legal DOS arguments for the system *dir* command. Note, however, that any '#' characters are interpreted as the *G7* comment delimiter, and so any text after '#' is ignored.

Example:

```
dir t*.* # This command displays all files beginning with 't'.
```

Note that the /p option will not function properly, because at present there is no means for the *G7* user to interact with the operating system in this case (where the user is prompted to hit a key after every page). Because the *G7* window may be scrolled, the /p option is probably not needed. If necessary, the following *G7* commands will allow use of the option, but the results will not appear in the *G7* output window.

```
dos –i {dir t*.* /p; pause}
dos –I dir t*.* & pause
```

do{ <G7 commands with variables> }[<Arguments>]

A *do* command allows you to run a set of *G7* commands like an *add* command without actually creating an add file. Argument substitution is done just as in add files, where variables are %1, %2, etc. A *do* command can continue on several lines. However, the arguments should be placed on the same line as the closing brace mark '}'. As with the *add* command, no more than 99 arguments are allowed, and the last two arguments may be groups.

Example 1:

```
do{ gr out%1 65 97 }(1-87)  # graph output of sectors 1 through 87
```

Example 2:

The *do* loop can also make use of an if test in the loop. Useful examples of the need for this capability would be: 1) special operations needing to be performed for particular industries; or 2) report displayed for particular iterations. In this example, we iterate over values 1 to 10, and display the iteration numbers 1-3, 5, and 10.

```
do{
  if( %1 < 3 || %1 == 5 || %1 == 10){
    ic Iteration %1
    }
  }(1-10)
```

Related Topics: Command Files, Groups, *add*, *addtype*, *function*

dos [<dos command>]

dos [<options>] [<batch file arguments>] <{. . . }>

The *dos* command allows the passing of a DOS command to a temporary DOS shell. Batch files can also be run. When the DOS shell starts, you will see its output in the *G7* output window. If the *dos* command is given with no arguments, then a DOS command window will open, and will remain open until you type *exit*.

Example:

```
dos runmodel.bat
```

To pass a group of commands to DOS, use the *dos{}* command. The braces tell *G7* to create a temporary batch file containing the dos commands between the braces. The temporary file is destroyed automatically upon completion. Arguments may be passed to the batch file by listing them after the *dos* and before the opening brace. Here is an example that copies a workspace bank to another bank:

```
dos ws newws {
  rem Copying bank %1 to %2
  copy %1.ind %2.ind
  copy %1.bnk %2.bnk
  }
```

When this *dos* command is given, the three lines after the opening brace are written to a temporary batch file, and then that batch file is called with the arguments "ws" and "newws", which replace "%1" and "%2" in the batch file, respectively. The output of the batch file is written to the *G7* output window. If you would like to run DOS commands in interactive mode (e.g. when user input is required), use the "-i" option. For example:

```
dos –i idbuild master
```

opens a command window and executes the program IdBuild with master as a command line argument. If IdBuild requires user input, the user will see the prompt in the DOS window and can enter a response. The same option is available in batch mode (using the braces.)

(dv)am <bank_letter>

This command sets the default Vam file. *G7* can have many banks open at once, and some of these may be Vam files. Most commands that involve using vectors or matrices in a Vam file will operate on the default Vam file. Each bank is opened at a different bank letter position, and that is that bank letter that should be given with this command. A Vam file already must have been assigned as a bank before it can be made the default. However, if several Vam files are assigned, the default can be switched from one to another as often as is needed.

Example:

```
vamcr vam.cfg hist # Create hist.vam with VAM.CFG as configuration file
vam hist b         # Assign hist.vam in position "b"
dvam b             # Make hist.vam the default vam file
```

Related Topics: VAM.CFG file, *pmfile*, *vam*, *vamcr*, *vf*, *vup*

3.6 *G7* Commands: E

ed [<filename>]

The *ed* command is equivalent to opening an existing or new file using the File Open or File New commands from the editor menu. The editor opened is internal to *G7*, and supports multiple files being open simultaneously, cut and paste between files, and editor windows can be open in the background while you are working with the *G7* Console.

Related Topics: Editing Files

edfontname <name>

Set the typeface in the *G7* editor window.

Related Topics: *xl font*

edfontsize <size>

Set the size of the type in the *G7* editor window.

Related Topics: *xl font*

edfontcolor <color>

Set the font color in the *G7* editor window.

Related Topics: *xl font*

edcolor <color>

Set the background color of the *G7* editor window.

Related Topics: *xl font*

edfontbold <0|1>

Specify the bold setting in the *G7* editor window. Options may be specified as <0|1>, <yes|no>, <false|true>, or <off|on>.

edfontitalic <0|1>

Specify the italic setting in the *G7* editor window. Options may be specified as <0|1>, <yes|no>, <false|true>, or <off|on>.

edfontunderline <0|1>

Specify the underline setting in the *G7* editor window. Options may be specified as <0|1>, <yes|no>, <false|true>, or <off|on>.

edfontstrikeout <0|1>

Specify the strikeout setting in the *G7* editor window. Options may be specified as <0|1>, <yes|no>, <false|true>, or <off|on>.

eqpunch <filename> | "off"

The *eqpunch* command is for writing equation results to a file in a tabular format. We follow the precedent of giving those files an ".EQP" extension, but of course any file name will do. The use of the *eqpunch* command is analogous to use of the *ipch* command, and in fact both often are used in the same regression .REG file. The *eqpunch* command sets up a file for writing by the *titpch* and *tpch* commands. The *titpch* command writes out a header for the table, and the *tpch* command writes out regression results, one line per sector. See the documentation on the *tpch* command for a complete example.

Related Topics: *ipch*, *titpch*, *tpch*

3.7 *G7* Commands: F

f <variable> = <expression>

f <variable>{<date1> [- <date2>]} = <expression>

This command is one of the most commonly used as well as most powerful and useful commands in *G7*. It is equivalent to the LET or GENR commands in other packages, and its job is to form a variable on the left hand side from an expression on the right hand side of the equals sign. The syntax of the expressions used by *f* or similar to those in most programming languages. Any algebraically legal expression is allowed, including nested parentheses.

The left side variable must be a single variable or a variable with a subscript to indicate a specific time. For example, z and z{2010.1} are valid variables on the left side. However, z[1] (z lagged once) is invalid on the left side. A range of dates also may be provided to specify the periods to carry out the calculations.

For a series that exists in the workspace or in the assigned bank, the *f* command modifies the value within the period specified by *fdates* without affecting the existing values outside of that period, and puts the series in the workspace. Otherwise, the *f* command creates a series, determines the values using the expression on the right hand side of the *f* command for the period specified by *fdate*, and sets missing values, (-0.000001), for those observations outside the period. If "missing n" is in effect, the observations outside of the fdates period will be set to zero rather than to -0.000001.

In addition to the assignment operator (i.e. "="), the *f* command can add <expression> to the left-hand series using the "+=" operator, or it can subtract <expression> from the left-hand series with the "-=" operator, or multiply with "*=", or divide with "/=". These operators follow the syntax for the C++ programming language, though here they operate over a range of values.

For example, if we have in effect the following *fdates* command:

```
fdates 1978 2010
```

then for an *f* command

```
f x = <expression>
```

if x is an existing series either in the workspace or the assigned bank, x will be placed into the workspace with values from 1978 through 2010 being taken from the expression. Otherwise, x is created with values taken from the expression from 1978 through 2010, and set to be missing for the rest of the period.

Variable names must begin with a letter and may contain up to 32 letters, digits, or the '$' or '_'characters. Do not use a digit as the first character.

Example:

```
f x  =  y*(z[3] + v*@log(w) )/ s{2005.1}
```

Here, the variable x is calculated and placed in the workspace using the variables y, z, v, w, and s. These variables must already exist. Note in the example:

z[3]
: means z lagged three periods, i.e. z(t-3) in a common notat

s{2005.1}
: means the value of s in the first quarter of 2005,

@log(w)
: means the natural logarithm function of the variable w.

The full list of @ functions are shown in the list of functions.

Powers are obtained using the @sq and @pow functions, not ** or ^.

Right-hand-side expressions too long to fit on a single line may be continued by using a +, -, *, or / as the last character on a line.

The commands *f*, *fex*, and *fdup* have exactly the same effect in *G7* but very different effects in *Build*. See the model building topic for the differences.

Related Topics: Missing Values, Model Building, *fdup*, *fex*, *frequency*, *id*

fadd <CommandFile> <ArgumentFile> [<arg> [<arg>] . . .]

The named <CommandFile> is executed first with the arguments from the first line of named <ArgumentFile>, then the <CommandFile> is executed again with the arguments from the second line of the <ArgumentFile>, and so on until the <ArgumentFile> is exhausted. Additional arguments may be supplied with the *fadd* command, and these will be added to each line of arguments provided in the arguments file.

For example, the <CommandFile> could be

```
ti %2
gr %1
```

and the <ArgumentFile>

```
gnp$    "Gross National Product"
vfnre$  "Investment in producer durable equipment"
vfnrs$  "Investment in non-residential structures"
```

Then the result would be three properly-titled graphs.

Related Topics: Command Files, *add*, *addtype*, *punch*, *title*

fdates [date1 [date2]]
fdates off
fdates ±n1 ±n2

This command sets or resets the dates in effect for subsequent *f* commands and other operations. For a series that exists in the workspace or in the assigned bank, values may be modified within that period without affecting the existing values outside of the period.

Example:

```
fdates 1980.1 2010.4
```

In this case, subsequent *f* commands only alter the value of a series between the first quarter of 1980 through the last quarter of 2010.

The default *fdates* implicitly are defined in G.CFG: <fdate1> is the first period of the "Default base year of workspace file" as defined in G.CFG, and <fdate2> is the period implied by "Default maximum number of observations per series in workspace" in G.CFG.

"fdates off" resets the *fdates* to the default.

If valid dates have been specified, then they can be adjusted, moving either or both dates either forward or backward in time.

Example:

```
fdates +0 -2
```

If the command above follows the command in the first example, then the adjusted *fdates* will specify the period quarter one of 1980 through quarter 2 of 2010.

Related Topics: Dates in *G7*, G.CFG file, *coef*, *f*, *flow*, *gdates*, *index*, *punch5*

fdup <variable> = <expression>

In the context of the *G7* program, the *fdup* command is equivalent to the *f* command. However, when passed to the *Build* program through a .SAV file, it does nothing. It's main purpose is to do a calculation in one .REG file that already was done in another .REG file for use in the same a model, and so the equation does not need to be repeated. ("dup" stands for "duplicate").

Related Topics: Model Building, *f*, *fex*, *id*

fex <variable> = <expression>

The *fex* command is just like the *f* command in the context of *G7*. However, when passed to the *Build* program in a .SAV file, it only computes the left-left side variable and puts it into the workspace. It does not put the right-side variables of the expression into the workspace and does not put code for this calculation into the model. It often is used to define historical values of EXogenous variables, such as tax rates, and historical values of dependent variables of behavioral equations.

Related Topics: Model Building, *f*, *fdup*, *id*

findmode < m | a>

When multiple databanks have been assigned, this command sets the mode for searching through banks when looking for a time series. There are two modes: manual (m) and automatic (a). Manual is the default. In this mode, *G7* searches first the workspace bank, and then the bank in position 'a'. If you want to access or type a series in any of the other banks, you explicitly must specify the desired bank before giving the series name. For example, if *listbanks* shows that you have four banks assigned in positions 'a', 'c', 'e' and 'f', and you want to type the series gnp in bank f, then you must type:

```
ty f.gnp
```

In automatic mode, *G7* automatically will search through the workspace bank and then the assigned banks in alphabetical order until it finds a series with the specified name. If the name is a common one, such as "gnp", then be warned that you might not get the series from the bank you expected! In other words, if there was a monthly series in bank 'c' named "gnp" and a quarterly series in bank 'f', then typing

```
ty gnp
```

would get the monthly version.

Related Topics: Assigned Banks, *id*, *listbanks*, *type*

flow <matrix> <vec> [year]

Multiply each column of the matrix by the corresponding element of the named vector. If no year is named, the command works over the *fdates* range. This command is the opposite of *coef* command.

Related Topics: *coef*, *fdates*, *getsum*, *move*, *ras*, *rowscale*, *scale*

format {<width> <decs> <obsPerLine>} | "off"

This command is used to override the default settings of the *type* and *stype* commands for the width of the data, the number of decimal points, and the number of observations printed on each line. The <width> and <decs> arguments also apply to the *matty* or *matpr* command. If you give the "format off" command, this returns *G7* to its default style of formatting.

If the command was not issued earlier, *G7* will determine the format to use based on the absolute value of selected elements of the series. This sometimes will result in a messy .SAV file if data are printed at different widths and decimal points.

Related Topics: *matty*, *save*, *stype*, *type*, *print*

(freq)uency <series name> [frequency()<1|2|4|12>]

Some of the most Frequently Asked Questions about *G7* involve the topic of series frequencies.

Each time series in a G data bank has associated with it a one byte integer representing the frequency of the series. This frequency is set at the time of the creation of the series, but it may be changed with the *freq* command. In a *data* statement, *G7* determines the frequency of the series by looking at the dates. An annual date (1995, 1995.0) will indicate that the series is annual. Quarterly dates (1991.3) and monthly dates (1994.005) work accordingly.

Certain functions in *G7* will change the frequency of data, for example by performing conversions from quarterly to annual and back, or from monthly to quarterly. These functions, of course, create the new series in the target frequency.

A formula or expression calculated with the *f* command must involve series that all possess the same frequency (or else no frequency), and the resulting series will have that same frequency. However, note that the *f* command can be used to set a series to a constant, or to a linear time trend, using the @cum() function. In these cases, the frequency is 0, or undefined. As just mentioned, *G7* is quite happy to use series created in this way in expressions with other series. However, the *Compare* program will complain if it is asked to print a series with no frequency. To ensure that such series are assigned a frequency, use the *freq* command, or use the *dfreq* command in the beginning of your session or add file.

The *freq* command displays a series' frequency and allows you to change it. The series and its new frequency will be put into the workspace bank. Note that this is true even if the series originally was in an assigned bank.

Related Topics: Compare, Dates in *G7*, *G7* Functions, *data*, *dfreq*, *f*

function <function name> { <function body> }

function clear [<name>]

The *function* command enables users to define a function that will be available for the rest of the *G7* session. Arguments may be passed to the function, in which case they are referenced in the same way as in add files, with "%1", "%2", etc. The following example is a function that just states its arguments and how many arguments have been given. *function clear* will remove all defined functions, or if a function name is specified, then that function will be removed.

Example 1:

```
function TEST {
  if(%NARGS == 0){      ic Zero arguments were passed to the function TEST.        }
  else if(%NARGS == 1){ ic The argument %1 was passed to the function TEST.        }
  else{                 ic At least two arguments, %1 and %2, were passed to TEST. }
  }
```

The second example demonstrates several tests, first for the comparison of strings and then for the comparison of numbers. The arguments may be replaced by the standard *G7* variables (i.e. %1, %2, etc.) defined for "add" files and loop iterators.

Example 2:

```
if( Clopper != Almon ){
  ic String inequality test failed.
  ic Clopper is not the same as Almon.
  }
if( A <= C ){
  ic Character less than or equal to comparisons.
  }
if( -1 < -0.5 ){
  ic Integer evaluation with negatives.
  }
if( ( 2 < 3 && 3 <= 4 ) || A <= C ){
  ic Multiple evaluations with AND, OR, and parentheses.
  }
```

Recursive techniques may be employed.

Related Topics: Command Files, *add*, *do*

fvread <vector_name> <year> <skip>

The *fv* in this command name stands for "folded vector." The vector recorded in this format is "folded" across several lines. Each line can begin with several columns of comment, and <skip> argument is the number of columns to skip at the beginning of each line. After the comment follows the elements of the vector, divided across several lines. There must be present as many elements as are in the vector, and they must be in order.

Example:

```
fvread fd 1997 10
finaldem 1   23 36 83 92 32
finaldem 6    8 23 73 80 75
```

This example will put the vector (23, 36, 83, 92, 32, 8, 23, 73, 80, 75) into the fd vector of the default Vam file for the year 1997.

3.8 *G7* Commands: G

gdates <gdate1> <gdate2> [gdate3]
gdates automatic

Sets the dates used by subsequent *graph* (or *plot*) commands. With two dates provided, the series will be graphed from <gdate1> to <gdate2>. If a third date is given, the series will be graphed from <gdate1> to <gdate3>, with a vertical line drawn at <gdate2>.

"gdates a" selects "automatic" dates for *graph* and *type* commands. The automatic dates are the first and last date of the series actually present. The default setting in *look* is for automatic dates, unless specific dates previously have been specified. Automatic dates also adjust automatically to the frequency of the series. This feature is not to be trusted when more than one series is being placed on the same graph.

Related Topics: Dates in *G7*, *fdates*, *look*, *tdates*

getsum <matrix> <r|c> [<group>] <vector>

Get the sum of the rows or columns of a given matrix for the range of periods specified by the *fdates*. The first argument must be the matrix or vector for which we want to obtain the sums. The matrix is assumed to be stored in the default vam file unless a bank letter is provided. The second argument must be 'r' or 'c' (to obtain the sum by rows or by columns). If only a partial sum is desired, then the columns or rows are specified next. The last argument is the vector where the result is to be stored; this vector must be located in the default vam file.

Example:

```
getsum amf c intcol
```

This example puts the column sum of the A-matrix in flows ("amf") into the vector intcol.

The sum is calculated and stored for all the years in the Vam file so that we cannot have mixed information in the target vector.

Related Topics: *coef*, *flow*

glist <groupname>

This command lists the sectors in a group. The <groupname> specifies the group name. For named groups to be available, there must be a GROUPS.BIN file in the current directory, created by the *G7* or by the *Fixer* program. (See the *Fixer* documentation for more on how to create named groups.) Otherwise, there is one dynamic group, named ":", which is available after using the *group* command.

Example:

```
group 19-25 (20 22)
```

The name of this dynamic group is ":". The names and content of the static groups defined in *Fixer* or with the *G7* *group* command are preceded by a : in commands that use groups. After the above *group* command, the command:

```
glist :
```

would give:

```
19 21 23 24 25
```

Related Topics: *group*, *listgroups*

(gn)ame <prefix> <number>

In working with multisectoral models, it is common to have add files that draw graphs for many sectors. In this case, the *gname* can get a series of graphs started with a name and number, as follows::

> gname out 1

The name of the save file for the first graph drawn will then be OUT1.WMF, for the second OUT2.WMF, and so on. Note that the numbers increase with each graph drawn whether or not it is saved.

Related Topics: Drawing Graphs, *gsave*

(gp)rint

Print the displayed graph to the currently assigned default printer. This command is useful when you would like to print a large number of graphs without user intervention. Set up the default printer setup with Graph | Printer setup, then use *gprint* to print each graph in turn.

Related Topics: Drawing Graphs

(gr)aph <name1> [<name2>] [<name3>] ... [<name7>] <date1> <date2> [date3]
(gr)aph (<name1>) [(<name2>)] [(<name3>)] ... [(<name7>)] <date1> <date2> [date3]

Constructs a standard Line Graph. Graph the named series from <date1> to <date2> or <date3>, if present. If <date3> is present, a vertical line (to separate history from forecast) will appear at <date2>. If dates have not changed since the last *graph* command, they need not be repeated. After a regression, the actual and predicted values may be plotted by *gr **. The actual data will be marked by squares and the predicted by plusses, unless the marks have been changed by a previous *line* command. Algebraic expressions may be provided in place of some or all of the series names, so long as the expressions are enclosed in parentheses.

Example:

```
title RTB -- The Treasury Bill Rate
gr rtb 70.1 85.1
```

Related Topics: Dates in *G7*, Drawing Graphs in *G7*, Printer, *autoprint*, *hrange*, *legend*, *lgraph*, *line*, *look*, *mgraph*, *sgraph*, *subtitle*, *title*, *vrange*, *vaxlab*, *vaxtitle*

(gridty)pe [file <filename>] [<date1> <date2>] <ser1> [<dp1> ...<seriesN> <dpN>] ;

This command displays a scrollable spreadsheet, or grid, in a new window. Each column holds a variable, and each row shows a separate date. Column and row headings are also included. The grid display is convenient for scrolling around to look at the data. One also can copy data from the grid display to the Windows clipboard. If the optional "file <filename>" part of the command is given, the results are written to an output file, exactly as with *matty*. The dates are optional. If none are given, then default values are determined by the current *tdates*. After each series name there is an optional "decimal places" number (<dp1> ... <dpN>). Each series can be specified to have a different number of decimal places. If not specified, the default value of 3 is employed, unless the *decs* or *format* command has been given to specify a different default. If a value for <dp> is given for one series, it will be in effect for all of the other series in the list.

Example:

```
# GRIDTY.TST - Test the operation of the "gridty" command.
hbk quip
gridty gnp c v;
```

Related Topics: *matty*, *show*

(gro)up <group definition>

Define the content of the dynamic group. A group definition works as follows. Any list of numbers separated by blanks or commas may be included. Two numbers joined by a dash ('-') indicate a consecutive range of numbers to be included. A group definition within a group definition, enclosed by parenthesis indicates a group of sectors to be excluded from the overall group.

For example:

```
group 1-10 (4, 7-9)
```

Indicates a group consisting of the numbers (1,2,3,5,6,10). The name of a group created by the *group* command in *G7* is :. This name is used in the various commands that may use groups. These commands may also use named groups, if they have been created in a GROUPS.BIN file by the Fixer program.

Related Topics: Groups, *ctrl*, *glist*, *index*, *lint*, *listgroups*

group <group name>
<group definition>

This version of the *group* command adds a named group to the GROUPS.BIN file. This group then can be used in other commands requiring group expressions by prefixing the group name with a colon ':'. The group definition include a sequence of integers, a range of integers, a set of letters to specify spreadsheet columns, and other named groups. To exclude a set of values from the group definition, simply include the set to be excluded in parentheses.

For example:

```
group Manufacturing   # All manufacturing sectors
1-58
f empmfg = @csum(emp, :Manufacturing)
group NonchemicalMfg  # All manufacturing sectors except chemicals
:Manufacturing (20-27)
```

(gs)ave <filename>

Save the current graph to a Windows Metafile. Only give the rootname of the file in <filename>, the extension .WMF will be automatically appended. The file will be saved to the current directory unless you supply a full path name.

Related Topics: Drawing Graphs, *gname*, *listgroups*

(gtf)ile <stubfile | "off">

The *gtfile* command scans the specified stub file and stores the starting position of each line in the stub file. This command is used only in preparation for giving a sequence of *gtitle* commands. The titles from the stub file that just has been scanned now are available to be used as titles, as specified by *gtitle*. To close the *gtfile* command, give "gtfile off".

> Related Topics: *gtitle*

(gti)tle <n>

This command requires that a *gtfile* command has been given with a valid stub file as an argument. The *gtitle* command then will use the n-th line of the stub file to take a title in the following *graph* command. (The title will be the text after the semicolon (';') in the stub file.) These commands, *gtfile* and *gtitle*, particularly are handy in combination with the *do* command or with an add file with group arguments. Since in these cases only sector numbers are being passed, the *gtitle* command allows you to specify a sector number and get back the title for that sector.

Related Topics: *gtfile*

3.9 *G7* Commands: H

help

The *help* command can be given from the *G7* command box, in which case it will bring up the *G7* Help files. Contents include the *G7* User Guide, Reference Manual, and the Tutorial, plus manuals for *Compare*, *Fixer* and *MacFixer*, *Banker*, and *IdBuild*.

Related Topics: Help Menu

hbk <bank_name> <bank_location>

The *hbk* command assigns a hashed bank (.HBK, .HIN). The optional <bank_location> parameter may be any letter between 'a' and 'z' except 'w'. If no letter is given, the position will be 'a'. The hashed bank comes in two files. The first file, with an extension .HBK, holds the data series in compressed format, and the second file, with extension .HIN, holds a hash table that allows quick access to a large number of series. When you give the <bank_name> to the *hbk* command, give only the "rootname" of the file, that is, without the file extension.

Note that when you do a *listnames* command on a hashed bank, the series names will not be in the order in which the bank was created by the *Banker* program. This is because the hashing algorithm puts the series names into various "bins", based on the hash value of the series name. When reading the series names back again, *G7* just goes through each bin in sequence, which is not the sequence in which the bank was created. See the *Banker* manual for more on how to create compressed and hashed banks.

Related Topics: Assigned Banks, Using Banker, *bank*, *cbk*, *dfr*, *listbanks*, *vam*

(hes)itate < y | n >

This command is almost the opposite of the *zip* command. If you give “hes y”, then *G7* will pause before displaying each regression so that you can read the commands that prepared the regression. To turn off hesitation, use “hes n”. Remember that if you wish to have the add file stop so that you can see what has just gone by on the screen, use the *pause* command in your add file.

Related Topics: Pause, *add*, *zip*

hl <rho1> <rho2> <incr> <y> = <x1>, [<x2>,] [<x3>,] . . . [<xn>]

The *hl* command applies the Hildreth-Lu procedure for correcting for serial correlation of errors. It is a special case of Generalized Least Squares (GLS). In the Hildreth-Lu procedure, a guess of the autocorrelation coefficient of the errors, rho, is chosen, multiplied by the equation lagged once, and subtracted from the unlagged equation. The resulting equation then is estimated by ordinary regression. Another value of the guess of rho then is chosen and the process is repeated. In this command, <rho1> is the starting guess of rho, <incr> is the amount by which it is changed on each iteration, and <rho2> is an upper limit on the guess. The <y> and <x1>, . . . , <xn> are as in the *r* command. At the end of the process, you will get a table with this heading:

```
RHO-HL    SEE 1 AHEAD  RHO-EST    SEE LONG-RUN
```

The RHO-HL shows the assumed rho, the SEE 1 AHEAD shows the standard error of estimate (SEE) of the estimated equation, RHO-EST shows the rho of the estimated equation, and SEE LONG-RUN shows the standard error of using the fitted equation on the original, undifferenced data, without a knowledge of the true lagged value of the dependent variable, as must be done in forecasts of more than a few periods ahead. If the *save* command is on, all of the estimated equations will placed in the “.SAV” file as undifferenced equations suitable for going into a model. You must choose which one you want.

For graphing the *hl* output, one should run the equation one last time with the chosen value of rho as both the starting and ending rho. Then also run the equation with just the *r* command (no autocorrelation)0. Then construct two graphs:

```
gr depvar predp1 hlshort
gr depvar predic hllong
```

The first will compare the actual with two one-period ahead forecasts, the one from ordinary least squares (+’s) and one from Hildreth-Lu (x’s). The second compares the actual with two forecasts not relying on the lagged value of the dependent variable. Again, the un-corrected least squares fit is shown by +’s and the fit using the Hildreth-Lu correction is shown by x’s.

The pauses at the end of each fit in the HL procedure can be eliminated by the *zip* command. If you use *zip*, however, you must remember to do “zip off” before using the *graph* command in order to see graphs.

(hr)ange <bottom> [top]

hr <bottom> [line1] [line2] . . . line[8] <top>

hr off

The syntax of the *hrange* command is identical to that of the *vrange* command. However, it is used only when doing *sgraph*, where each axis corresponds to the values of one variable. In this case, it defines the range or locations of tick marks for the horizontal axis.

For example,

```
ti The Phillips Curve
subti from 70.1 to 96.1
gdates 70.1 96.1
vaxl out
vaxti Inflation
legend s
vr 0 2 4 6 8 10 12
hr 4 6 8 10 12
f infl = 400*(dgdp - dgdp[1])/dgdp[1]
sgr infl unempc
```

In this example, inflation (infl) is measured on the vertical axis, which runs from 0 to 12, and unemployment (unempc) is measured on the horizontal axis, which runs from 4 to 12. Axis labels and tick marks on both axes are every two percentage points.

Related Topics: Drawing Graphs, *graph*, *sgraph*, *vrange*

3.10 *G7* Commands: I

ic <comment>

The *ic* (insert comment) command is used to put comments (lines beginning with '#') into save files. This can be useful when you are writing a *G7* data file and would like to put titles or comments next to each data series.

Related Topics: Comments, *save*

id <identity expression>

The *id* command has no effect in *G7* but when passed to *Build* puts all variables (without computing the left-side one) into the workspace and puts an identity into the model. The syntax for the expression is the same as the *f* command.

Example:

```
ti 4. Additions and Alterations
fex cst4h = cst4$/hhld
r cst4h = ddi87h, rcmor, ratdif[1], hhead, singsh
id cst4$ = cst4h*hhld
gr *
```

This example is one of the construction equations from the IdLift model. The *id* statement is used here so that there will be a statement in the model to create cst4$ after calculating the regression. However, we don't want the regression runstream to recalculate the value of cst4$ and put it into the workspace bank since the proper values of this series already were put into the bank by the previous *fex* statement.

Related Topics: Model Building, *f*, *fdup*, *fex*

if([comparisons]){ [. . .] }
[else if([comparisons]){ [. . .] }]
[else{ [. . .] }]

The *if* command evaluates a series of logical expressions. If the arguments are true, then the following block of code, contained in a set of brackets, is executed. If the argument is false, then *else* *if* statements are processed in the same way. If all *if* and *else if* arguments are false, then the code following the final *else* command is processed.

Three types of comparisons are allowed. First, a number may be compared to a second number. Second, a single number may be given and implicitly compared to zero. Finally, a string may be compared to a second string.

Valid comparisons between two strings, with the comparison based on the lexicographical ordering, or between two numbers are

<
: is LESS THAN

<=
: is LESS THAN OR EQUAL TO

==
: is EQUALS

>=
: is GREATER THAN OR EQUAL TO

>
: is GREATER THAN

!=
: is NOT EQUALS

!
: is the logical negation operator. If given before a number, then the result is FALSE if the number is nonzero and the result is TRUE if the number is zero. If given before a set of parentheses, then the result is FALSE if the expression within the parentheses is TRUE, and the result is TRUE if the expression within the parentheses is FALSE.

Additional string comparison tools include:

strcmpi(<string1>,<string2>)
: Compares string1 and string2, without case sensitivity. Value is TRUE if strings match.

strncmp(<string1>,<string2>,<N>)
: Compares the first N characters of string1 and string2, with case sensitivity. Value is TRUE if the first N characters of the strings match.

strncmpi(<string1>,<string2>,<N>)
: Compares the first N characters of string1 and string2, without case sensitivity. Value is TRUE if the first N characters of strings match.

Note: These routines have been replaced by the %strcmpi(), %strncmp(), and %strncmpi() routines.

Multiple comparisons may be performed, with the following operators joining them

||
: Logical OR, which returns TRUE if either the left hand side or the right hand side is TRUE.

&&
Logical AND, which returns TRUE if both the left hand side and the right hand sides are TRUE, and returns FALSE otherwise.

The following list of keywords may be used:

NARGS
The number of arguments passed to a function or to an add file.

exists(<series name>)
The presence of a series in a data bank can be tested. "exists(a.x)" returns true if variable x is found in bank a.

eval(series, date)
A number may be retrieved from a data bank for comparison. The value of series in period date is retrieved, either from the workspace or from the specified bank. The value may be compared to a scalar or to a second eval result.

Note: These routines have been replaced by the %NARGS, *%exists()*, and %getval() routines.

Comparisons may be evaluated as a group by surrounding them with parentheses. Blocks of code following the arguments must be surrounded by brackets {}.

Examples:

```
if( 1 - 2 < -0.5 )  { ic Subtraction }
if( 2 < 3 && 3 <= 4){ ic Multiple evaluations }
if( 2 < 3 || 3 <= 4){ ic Multiple evaluations with OR }
if( 1 == 1 )        { ic Equality }
else if( !0 )       { ic Else if and negation }
else                { ic Else }
if( D != N ){ ic String inequality comparisons }
if( D <= N ){ ic String less than or equal to }
if( mod(7,2) == 1 ) { ic Modulo arithmetic }
do{
  if( %1 == 3){
    ic Continuing on Iteration %1
    continue
    }          if( %1 == 5 ){
    ic Breaking on Iteration 5
    break
    }
       if( %1 > 5 ){ return ERR }
       }( 1-10 )
```

(in)dex <base date> <guide> <vector name>

(in)dex <base data> <guide> <groupname>

A quick way to fill in values of a vector so that they all grow at the same rate is to use of the *index* command. It is used in conjunction with a guide series, a previously established single-variable time series, whose growth rates will be applied to the elements of the vector. It acts over the range of the currently specified *fdates.*

For example, if "years" is the name of a series of the numbers of years – 1980, 1981, etc. – then to make all elements of the vector “pce” grow by 2.0 percent per year from 1999 to 2010, the following commands can be used:

```
f  g02 = @exp(0.02*(years - 1980))
index 1999 g02 pce
```

The use of groups is allowed in the names of series to be affected. To use groups, we must first use the *load* command to load the vector into memory. For example, to move only the sectors 1,7, and 9 of pce, we could do:

```
group 1 7 9
load pce
index 1999 g02 :
```

Recall that *:* is the name of the dynamic group, if it exists. To move just the sectors in the static or named group ConsumerDurables, the command would be:

```
load pce
index 1999 g02 :ConsumerDurables
```

To index only a specific vector element like “exp2”, the command would be

```
index 1999 g02 exp2
```

Provision of a vector or matrix name without a sector number means to move the whole vector or matrix by the index. This device can be used to project a whole input-output matrix, say, am, to be constant, as in this example:

```
fdates 1998 2020
f one = 1
index 1998 one am
```

A matrix name followed without a space by a sector number means to move that row of the matrix by the guide. Thus, “am2” means move the second row of the am matrix by the guide.

If <guide> is the name of a vector and <name> is the name of a matrix, then the rows of the matrix are indexed by the corresponding elements of the vector. If an element of the vector is zero in the base year, the corresponding rows of the matrix are unchanged. This feature works for both standard and packed matrices. It can be used to apply across-the-row coefficient change to an input-output matrix. For example, if mover is a vector, we can make each row of the am matrix follow the index of the corresponding element of mover with:

```
fdates 1998 2020
index 1998 movers am
```

In applying this sort of across-the-row coefficient change, one usually does not want it to apply to the diagonal elements, for they have little to do with the technological changes affecting other coefficients. For this, we would use the *diagextract* and *diaginsert* commands to save the diagonal first, then do the indexing, and then put the diagonal back into the matrix, as in the following example:

```
fdates 1998 2020
diagextract am diags
index 1998 movers am
diaginsert am diags
```

Related Topics: *diagextract*, *diaginsert*, *fdates*, *group*, *rowscale*

ipch [<which>] <label> <sector> [<type>] [<psn1>] ... [<psnN>] [extra <var1> .. <varN>]

The *ipch* (*Interdyme* punch) command is used to create equation files for use in building *Interdyme* models, where:

<which>
is the suffixed number

<label>
is the name of the vector for which the equation is estimated

<sector>
is the number of the sector to which the equation applies

<type>
is a character ('a', 'b', etc.) that signals the type of form of the equation. The type should not be a numeral.

<psn1>
(optional) the column number in the matrix where the first regression coefficient belongs.

<psnN>
The column number in the matrix where the n'th regression coefficient belongs.

At the end of the command, type the word "extra" (without the quotes), and then type several variable names that hold the values of extra parameters you would like to pass to the file. For example, for an investment equation, which calculates capital stocks, and replacement investment, a physical service life is needed to calculated the spill rate. The following code would write the physical life at the end of the parameter list:

```
f plf1 = 4.9
r eqi1 = out1, out1[1], out1[2]
ipch eqi 1 a extra plf1
```

The *ipch* is not useful unless an equation file has already been opened with the *punch* command. See the *Interdyme* manual, chapter 4, for more information on the *punch* and *ipch* commands, and for details on detached-coefficient equations in *Interdyme*.

Related Topics: Interdyme Models, *eqpunch*, *pch*, *punch*, *zip*

intvar <prefix> <date1> <date2> [<date3> <date4> ... <dateN>]

The *intvar* command makes up linear interpolation functions between the given dates. Each linear interpolation function begins at 0 and remains at 0 until its particular time interval comes; then it rises by 1 each period until the end of its interval. After that, it remains constant at whatever value it has reached. For example, if the <prefix> is "tax", and we have 6 dates, 6 linear interpolation functions will be created, with names "tax1", "tax2", . . . , "tax6". Then, if we regress a policy variable, such as the federal tax rate, on these functions, the predicted values from the regression take the shape of a set of spliced line segments, approximating the curve of the left hand side variable. For example, to approximate the federal personal tax rate in the *Quest* model, use:

```
r pitfBR = tax1, tax2, tax3, tax4, tax5, tax6
```

Related Topics: Optimization

3.11 *G7* Commands: L

(le)gend <a> [<b>]

The *(le)gend* command controls printing a legend at the bottom of a graph.

<a> = y
for yes, prints the legend (default)

<a> = n
for no, do not print the legend

<a> = s
leave space for legend but do not print. It allows the user to annotate the legend.

<b> = y
for yes, mark the dates (default)

<b> = n
for no, do not mark dates. It is used to make bar graphs of data occurring at arbitrary intervals.

Related Topics: Drawing graphs in *G7* *graph*

(lgr)aph <name1> [<name2>] [<name3>] . . . [<name7>] <date1> <date2> [date3]
(lgr)aph (<name1>) [(<name2>)] [(<name3>)] . . . [(<name7>)] <date1> <date2> [date3]

Construct a Semi-Logarithmic Graph. This is exactly like the *graph* command except that the series are expected to be in logarithms. The vertical scale will be marked in the natural units, not the units of the logarithms. If vertical range control is in effect (see below), the vertical ranges will be presumed to be in natural units. Algebraic expressions may be provided in place of some or all of the series names, so long as the expressions are enclosed in parentheses. This type of graph also is called a semi-log graph.

Related Topics: *graph*

(lim)its <date1> <date2> [<date3>]
(lim)its <±n1> <±n2> [<±n3>]

The *limits* command sets limits for regressions and for @sum() commands. Estimation begins at <date1> and ends at <date2>. Simulations or forecasts will be made to <date3>. Remember that quarterly dates require .1, .2, .3 or .4. Monthly dates require .001 . . . .012. Annual dates are integer values.

Example:

```
lim 1975.1 2010.4 2015.3
```

If valid dates have been specified, then they can be adjusted, moving either or both dates either forward or backward in time.

Example:

```
lim +0 -2 +1
```

If the command above follows the command in the first example, then the adjusted limits specify the period quarter one of 1975 through quarter 2 of 2010, and a forecast period through quarter four of 2015.

Related Topics: Dates in *G7*, Regression, *mode*, *recursive*

line <linenumber> <color num | name> <thickness> <mark> <style> <barfill> <left> <right>

line <linenumber>

The *line* command is used to control the appearance of lines and bars in graphs. The command may be typed interactively, or used in add files to change the appearance of graphs in a "show" file. In *G7*, the *line* command can be written from the Graph | Settings dialog box by clicking the Save To File button. This brings up another dialog which asks you for a name of the text file to which you would like to save the graph settings. Suppose you call this file GRAPH.SET. The contents of GRAPH.SET might look something like:

GRAPH.SET							
line 1	255	2	+	0	0	0.000	1.000
line 2	16711680	2	s	1	0	0.000	1.000
line 3	32768	2	x	2	0	0.000	1.000
line 4	16776960	2	d	3	0	0.000	1.000
line 5	8388736	2	^	4	0	0.000	1.000
line 6	8388608	2	v	0	0	0.000	1.000
line 7	5315660	2	n	0	0	0.000	1.000

Alternatively, providing the *line* command followed by a single "linenumber" argument causes the settings for the corresponding line to be printed to screen. Settings for up to 7 series can be specified by the *line* command. The first argument, <linenumber>, corresponds to the series given in that position on the *gr* command line. The first series after *gr* is line 1. The second argument, <color_number>, can be a decimal number representing the components of red, green and blue of the color, or it can be a color name. The third argument, <thickness>, may be set to anything you like, but something in the range of 2 to 4 generally is pleasing. The fourth argument specifies the type of marker on the line, if any. Here is a short table of the possible characters for this argument, and what they mean:

Symbol	Meaning
+	Plus signs
x	X marker
s	Square
d	Diamond
>	Arrow in direction of line
v	Downware pointing triangles
b	Bars
f	Bars with no surrounding rectangle

The next argument is the <style> of the line, possible numeric values for this argument are:

Value	Meaning
0	Solid
1	Dashed
2	Dotted
3	Dash dot
4	Dash dot dot
5	Clear (blank line)

The next argument, <barfill> indicates the type of fill pattern that a bar will take, if a bar graph has been chosen. Possible numeric values for this are:

Value	Meaning
0	Solid fill
1	Clear (empty)
2	Horizontal lines
3	Vertical lines
4	Forward diagonal fill (/////)
5	Backward diagonal fill (\\\\\)
6	Cross hatch
7	Diagonal cross hatch

The <left> and <right> parameters, must be between 0 and 1. They show where, within the space allocated for the bar, the left and right edges of the bar go. For example, with left = 0.1 and right = 0.9, the bar will be in the center of the allowed space, and the bars will be four times as wide (0.8 units) as the space between them (0.2 units). To eliminate the line connecting points marked by bars, set its width to zero. Parallel bars are drawn by assigning non-overlapping intervals to different series. Stacked bars or high-low graphs are produced with overlapping intervals. The table below shows some of the legal color names. Color names may be capitalized or not. Where names have "light" or "dark" these can be abbreviated as "lt" or "dk". For example, "Lightgray" is the same as "ltgray".

Basic	Light/Dark	Others	
black	lightgray	olive	pumpkin
white	darkgray	lavender	seagreen
red	darkblue	forest	chocolate
green	darkred	aqua	slategray
blue	darkgreen	chartreuse	skyblue
yellow	lightblue	maroon	tomato
orange	lightred	fuchsia	teal
purple	lightgreen	azure	coral
brown	darkbrown	midnight gold	

Related Topics: Drawing graphs, Graph settings, *vr*

lint <name>

lint <group>

Linear interpolation of missing values in a vector is done by the *lint* command. The *lint* command replaces missing values in the time series by linearly interpolated values. Zeroes before the first or after the last observation are not replaced. This command applies only to series in the default Vam file. Groups may be used in the <names> field to refer to sectors of the currently loaded vector.

Example:

```
lint pce1
```

This example loads the pce vector and then fills in the missing values in sector 1 by linear interpolation. We could similarly fill in the values for all the sectors in the present dynamic group by

```
load pce
lint :
```

or all the values for the static group Services by:

```
load pce
lint :Services
```

Entire matrices may be interpolated with one command. For example,

```
lint am
```

will interpolate the entire am matrix. This interpolation also works for packed matrices. The *lint* command works on the entire range of the Vam file without regard to *fdates*.

Related Topics: *group*

linv [bank letter.]<matrix A> [period]

The *linv* command calculates the Leontief inverse and places it in the source matrix. It works over the current *fdates*, or for the [period] specified on the command line.

For example,

```
linv A 1995
```

replaces A with its Leontief inverse for 1995.

Related Topics: Matrix operations, *madd*, *mcopy*, *minv*, *mmult*, *mtrans*, *vc*

listbanks (lb)

This function provides information about the banks that are currently assigned in *G7*. Up to 25 banks of various types may be assigned, to positions 'a' through 'z' except 'w'. The *lb* command reviews which banks are assigned to which position and what type of bank they are.

Related Topics: Assigned banks, *bank*, *cbk*, *dfr*, *findmode*, *hbk*, *vam*

(listg)roups

List the names of all groups currently in the GROUPS.BIN file.

Related Topics: *glist*, *group*

(lis)tnames [-srgv] [-l <root>] <bank_location> [wildcard]

Displays the series names in the workspace (w) or the assigned bank (a). For example,

```
lis a
```

This command also has a "wildcard" option, which allows you to list variable names matching a certain pattern. For example,

```
lis a out*
```

will list all series in the assigned bank that start with out (such as out1 through out85).

Option ‘s’ sorts the series in alphabetical order, and ‘r’ reverses the order. If a Vam file is associated with the G bank, then option ‘g’ prints only macro series and option ‘v’ prints only Vam bank series. If the ‘l’ option is specified and a root name is provided, then the a set of strings will be defined. The string names will be constructed as the root name followed an integer, and the string definition will be the name of the series. One string will be created for each series in the bank.

Related Topics: *listnamescol*

listnamescol [-srgv] [-l <root>] <bank_location>
lnc [-srgv] [-l <root>] <bank_location>

The *listnamescol* or *lnc* command works just like the *lis* command, but all series names in the workspace (w) or assigned bank (a) are output in one column. You may find it convenient to capture output to a file, and then use this command to get a list of all the series in the databank in that file. Then, using an editor that supports macros, you can change all lines to create an addfile that does the same thing with each series in the databank. Use of wildcard is optional, and like in DOS, ‘*’ will match any number of characters. For any single character, however, you need to use ‘^’, instead of ‘?’ as in DOS. Option ‘s’ sorts the series in alphabetical order, and ‘r’ reverses the order. If a Vam file is associated with the G bank, then option ‘g’ prints only macro series and option ‘v’ prints only Vam bank series. If the ‘l’ option is specified and a root name is provided, then the a set of strings will be defined. The string names will be constructed as the root name followed an integer, and the string definition will be the name of the series. One string will be created for each series in the bank.

Related Topics: *bupdate*, *listnames*, *save*

listvecs
lv [-sr] [<bank_location> [<wildcard>]]
lvc [-sr] [<bank_location> [<wildcard>]]

The *listvecs* or *lv* command works just like the *lis* command, but all vector and matrix names in the default vam file or specified bank are printed. The *lnc* is the same but prints names in one column. Option ‘s’ sorts the series in alphabetical order, and ‘r’ reverses the order. By default, the command will operate on the default vam bank, but other bank locations may be specified. Use of wildcard is optional, and like in DOS, ‘*’ will match any number of characters. For any single character, however, you need to use ‘^’, instead of ‘?’ as in DOS.

load <vector_name>

When *G7* is working on a vector in a Vam file, it pulls the whole time series for the vector into a sort of vector workspace. Usually, this is done implicitly simply by referring to the vector or one of its elements. The *load* command enables the user to do this loading explicitly. It is used principally in connection with the *index* command described below.

Example:

```
load pce
```

Related Topics: *store*

look <bank_location>

This command from the *G7* console is equivalent to the Bank | Look menu command. As usual, <bank_location> is a letter between ‘a’ and ‘z’ to which you have assigned the bank you would like to look at. For *look* to work, there must be a stub file created with the same root name as the bank. For example, the stub file for the NIPAA.HBK, databank, which is a hashed bank, is NIPAA.STB.

The *look* command brings up the stub file in a scrolling list box, from which you can select lines of the file to print out and graph the various series in the bank. If you double-click on the series name, the series will be printed out using the current graph dates (*gdates*), and the title of the graph will be the line of the stub file. The series also will be printed to the *G7* console.

Related Topics: Dates in *G7*, Stub Files, *gdates*, *graph*, *tdates*

ls [-<flag>] <x> <y> <date1> [<date2>] [direction]

This *linkseries* command takes series <x> and <y> from the workspace or the assigned bank, calculates their ratio at the specified linking date <date1>. This ratio then is used to move <x> with non-zero entries from series <y>. The linking direction can be 'f' for forward or 'b' for backward. The default is 'f'. If the variable <x> is found in the workspace, then the result is stored with name <x> in the workspace. If a bank letter is provided for <x>, and if this bank is the default Vam file, then the modified value of <x> is stored in the Vam file. Otherwise, the result is stored to the workspace. If the optional date <date2> is provided, then linking will extend from the base period <date1> to the terminal period <date2>. <date2> is the last period (if the direction is 'f') or first period (if the direction is 'b') that will be modified. If not given, then the values will be modified to the end of the series, with no regard for the *fdates* settings that ordinarily will limit the extent of operations.

If the signs of the values in x and y are not the same, then the result will move in the opposite direction of the guide series; this usually is not desirable and a warning will be given. Three alternatives to the basic routine may be implemented by using a flag.

-f

the calculation using standard algorithm without reporting problems.

-i

if x and y are of opposite sign in the base year, then ensure positive correlation between x and y such that

x = a + b*y b = x{date} / y{date} * sign(x{date}*y{date}) a = x{date} - b * y{date}

-a

if x and y are of opposite sign in the base year, and the problem is ill-conditioned such that y lies close to zero in the link year, then ensure positive correlation between x and y such that

x = a + b*y b = x{date}/ AVG(y) * sign(x{date}*AVG(y)) a = x{date} - b * y{date}

where AVG(y) is calculated from fdate1 to <date>, if direction is 'b', or from <date> to fdate2, if direction is 'f'. This may be useful when y{date} is close to zero so that scaling parameter b is large, thus scaling x excessively. If AVG(y) is close to zero, then b = x{date} * sign(x{date}*AVG(y)). It is the user's responsibility to ensure that the results are useful.

Related Topics: @bmk function

3.12 *G7* Commands: M

madd <matrix A> = [bank letter.]<matrix B> + [bank letter.]<matrix C> [period]

The *madd* command performs addition or subtraction of two matrices and stores the results in a third. If the optional date <period> is absent, then the operation is done over the *fdates* range. If no bank letter is specified, the right-hand side matrices are assumed to be in the default Vam file.

Example:

```
fdates 1972 2020
madd A = B + C
```

This example forms A as the sum of B and C, over the period 1972 to 2020, with all matrices in the default Vam file.

Example:

```
fdates 1972 2020
madd A = e.B – e.C
```

This example forms A in the default Vam bank as the difference of B and C from bank 'e', over the period 1972 to 2020.

Related Topics: @bmk function, *linv*, *mcopy*, *minv*, *mmult*, *mtrans*

mainfontname <name>

Set the typeface in the main *G7* window.

Related Topics: *xl font*

mainfontsize <size>

Set the size of the type in the main *G7* window.

Related Topics: *xl font*

mainfontcolor <color>

Set the font color in the main *G7* window.

Related Topics: *xl font*

maincolor <color>

Set the background color of the main *G7* window.

Related Topics: *xl font*

mainautofontsize <0|1>

Specify whether to adjust the font size automatically in the main *G7* window. Options may be specified as <0|1>, <yes|no>, <false|true>, or <off|on>.

mainfontbold <0|1>

Specify the bold setting in the main *G7* window. Options may be specified as <0|1>, <yes|no>, <false|true>, or <off|on>.

mainfontitalic <0|1>

Specify the italic setting in the main *G7* window. Options may be specified as <0|1>, <yes|no>, <false|true>, or <off|on>.

mainfontunderline <0|1>

Specify the underline setting in the main *G7* window. Options may be specified as <0|1>, <yes|no>, <false|true>, or <off|on>.

mainfontstrikeout <0|1>

Specify the strikeout setting in the main *G7* window. Options may be specified as <0|1>, <yes|no>, <false|true>, or <off|on>.

(matd)ata

(matu)pdate

This command brings data into *G7* in a rectangular format, which easily could be prepared using Excel or OpenOffice. There are three alternative forms in which the series can be arranged. The series can be arranged either in vertical columns with the name of the series at the top of the column, or in horizontal rows with the names appear on the same line of the *matdat* command. Up to 20 columns of numbers occupying up to 160 characters per line may be given. The *matdata* command is used to provide data for a new series. If the series already is present in the databank, it will be overwritten. The *matupdate* command requires that the data already are present in the workspace bank, and it merges the new data with the existing series. The three forms of the command are:

Form 1:

```
matdat
          gnp    c     cd    cnd   cs
   1981.1  1513   950   146   359   445
   1981.2  1512   949   140   361   448
   1981.3  1522   956   143   361   450 ;
```

Dates appear as the first number on each line. Here 1513 is the value of “gnp” in 1981.1, 1522 is the value in 1981.2, etc.

Form 2:

```
matdat 1981.1
   gnp    c     cd    cnd   cs
   1513   950   146   359   445
```

(continues on next page)

(continued from previous page)

```
    1512    949     140     361     448
    1522    956     143     361     450 ;
```

The date of the first observation appears on the command line, and no dates appear on subsequent lines. Use a semicolon to end the data. (You must have a semi-colon on the last line of data for either form.)

Here is yet another version of legal format for *matdata*.

Form 3:

```
matdat  gnp out(1-4) 1981.1
    1513    1512    1522
    950     949     956
    359     140     143
    359     361     361
    445     448     450;
```

Here, series names are given immediately after *matdat* with starting date given as the last argument on the *matdat* line. The first series reads from the first line after the *matdat*, the second series reads from the second line, and so on. Note also that group definitions are allowed in this version.

Related Topics: *data*

matin <matrix_name> <year> <firstrow> <lastrow> <firstcol> <lastcol> [<skip>] [form]

The *matin* command reads matrix data for one year and stores it in a Vam file. For it to work, a Vam file already must be assigned and set as the default Vam file. Then a rectangular array must follow with numbers matching the <firstrow>, <lastrow>, <firstcol>, and <lastcol> parameters of the command. As in the *vmatdata* command, the optional <skip> parameter is the number of spaces to be skipped in reading each line. The <skip> parameter and the <form> line work together exactly as for *vmatdata*: the absence of a <skip> parameter indicates the presence of a form line. A '#' in the first space of a line means to skip the whole line. Use of *matin* does not affect entries outside the specified area, so it can be used to update a matrix as well as to introduce it originally.

Example:

```
matin govstr 1987 1 25 1 5 4
    1     48      0       0       0      61
    2    495      0      32       0     453
    3      0      0       5       0       0
    4   1344      0       0       0    1651
    5    812      0     201       0       0
    6    859    555       0       0    3012
 ...
```

The above example actually is used to introduce the distribution of government structures by 25 categories to 5 government types in *IdLift*. The data presented is for 1987, and covers rows 1 to 25 and columns 1 to 5 of the matrix. Four columns are to be skipped at the beginning of each line, and this is where we keep the row number, which serves to make the file easier to read.

Related Topics: *matin5*, *pmatin*, *pmatin1* *vmatdata*

matin5 <matrix_name> <year>[<width> <decs> <IndexWidth>]

The *matin5* command is another command for reading data for a matrix. These data are in a format called "punch5." This is an input format for matrices that is useful when a significant number of the matrix elements may be zero. On each line, there are 5 cells (hence the name), and for each cell, the row, column and value are specified. For this command to work, a Vam file must be assigned and set as the default Vam file. In the above syntax, <matrix_name> is the name of the matrix as shown in VAM.CFG, <year> is the year of the data, <width>, <decs> and <IndexWidth> are optional arguments that specify the width of the data field for each cell, the number of decimal points, and the width of the field for the row and column indexes. The end of the matrix data is signaled either by a ';' in the first space on a line or by the end of the file.

Here is a sample from the beginning of an A-matrix:

```
matin5  am 1977 9 5 3
# A-matrix for 1977. 83 rows and 83 columns.
AM     1  1 0.245790  1  4 0.000220  1  8 0.000410  1  9 0.281300  1 10 0.058360
AM     1 11 0.002070  1 12 0.005680  1 13 0.000380  1 15 0.001700  1 16 0.003060
AM     1 22 0.089770  1 24 0.000070  1 48 0.001210  1 49 0.000130  1 50 0.000030
```

Related Topics: *matin*, *pmatin*, *pmatin1*, *punch*, *punch5*,

matty | matpr [file <file_name>] <date1> <date2> [<option>] <series1> <series2> <series3> .... <seriesN> ;

The *matty*, or *matpr*, command creates or overwrites a file of data in parallel columns for input into spreadsheets or other software. The <file_name> is the name of the file created (if given it must be preceeded by "file") and <date1> and <date2> are the starting and ending dates of the transfer. The only option currently available is the "dump" option, which prints data in a compact format. Up to 100 series may be transferred in one file. If no filename is given, the data are written to the screen and to the "save" file, if one currently is open. Width and precision are set by the current values given by either the *format* command or the *width* and *decs* commands. The year format is given by the *yearformat* (*yf*) command.

Example:

```
#  MATTY.TST -- Test the output of the "matty" command.
hbk quip a
matty file nipa.prn 1985.1 2000.2
gnp c v g;
```

Related Topics: Writing Data from *G7*, *gridtype*, *p123*, *rp123*, *format*

maxobs <number>

This command sets the maximum number of observations that can be in any series in the workspace bank.

Example:

```
maxobs 1500
```

allows series with up to 1500 observations. The default value of *maxobs* is 600. Increasing the value slows down most data operations and may reduce the number of variables that can be included in any regression. If long series are to be stored in the workspace bank, the maximum number of observations in this bank should also be increased. This number is set at the end of the G.CFG file.

Note that you may delete the workspace bank and set all new parameters all in one fell swoop with the *zap* command.

Related Topics: G.CFG file, *zap*

mcopy [bank letter.]<destination> = [bank letter.]<source> [<period>]

The *mcopy* command copies the matrix (or vector) from the source to the matrix (or vector) destination. If no bank letters are specified, then both the source and destination are assumed to be in the default Vam file. If a value is given for [period] then only the values for that period are copied. Otherwise, the starting period and ending period are determined by the *fdates* command.

Example:

```
vam \idlift\devel\hist b
vam \idlift\model\hist a
fdates 1972 2040
mcopy a.am = b.am
```

This example copies the matrix "am" from the *IdLift* development Vam file to the production Vam file for the period 1972 to 2040.

Note that the *vc* command also will copy vectors within the same Vam file, and the *vc* command also is capable of evaluating certain algebraic equations involving matrices.

Related Topics: Matrix Operations, *madd*, *mmult*, *minv*, *linv*, *mtrans*, *vc*

(mg)raph <name1> [<name2>] [<name3>] ... [<name7>] <date1> <date2> [date3]

(mg)raph (<name1>) [(<name2>)] [(<name3>)] ... [(<name7>)] <date1> <date2> [date3]

Consruct a Multi-scale Graph. This is exactly like the *graph* command except that the series has its own vertical scale chosen so that its graph extends from the top to the bottom of the screen. The scale shown on the left is for the first series; the scale shown on the right is for the second. If more than two series are displayed, the other series will also be plotted on their own scale, but that scale will not be displayed. Algebraic expressions may be provided in place of some or all of the series names, so long as the expressions are enclosed in parentheses.

Related Topics: Drawing Graphs, *graph*

minv [bank letter.]<matrix A> [period]

The *minv* command performs in-place matrix inversion. It works over the current *fdates*, or for the <period> specified.

Related Topics: Matrix Operations, *linv*, *madd*, *mcopy*, *mtrans*, *mmult*, *vc*

missing <y | n> [<missing_text>]

This command does two things. First it tells *G7* whether to rely on missing values to be present when performing regressions, interpolations, or other functions. When *missing* is set to "y", then *G7* will complain if missing values are in any of the data in a regression. Functions such as *@benchmark* and *@lint* rely on missing values to tell which periods of a series need to be filled. If *missing* is "n", then *G7* treats zeroes as missing values, and the regression command will not complain if there are zeroes.

The second function of the *missing* command is to supply some text to be displayed when missing values are present. A missing value is equal to -0.0000001, so by default it will be displayed as -0.0. However, by using the *missing* command, you can tell *G7* to display missing values as "miss", "N/A", or some other character string. Just make sure that the <missing_text> is short enough to line up well with the other data being displayed.

You can convert the zeroes in a series to missing values by using the *@miss()* function. Conversely, you can convert missing values to zeroes using the *@zeroes* function.

Related Topics: Functions in *G7*, Missing Values

mmult [bank letter.]<matrix A> = [bank letter.]<matrix B>*[bank letter.]<matrix C> [period]
mmult [bank letter.]<matrix A> = [bank letter.]<matrix B>'[bank letter.]<matrix C> [period]
mmult [bank letter.]<matrix A> = [bank letter.]<matrix B>/[bank letter.]<matrix C> [period]
mmult [bank letter.]<matrix A> = [bank letter.]<matrix B>&[bank letter.]<matrix C> [period]

The *mmult* command is actually a family of commands, with four operations available:

1. The asterisk (*) operator is matrix multiplication. If the matrices are conformable, then

   ```
   mmult A = B*C
   ```

   multiples matrix B by matrix C and stores the result in matrix A. If a date is specified on the command line, the operation will be done for that period only. Otherwise, it is performed over the current *fdates*.

2. The apostrophe (') operator forms the transpose of the first matrix and then calculates the matrix multiplication. If the matrices are conformable, then

   ```
   mmult d.A = e.B'e.C
   ```

   multiples the transpose of matrix B in bank 'e' by matrix C in bank 'e' and stores the result in matrix A in bank 'd'.

3. The slash (/) operator does element-by-element division. So,

   ```
   mmult A = B / C
   ```

   divides each element of B by the corresponding element in C and places the result in A.

4. The ampersand (&) operator does element-by-element multiplication. So,

   ```
   mmult A = B & C
   ```

   multiplies each element of B by the corresponding element in C, and places the result in A.

Related Topics: Matrix Operations, *linv*, *madd*, *mcopy*, *minv*, *mtrans*, *vc*

mode <test | forecast> or mode <t | f>

mode determines what is to be done between <rdate2> and <rdate3>. The default mode is test. In that case, the values "predicted" by the equation for the period from <rdate2> to <rdate3> are compared to the actual data and SEE and MAPE parameters are computed for the test period. For forecast mode to work, the series for all independent variables except lagged values of the independent variable must extend to <rdate3>. Then, in forecast mode, two forecasts are made for the period from <rdate2> to <rdate3>. The first simply is the predicted value; the second is the predicted value plus the "rho adjustment" to take account of the error in the last period of fit.

Related Topics: Ordinary Regression, *limits*, *regress*

monup(mup) <series_name>

This command is used for updating a quarterly series with monthly data.

Example:

```
mup rtb
  1983.3  9.120 9.390 9.050   8.710 8.710 8.960
  1984.1  8.930 9.030 9.440;
```

"rtb" is a quarterly series being updated with monthly data. 9.120 is the value of rtb in July 1983; 9.390 is the value in August 1983, etc.. Note that quarterly dates are the first numbers on each line and three monthly observations must be present for each quarter.

move <matrix> <up|down|left|right> <first> <last> <newfirst> [year]

This command moves a block a rows up or down, or moves a block of columns left or right, where:

<matrix>
is the name of a matrix whose rows or columns are to be moved.

<up|down|left|right>
indicates the direction of movement

<first>
indicates the first row or column of the block that is to be moved.

<last>
indicates the last row or column of the block that is to be moved.

<newfirst>
indicates the first row or column to which the block is to be moved.

[year]
specifies a single year. If not specified, the function will do the move for all years between the active *fdates*.

Example:

```
move fact left 2 4 1 1997
```

In this example, columns 2 through 4 are moved left and overwrite columns 1 through 3. Column 4 will be zero.

Related topics: *coef*, *flow*, *rdras*

mtrans <matrix A> [bank letter.]<matrix B> [period]

The *mtrans* command takes the transpose of the second matrix (B) and places it in the first matrix (A). The two matrices can be in different Vam files, but A must be in the default Vam bank. If no bank letter are specified for B, then B is assumed to be in the default Vam file.

```
mtrans A e.B
```

sets A to the transpose of B, where B is found in bank 'e'.

Related topics: Matrix Operations, *linv*, *madd*, *mcopy*, *minv*, *mmult*, *vc*

3.13 *G7* Commands: N

nl or nlp <y> = <non-linear function involving n parameters, a0, a1, . . . an-1>
n
<starting values of the parameters>
<initial variations>

G7 offers two algorithms for non-linear regression. The *nl* command uses the simplex method (sketched below and not to be confused with the simplex method of linear programming) and the Powell method. The form for the two is exactly the same except that the first is invoked by the *nl* command, while the second is invoked by the *nlp* command. We illustrate the with *nl* here.

Example:

```
nl y = @exp(a0*@log(a1 + a2*x))
3
1.5  25.0  2.00
0.1   1.0  0.05
```

Since the non-linear function must be evaluated repeatedly, anything the user can do to speed the evaluation is helpful. For example, put all the variables involved into the workspace; make the workspace no larger than necessary; take any functions of variables, such as logs or squares, which do not change through the calculations and put the results in the workspace and use them instead of, say, taking the log over and over.

In the simplex algorithm, S, the sum of squared errors, initially is calculated at the initial value of each parameter. Then, one-by-one, the parameters are changed by adding the initial variations and S is recalculated at each point, thus yielding values at n+1 points (a simplex). By the algorithm sketched below, points in the simplex are replaced by better points or the simplex is shrunk towards its best point. The process continues until no point differs from the best point by more than one-tenth of the initial variation in any parameter. Each time a replacement is done, the simplex and the operation that yield it is reported on screen. Like all non-linear algorithms, this one is not guaranteed to work on all problems. It has, however, certain advantages. It is easily understood, no derivatives are required, the programming is easy, the process never forgets the best point it has found so far, and the process either converges or goes on improving forever. The display show the regression coefficients, their standard errors, t-values, and variance-covariance matrix. The Powell method also requires no derivatives and has the property of "quadratic convergence." That is, applied to a quadratic form, it will find the minimum in a finite number of steps.

Following the *nl* command, there can be *f* commands, *r* commands, and *con* commands before the non-linear equation itself. These may contain the parameters a1, a2, etc.; they should each be terminated by ';'. The non-linear search then includes the execution of these lines. E.g.:

```
nl f x1 = @cum(s,v,a0);
y = a1 + a2*x1
```

If the name of the left side variable is "zero", no squaring of the difference between the left and right side will be done. Instead, the squaring must be done by the *@sq()* function. This feature allows soft constraints to be built into the objective function. For example,

```
nl zero = @sq(y - ( a0 + a1*x1 + a2*x2)) + 100*@sq(@pos(-a2))
```

will require "softly" that a2 be positive in the otherwise linear regression of y on x1 and x2.

If the name of the left-side variable is “last”, no summing will be performed. Instead, the value of the function on the right at the “last” period will be minimized. This “last” period is the one specified by the second date on the preceding “limits” command. In order to combine summing of some components of the function on the right with the use of the “last” option, the *@sum()* function is used. It places the sum of the elements from the first date to the second date in the second date position of the output series, which is otherwise zero. This device is useful in some maximum liklihood calculations.

Both the simplex method and Powell’s method are described in the book, Numerical Recipes in C, by Wm. H. Press, et al., Cambridge University Press.

Sketch of Simplex Method of Non-linear Regression The method begins with a point in the parameter space of, say, n dimensions and step sizes for each parameter. Initially, new points are found by taking one step in each direction. Actually, each step is made both forward and backward, and the better of the two positions is chosen. This initial operation gives n+1 points, the initial point and n others. These n+1 points in n dimensions are known as a simplex. The algorithm then goes as follows.

```
Reflect the old worst point, W, through mid-point, M, of the other points
to R (reflected). (R is on the straight line from W to M, the same distance
as W from M, but on the other side.)

If R is better than old best, B {
   expand to E by taking another step of length MR in the same direction.

   if E is better than R,
      replace W by E in the simplex.
   else
      replace W by R. (reflected)
   }
Else if R is better than W {
   replace W by R in the simplex. (reflected)

   }
Else{
   contract W half way to mid-point of other points, to C. (contracted)
   if C is better than W,
      replace W by C.
   Else
      Shrink all points except B half way towards B. (shrunk)
   }
```

(norm)ality <y | n | f>

This command turns on (or off) tests of normality of the error terms. If the option chosen is ‘y’, then the Jarque-Bera test appears on the standard regression result screen labeled “JarqBer”. Under the hypothesis of normality, it has a chi-square distribution with two degrees of freedom.

If the option ‘f’ (for full) is chosen, then before the regression results are shown, a table appears with moments and measures of symmetry and peakedness, as well as the Jarque-Bera statistics. If a “catch” file active, this table will go into it.

Related Topics: Regression Tests

3.14 *G7* Commands: P

p123 <file_name> <date1> <date2> [width] [decs]
<ser1> [<ser2> <ser3> . . . <sern>] ;

p123 <file_name> <date1> <date2> [width] [decs] [<ser1> <ser2> . . . <sern>] ;

This command is similar to the *matty* command, except that the data are written directly to a Lotus worksheet. Do not include the WK1 extension with the filename; *G7* automatically appends the extension WK1. <date1> and <date2> are the desired starting and ending dates. Up to 239 series names can be entered in free format. End the list with a ';'. Width and decs are optional; width specifies the display width within Lotus for all series transferred, decs specifies the number of decimals displayed. Default values are 9 and 3 respectively. *p123* worksheets are compatible with Lotus 1-2-3, OpenOffice, older versions of Excel, and many other programs.

Example 1:

```
p123 natacct 60.1 91.1 8 1
   gnp c v vf vi fe fi g
```

Example 2:

```
p123 natacct 60.1 91.1 8 1 out(1-85)
```

Related Topics: Writing WK1 Files. *matty*, *xl*

pause ["<message>"]
pause [<seconds>]

The *pause* command instructs *G7* to pause while processing the script. If no arguments are specified, or if a message (in quotes) is given, then the user is prompted to continue or to cancel the processing of the script. If a message is given, then the message is printed with the prompt. If a number, either integer or real, is specified, then *G7* will pause for the given number of seconds before continuing.

pch <name> <number>

The *pch* command is an older form of the *ipch* command and is used to generate equation results for *SlimForp* style models. As with *ipch*, first an equation file is opened with the *punch* command, then, after doing a reression, *pch* is used to write lines to the equation file in the format expected by *SlimForp*.

Related Topics: *ipch*, *punch*

pmatin <matrix_name> <packed_matrix_filename>

The *pmatin* command introduces data for packed matrices to a Vam file. A Vam file already must have been assigned and set as the default Vam file. A packed matrix has all zero elements removed, and the format is efficient for presenting the nonzero cells of the matrix. The packed matrix data resides outside of the Vam file itself, and the Vam file contains only a pointer to the filename.

In the above syntax, <matrix_name> is the name of a matrix in the VAM.CFG file; it must have the letter 'p' in the "lags" position. The <packed_matrix_filename> is name the DOS name of the packed matrix file (these files always have the .PMX extension, but the .PMX should be included in the name.) The format of the data which follows is:

```
<year>
  <row> <num_non-zero_elements>
  <col1> <element1> <col2> <element2>  ...
```

For example:

```
pmatin bm bm.pmx
  1985
  1  5   # row 1 has 5 non-zero elements
  1  0.656  7 0.352  11 0.038   23 0.019  27 0.218
  3  2   # row 3 has 2 non-zero elements
  7  0.098  51 0.923
  ...
```

If the matrix is available for several years, the data for all years should be introduced with only one *pmatin* command, with each year's data preceded by the number of the year. It is essential that the year numbers be in 4-digit format, i.e. use 1987 please, not 87. If a cell is non-zero in any year for which there is data, the packed matrix will have a place reserved for it in all years.

Related Topics: VAM.CFG File, *matin*, *pmatin1*, *pmfile*, *pmpunch*

pmatin1 <matrix_name> <packed_matrix_filename>

An alternative way of introducing data into packed matrices is the *pmatin1* command. The format is just like that of the *pmatin* command, except that the data should be arranged in rectangular rows and columns like the *matin* command. For example, if you wish to treat as a packed matrix a 3 x 3 matrix known as B in the Vam file, you can introduce it with the *pmatin1* command as follows:

```
pmatin1 B B.pmx
1995
   1  0  0
   5  0  1
   6  2  0
1996
   1.5 0  0
   6   0  2
   7   3  1
```

Note that in the above example, only those cells that are zero in all years will be left out of the packed matrix. Cell (3,3) actually will be stored since it has a non-zero value in 1996. As with the *pmatin* command, you should put the data for all years you want to read in the same file, and use only one *pmatin1* command per matrix.

Related Topics: *matin5*, *pmatin*, *pmfile*

pmfile <matrix_name> <packed_matrix_filename>

The *pmfile* command associates a matrix name with a particular packed matrix file. This is useful when you need more than one version of a matrix. For example, one copy might hold coefficients, and the other copy hold flows. Or, suppose we wished to study the effects of changes in the A matrix and the A matrix was packed, then we would need two different .PMX files. If the original was called AM.PMX, then to create the alternative, say, AMALT.PMX, we would go to the DOS prompt via File | DOS and do:

```
copy am.pmx amalt.pmx
copy hist.vam vamalt.vam
```

Then from *G7's* command box do:

```
vam vamalt b
dvam b
pmfile am amalt.pmx
```

This *pmfile* command will both assign AMALT.PMX to be used whenever the am matrix is referred to, and put AMALT.PMX into AMALT.VAM as the name of the file to be used for the packed am matrix.

Related Topics: *pmatin*, *pmatin1*, *dvam*, *vam*

pmmode { "simple"| "advanced" }

Note that the <packed_matrix_filename> setting is limited to 30 characters, including any path specification. Sometimes more than one vam bank is loaded in *G7*, where the vam banks include packed matrix files with the same name but that are stored in different locations. Sometimes it is not convenient to reset the link for each vam bank using the *pmfile* command, or perhaps inclusion of a full path specification would be needed to distinguish between the files but the full specification exceeds the limit of 30 characters. In such cases, it might be useful to specify a link to each packed matrix file relative to the placement of the corresponding vam bank. By default, *G7* will assume that the packed matrix file is in the current working directory, or, if a relative path is specified, that the specification is relative to the current location. This behavior can be controlled with the *pmmode* command. The *pmmode simple* command corresponds to the default behavior. The "advanced" mode instructs *G7* to attempt to find PMX files in the same location as the corresponding vam bank, or if a relative path is specified for the PMX file, the root location should be the location of the parent vam bank.

Related Topics: *pmfile*, *vam*

pmpunch <filename> <matname> [decs]

The non-zero elements of the matrix <matname> are written into the named file as input for the *pmatin* command. The <decs> argument gives the number of decimal places. The years written are determined by the current values of *fdates*. This command especially is useful for converting a rectangular matrix in a Vam file into a packed matrix. One writes it out as an ASCII file with this command and then reads it in with the *pmatin* command.

Related Topics: *pmatin*, *punch*, *vp*

psras [-f] <matrix> [r <group_exp>] [c <group_exp>] <rowcon> <colcon> [year] <r | c> [cnt]

This "proportional scaling" routine is another form of the *ras command.*. However, in the case where some of the matrix cells are negative, the proportional scaling changes all elements in the same proportion.

The procedure can be seen most easily by writing down what the X vector is after scaling, call this X^*. This is determined in proportional scaling by the following equation:

$$x_i^* = x_i + \frac{|x_i|}{\sum |x_i|}\left(c - \sum x_i\right)$$

The intuition behind this scaling is that it preserves the proportion between the difference of the old and new valuesand the final scaled values. One of its strengths is that it is able to provide a result when for example, all the elements of X are negative, yet the control total is positive.

Related Topics: *Ordinary RAS*, *ras*, *rdras*

punch <punchfile | "off">

The *punch* command is used to open up equation files for writing using the *ipch* command. These files usually have the extension .EQN, and are used in *Interdyme* models to construct an Equation object, which contains regression parameters, values of rho, equation types and other information for multisectoral regression equations. To close the file, use "punch off".

The *punch* command usally is given at the beginning of a large regression add file. It often is useful to use the *fadd* command, along with an argument file that contains sector numbers, titles, equation types, starting dates, and other information that changes with each sector. Within the body of the *fadd*, a *(r)egress* command will be given followed by an *ipch* command. This will estimate the regression, and put the equation results into the equation file.

Related Topics: Interdyme, *fadd*, *ipch*, *pch*, *regress*, *vp*, *zip*

punch5 <filename> <vecname> [<field_width> <precision> <index_width>]
p5

This command prints a matrix or packed matrix to a file in *punch5* format. The years that to be written are determined by the current value of fdates. Each year is written with a header that is a *matin5* command, which tells *matin5* also what <field_width>, <precision>, and <index_width> to use. (The <index_width> parameter is the width of the row and column numbers in the file.) Thus, a file created with *punch5* can be read back into Vam using the *matin5* command. This provides a convenient way to convert packed matrices into normal matrices. The *punch5* format prints 5 non-zero matrix cells to a line, with row number, column number and cell value for each cell.

Related Topics: *fdates*, *matin5*

punchvec <filename> <vecname> <startyear> <endyear> [<field_width> <precision>]
pv

The *punchvec* command is used to print a vector to a file for all sectors for the years specified. The optional <field_width> and <precision> arguments again specify the field width and number of decimal places. The output file <filename> starts with a number of comment lines, telling the name of the vector, starting and ending years, and format information. Then one comment line gives the years as column headings. Each following line of the file contains the time series for one sector, with the name of the series, followed by the time series of data.

Related Topics: *vp*

pwd

The *pwd* command prints the path of the current working directory.

Related Topics: *cd*

3.15 *G7* Commands: Q

qpcon b <sign> [constant][*]ai [< > [constant][*] aj . . .]

This command establishes constraints on parameters for a linear regression equation. <sign> is '<', '=', or '>' ("<=" and ">=" also are accepted but are recorded as strict inequality constraints). <constant> terms are optional scalar multiples for the respective parameters. An asterisk may be included to clarify that the parameters are multiplied by the corresponding constant. The parameters are denoted by <ai>, where 'i' is the position in the regression equation of the corresponding variable. If a constant is included in the regression, then the parameters are denoted as a1, a2, ... Right-hand-side terms are separated by either a '+' or '-'.

Examples:

```
qpcon 25.0 > a1          # constrain the constant
qpcon 13.0 = 1*a2 + 1*a3 # constrain the sum of slope parameters

  Related Topics: Soft Constraints, :ref:`con <G7RMcon>`
```

quit

This command quits the execution of *G7*. Note that if any editor windows are open, they will be closed. If you have any unsaved changes in these files, *G7* will ask you if you want to save the file. Also, before exiting *G7* will store the workspace bank to file.

3.16 *G7* Commands: R

ras [-f] <matrix> [r <group_exp>] [c <group_exp>] <rowcon> <colcon> [<year>] <r | c> [<count>] [-<tolerance>]

where:

-f|c

specifies the flow (default) or the coefficient mode,

ras|psras|rdras

specifies the method of scaling to be used,

matrix

is the name of the matrix whose elements are to be balanced,

(r rgroup)
(c cgroup)

are options that determine which rows and columns are included,

rowctrl
colctrl

are vectors that contain the row and column totals,

yr

optionally specifies the year for which the matrix is to be balanced. If a year is not specified, the matrix is balanced over the period specified by the current *fdates*.

r|c

optionally specifies whether the sum of the row totals or the sum of the column totals controls the calculations if the two are not the same.

-maxiter

optionally determines maximum number of iterations. The default is 100.

-precon

optionally specifies that name of a file containing the preconditions that are to be imposed.

-tolerance

optionally specifies the tolerance for the convergence test. The default value is 0.000002.

Balance the named matrix by the rAs method so that matrix will have the row sum of <rowcon> and the column sum of <colcon>. It actually does not matter whether <rowcon> and <colcon> are columns or rows, though in the above equation they are to be thought of as columns. If the <year> parameter is missing, the command works over the current *fdates* range. If the sum of the elements of <colcon> does not equal the sum of the elements of <rowcon>, the vector to govern is specified by the ‘r’ or ‘c’ at the end of the command. The other vector will be scaled to have the right total.

The rows will then be scaled to get the correct row totals, then the columns scaled to get the correct column totals. Every five iterations, a report is printed with details on the row and column in which the maximum scaling occurs. When the maximum scale factor differs from 1.0 by less than 0.00002, convergence is declared to have been reached. The last scaling will be of the rows. The <count> parameter at the end determines maximum number of iterations; the default value is 100. The default tolerance is 0.000002, but an alternative may be specified. If convergence does not occur after the maximum number of iterations, the program reports the problem and continues to the next command. Once the balance is achieved, the columns are scaled to sum to 1.0. The reason for this normalizing and how to deal with it if it is not wanted is explained below.

The optional arguments [r <group_exp>] and [c <group_exp>] allow you to focus on doing a ras on a submatrix, where <group_exp> is a legal group expression. If you specifiy the 'r', the group expression will determine which rows of the matrix are scaled. If you specify the 'c', the group expression will determine which columns of the matrix are scaled.

In our experience, the matrices most frequently in need of balancing are the distribution matrices, such as those that determine how personal consumption expenditure by budget category is to be distributed to industries. For this reason, the *ras* command, after balancing, automatically ensures that the column sums are 1.0. Of course, if the matrix is the intermediate coefficient matrix, one does not want the columns to sum to 1.0. What do we do then? Let us suppose that am is an initial coefficient matrix, q, is the vector of outputs, cs is the vector of column sum controls, and rs is the vector of row sum controls. Then we would do:

```
flow am q
coef am cs
ras am rs cs r
flow am cs
coef am q
```

Note that the scaling algorithm used by the *ras* command is simple scaling. If the control total is C and the vector is X, then the scaling factor S is applied to each element, as is calculated as:

$$S = \frac{C}{\sum X_i}$$

Related Topics: *Ordinary RAS*, *coef*, *flow*, *psras*, *rdras*

rdras [-f] <matrix> [r <group_exp>] [c <group_exp>] <rowcon> <colcon> [year] <r | c> [<count>]

This command works identically to the *ras* command, except in the case where some of the matrix cells are negative. In this case, it performs right-direction scaling on the rows and columns. The essence of right-direction scaling is that both positive and negative numbers are scaled in the same direction. A scaling factor *s* is found such that when multiplying the positive numbers by *s* and dividing the negative numbers by *s*, the numbers add to the correct total. This algorithm also is used when applying group fixes in *Interdyme*.

Note that the right-direction scaling is a bit more complicated than the simple scaling technique used by the *ras* command. Right direction scaling preservers the relative ordering of the elements, and scales them all in the same direction, the "right" direction. Assume that A is the sum of all positive elements of the vector X, B is the sum of all negative elements, and C is the control total. The problem is formulated so as to find the number S for which the following equation is true:

$$A \times S + B/S = C$$

This can be rewritten as the quadratic equation in S:

$$A \times S \times S - C \times S + B = 0$$

To solve for S, we use the general quadratic formula to obtain:

$$S = \frac{C + \sqrt{C \times C - 4 \times A \times B}}{2 \times A}$$

Related Topics: *Ordinary RAS*, *move*, *psras*, *ras*

(rec)ursive <y> = <x1>, <x2>, <x3>, . . . , <xn>
rec <y> = ! <x1>, <x2>, <x3>, . . . , <xn>

This command is used to perform recursive OLS regression. This means that if we have limits ::

```
limits <date1> <date2> <date3>
```

then the indicated regression will be performed from <date2> to <date3>, then from <date2>-1(the period before <date2>) to <date3>, then from <date2>-2 to <date3>, and so on back to <date1> to <date3>. The regression coefficients are made into time series and stored in the workspace. <b1> is the series for the first regression coefficient; <b2>, for the second; and so on. The regression coefficients are stored in the date corresponding to the first date in their regression. Thus, b1{date1} would be <b1> for the regression over the period <date1> to <date3>. Similarly, the standard errors of the regression coefficients are stored in s1, s2, s3, etc.

If a *zip* command preceeds the *recur* command, no regressions will be displayed, but the *zip* command will be turned off at the end of the *recur* command, for you surely will want to create graphs after the *recur* and they cannot be made with *zip* on.

Example:

```
zip
limit 1980.1 1985.1 1991.2
recur gnp$ = g$,v$,fe$,fi$
gdates 1980.1 1985.1
fadd bounds.fad (1 - 5)
```

where BOUNDS.FAD is the file

```
ti Estimate and 2-Sigma Limits for Coefficient %1
f upper = b%1 + 2*s%1
f lower = b%1 - 2*s%1
gr b%1 upper lower
```

Related Topics: *limits*, *regress*, *zip*

(r)egress <y> = <x1>, <x2>, <x3>, . . . , <xn>
(r)egress <y> = ! <x1>, <x2>, <x3>, . . . , <xn>

Both forms of the *r* command regress the dependent variable <y> on the independent variables <x1>, . . . , <xn>. The first form automatically supplies a constant term, while the second does not. Independent variables may be expressions, as on the right side of an *f* command. The dependent variable should be a single variable. If the line ends with a ',' then the command continues on the next line.

If lagged values of the dependent variable occur among the explanatory variables, then they will be supplied automatically from previously calculated values when forecasting. Regression coefficients and other values are stored in the "rcoef" variable in the workspace bank.

Related Topics: Ordinary Regression, *catch*, *mode*, *punch*, *recursive*

(ren)ame <old_series_name> <new_series_name>

This command is usually used to rename a series in the workspace bank to another name. For example, it is used in the example on 2SLS to rename to predicted values from the first stage regression which will be used on the right hand side in the second stage regression.

Related Topics: 2SLS, and 3SLS, *del*

return [arg]

This command interupts execution of the *G7* script. If *G7* is in the midst of processing a *do* loop or an *add* file, then the arguments determine the next actions of *G7*. Arguments may be given in upper or lower case. Optional arguments are:

OK

The default argument. Exits a *do* loop or *add* file, but processing continues.

ESC

Processing halts entirely, but no error messages are shown.

ERR

Processing halts entirely, with error messages displayed.

Examples:

```
return ERR
```

(rows)cale <matrix> <row> [y|n]

This command is another method to convert flow matrices to coefficients. <matrix> is the name of a matrix whose rows are to be multiplied by the elements of the vector <row>. The last argument indicates whether or not to skip the diagonal elements. The default for this argument is 'n'. Note that this command is different than the *flow* command, as each element of the vector multiplies all elements in the corresponding row of the matrix instead of the corresponding column. This command thus could be used to implement across-the-row coefficient change, with the coefficient change indicator vector in <row>. The command takes effect over the current *fdates*.

Example:

```
rowscale am ami y
```

In this example, am is the A matrix. ami is a vector which indicates the time path of coefficient change and which is equal to 1 in the base year of the A-matrix. In the current example, we specify that the diagonal elements must not be moved.

Related Topics: *coef*, *flow*, *index*, *scale*

**rp123 <file_name> <date1> <date2> [width] [decs]
<ser1> [<ser2> <ser3> ... <sern>] ;**

rp123 <file_name> <date1> <date2> [width] [decs] [<ser1> <ser2> ... <sern>] ;

The format of the *rp123* command is identical to that of the *p123* command. The only difference is that the *rp123* command writes the series to the WK1 file by row. In other words, rows are series, and columns are dates. If you have very many series, with not so many observations (<250), *rp123* is more suitable than *p123*.

Related Topics: Writing Excel Files, Writing WK1 files, *matty*, *p123*

rs create <filename> [<Number_of_Columns> <Maximum_Number_of_Sectors>]

The *create* function opens the key file and reads in up to 20 columns of aggregation information (6 is the default). An optional argument can be supplied to limit the reading to more or less columns than the default. In principle, the routines handle aggregation or disaggregation between any two of the schemes read from the file. With six aggregation schemes, this results in 36 possible combinations of aggregation.

The beginning of a sample file is displayed below:

```
575 495 360 432  85 BEA82
  1   1   1   1   1 10100  Dairy farm products
  2   2   2   2   1 10200  poultry and eggs
  3   3   3   3   1 10301  Meat animals
  4   4   3   3   1 10302  Miscellaneous livestock-horses,bees,ho
  5   5   4   4   1 20100  Cotton
  6   6   5   5   1 20201  Food grains: wheat,rye,rice,buckwheat
  7   7   5   6   1 20202  Feed grains: corn,oats,barley,hay,sorg
  8   8   5   6   1 20203  Grass seeds
  9   9   6   7   1 20300  Tobacco
 10   10   7   8   1 20401  Fruits
 11   11   7   8   1 20402  Tree nuts
 12   12   7   9   1 20501  Vegetables
 13   13   7  10   1 20502  Sugar crops
```

The purpose of these tools is to provide a convenient way to aggregate and disaggregate data from various sectoral levels. The routine begins by reading a file that contains concordances between various sectoral aggregation schemes. The column heading on the first line must specify the number of sectors in that aggregation scheme. This number of sectors is also the "handle" that is used to refer to that column in the resector routines.

Please see the demo section of the Inforum web site for examples of the *resector* capabilities.

rs formagg <Number_From> <Number_To>]

This is the first function that should be called after creation. It performs the most important initialization tasks. It sets up all of the information that is needed to aggregate from the scheme indicated as <Number_From> to the scheme indicated as <Number_To>. It sets up concordance lists and "split lists" that will be used by other functions listed below. In some programs, you may need to issue this command several times, especially if you need to use some of the "cross-aggregation" techniques described below.

One of the functions of initialization is to set up lists of correspondences between sectors, as well as lists of splits, where one sector from one scheme corresponds to one or more sectors from the other scheme. Since this initialization is time and memory consuming, an explicit function called *formagg* performs this task, and this function is called for only needed aggregation relationships. *formagg* takes as its arguments the maximum sector numbers of the source and destination schemes. Until *formagg* has been called with a certain aggregation pair, no other functions using that pair are allowed.

rs aggvector <Number_From> <Number_To> <Input_Vector> <Output_Vector> <Split_Vector>]

This function is designed to aggregate or split a vector from one aggregation scheme to a vector of another aggregation scheme. <Number_From> should be the number of sectors of the source vector, and <Number_To> should be the number of sectors of the destination vector, as shown in the heading in the key file that was read from the create routine. <Split_Vector> must be of the same aggregation level as the destination vector. It is used by the routine where there is a one-to-many relationship going from source to destination sectors. If you purely are aggregating, the <Split_Vector> will not be used and its values may be set to arbitrary levels.

rs ctrlvec <Number From> <Number_To> <Detailed Vector> <Aggregate_Vector>
This function controls a detailed vector to a more aggregate vector.

rs rdctrlvec <Number_From> <Number_To> <Detailed Vector> <Aggregate_Vector>
This function controls a detailed vector to a more aggregate vector, using right-direction scaling.

rs crossaggvector <Number_From> <Number_To> <Number_In_Between> <Input_Vector> <Output_Vector> <Split_Vector>
Sometimes conversion of a sectoral scheme is more complicated than merely a combination of aggregations and splits. For example, there may exist a many-to-many relationship, where a <SplitVector> of either sectoring level would not provide enough information on how the flows should be allocated. The name "cross-aggregation" indicates the method of translation used by this function, whereby the source vector is split to the level of a more detailed intermediary vector, which then can be aggregated directly to the destination vector. It must be possible to aggregate the intermediate sectoring level both to the source and to the destination levels for this function to work.

In the argument list, <Number_From> is the number of sectors of the source vector, <Number_To> is the number of sectors of the destination vector, and <Number_In_Between> is the number of sectors of the intermediary vector. <Input_Vector> is the source vector and <Output_Vector> is the destination vector. In this function, <Split_Vector> is a vector of length <<Number_In_Between>> that is used to split the source vector.

rs aggmatrows <Number_From> <Number_To> <Input_Matrix> <Output_Matrix>> <Split_Vector>
rs aggmatrows <Number_From> <Number_To> <Input Packed Matrix> <Output Matrix> <Split_Vector>
This function is similar to *aggvector* but aggregates the <Input_Matrix by row to obtain the <Output_Matrix>. The routine also will accept a packed matrix as the input, though the <Output_Matrix> must be a full matrix.

rs aggmatrix <Number_Row_From> <Number_Row_To> <Number_Column_From> <Number_Column_To> <Input Matrix> <Output Matrix>
This function aggregates a matrix both by rows and columns. Earlier editions of the routine required both the source and destination matrix to be square, and the same aggregation mapping was applied to both dimensions. The current routine operates on rectangular matrices, and different aggregation mappings may be applied to the rows and columns.

rs ctrlmatrows <Number_From> <Number_To> <Input_Matrix> <Output_Matrix>
rs ctrlmatrows <Number_From> <Number_To> <Input Packed Matrix> <Output Packed Matrix>
This function controls a more detailed matrix to an aggregate matrix.

rs ctrlmat <Number_From> <Number_To> <Input_Matrix> <Output_Matrix>
rs ctrlmat <Number_From> <Number_To> <Input Packed Matrix> <Output Packed Matrix>
This function controls all cells within a block of the more detailed matrix to a single cell of the less detailed matrix. The function handles all of the different cases: single cell to single cell, row vector to single cell, column vector to single cell, and sub-block to single cell. The function is useful for

updating a more detailed matrix such as a benchmark IO table to a more aggregate (and more current) matrix. The routine might work only if the matrix is square.

rs crossaggmatrows <Number_From> <Number_To> <Number In Between> <Input_Matrix> <Output_Matrix> <Split_Vector>

This function works much like crossaggvector but aggregates rows of a matrix.

3.17 *G7* Commands: S

save <savefile | "off"> [<datacommand>]

The *save* command opens a file of the name <savefile> to receive output produced by many of the command of *G7*, including *type*, *f*, *fex*, *id*, *cc*, *(r)egress*, *matty*, and other commands. The save file is closed and saving is terminated by:

```
save off
```

The default flag "-w" causes a new file to be created. The optional "-a" flag causes *G7* to open an existing file and append text to the end. The save file represents a crucial step in the procedure of model building using the *Build* program. Normally, the procedure is as follows. One or more .REG (regression) files may be added during a *G7* session to estimate the equations of a model. Each .REG file will include a *save* command, to save the regression results, as well as the steps of forming variables, using the various flavors of the *f* command. These files are usually given the file extension .SAV. Then the .SAV files are added by *add* commands in *Build*, or they may included in an *Interdyme* model using *IdBuild*'s *iadd* command.

Note that the *save* command also is a powerful tool for using *G7* to create data files. You can use the *G7 format* command to control the width, decimal points, and observations per line of data series typed using either the *type* or *stype* commands.

The optional <datacommand> argument specifies how series will be printed out when using *type*. The default is for *G7* to create an *update* command, to use for updating series in *G7*. However, other legal values of <datacommand> are:

data
write *data* cards

update
write "update" cards

ovr
write "ovr" fixes

cta
write "cta" fixes

mul
write "mul" fixes

ind
write "ind" fixes

gro
write "gro" fixes

stp
write "stp" fixes

vdata
write *vdata* cards for Vam

vupdate
write *vupdate* cards for Vam

fix
write LIFT style index fixes.

dump
write data in compact format.

gf
record estimated regression equations in *G7* equation format.

Since Vam likes 4-digit dates, you can specify "yf 4" before using *save* to print *vdata* cards. Also, remember that you can insert comments into the save file using the *ic* command.

The "ovr", "cta", "mul", "ind", "gro", and "stp" options above are for writing macro or vector fixes for *Interdyme* models. Remember that if you write a vector fix then you will need to hand-edit the file and change names like "out5" to "out 5".

Related Topics: Build, Interdyme, Model Building, *format*, *ic*, *listnamescol*, *print*, *sprint*, *stype*, *yearformat*, *zip*

scale <control> <vectorname> [(<group>)] [<year>] [<rdscale>]
scale <control> <matrixname> <r rnum> [(<colgroup>)] [<year>] [<rdscale>]
scale <control> <matrixname> <c cnum> [(<rowgroup>)] [<year>] [<rdscale>]

This command can scale a vector, or scale a row or column of a matrix, where:

<control>
is a series from any bank (a.ctrl) or from any vector <b.vec2) or from any matrix(c.mat1.1) that contain the value to which the vector or matrix row or column is to be scaled.

For a vector:

<vectorname>
is the vector whose elements are to be scaled.

<group>
Optionally specifies which elements of the vector are to be scaled.

<year>
Optionally specifies a single year for which the elements of the vector are to be scaled. If not specified, the operation will be done over the current *fdates.*

<rdscale>
Optional argument of 'r' specifies that right direction scaling is to be used.

For a matrix:

<matrixname>
is the name of the matrix whose row or column is to be scaled.

<r rnum>
<c cnum>
specifies that a row or column is to be scaled and gives the number of the row or column.

<colgroup>
<rowgroup>
Optionally specifies which elements of the column or row are to be scaled.

<year>
Optionally specifies a single year for which the elements of the matrix are to be scaled. If not specified, the operation will be done over the current *fdates*.

<rdscale>
Optional argument of 'r' specifies that right direction scaling is to be used.

Example:

```
scale eqi2 bmf c 2
```

This example scales column 2 of the matrix bm to sum to the control eqi2, for all years in the current *fdates*.

Example:

```
scale cont mat r 5 (1-10) 2000
```

This example scales row 5, elements 1 to 10 of the matrix mat to sum to the control cont for the year 2000 only.

Related Topics: *coef*, *ctrl*, *flow*, *rowscale*

second <on | off>

This command is used in estimating systemic two-stage least squares. This procedure is simple. Estimate the equations of the model by OLS or other single-equation method. Build the model with these equations and simulate it over the historical period. The simulated values of the endogenous variables are not influenced by the errors in the structural equations. They therefore may be used as regressors in the second stage to obtain estimates free of simultaneous equation bias. There are two variants of this procedure. In the first, the simulation is done using the simulated values for lagged values of endogenous variables; the other uses the actual values of these variables.

To perform the first variant, perform the simulation without any special command, copy the bank of simulation results to the *G7* workspace, start *G7*, select the b option on the opening menu, do "bank bws" to put the actual values in the assigned bank, give the command "second on" so that *f* commands are ignored, and re-estimate the equations with the same add files as originally used. With *second* on, the dependent variable of the regression will be taken from the assigned bank; all others come from the workspace, which contains the simulated values. To turn off the *second* feature, the command is "second off".

Related Topics: Build, Model Building

seed(<integer>)

The *seed* command allows the user to reinitialize the random number generator. The random number generator is used with the *@rand()* and *@normal()* functions. The *@rand()* function draws from the uniform (0,1) distribution, and the *@normal()* function draws from the normal(0,1) distribution. If the *seed* command is called with identical arguments between calls to *@rand()* or *@normal()*, then these random number generators should return identical series.

Example:

```
seed(17568)
f uni1 = @rand()
seed(17568)
f uni2 = @rand()
```

In this case, the series uni1 and uni2 will contain identical elements.

Related Topics: Functions, Particularly the @rand() and @normal()

setveclag <vector> [<# lags>]

Get or set the number of lags for a vector in the default vam file.

(sgr)aph <series 1> <series 2>

(sgr)aph (<series 1>) (<series 2>)

Construct a Scatter Graph. This will display a scatter diagram of <series 1> and <series 2>. The graph is created with <line 1> characteristics. Use the "0 width" option on the *line* command to plot only scatter points; otherwise the points will be connected. Each point on the two-dimensional graph measures the value of the first variable on the vertical axis, and the value of the second variable on the horizontal axis. For use only with the *sgraph* command, there is an *hrange* command that works just like the *vrange* command but it controls the horizontal axis. Algebraic expressions may be provided in place of either or both of the series names, so long as the expressions are enclosed in parentheses. A common use of the *sgraph* command would be to show a Phillips Curve, as in the following example::

vr 0 2 4 6 8 10 12 hr 4 6 8 10 12 sgr infl unempc

Related Topics: Drawing Graphs, *graph*, *hrange*

(sh)ow <bank letter>.<vector_name> [<first_row> <first_period>] [-rt <filename>] [-ct <filename>]

(sh)ow <bank letter>.<matrix_name> < view> <line> [<first_row> <first_period>] [<-rt <filename>>] [<-ct <filename>>]

The *show* command shows vectors and matrices in a grid similar to a spreadsheet. If a vector is displayed, its values in successive years will appear as columns; dates run across the top and sector numbers down the side.

For matrices, the *show* command has second format. The "view" argument must be one of the following:

r
for row

c
for column

y
for year.

If view is 'r', then <line> is the number of the row to be displayed. (A row is displayed as if it were a column.) If view is 'c', then <line> is the number of the column to be displayed. If view is 'y', then <line> is the year number.

Examples:

```
show b.am r 5
show b.am c 7
show b.am y 1997
```

If <first_row> and <first_period> are specified for a vector, then the window will be scrolled so that the specified row and period will appear in the top left cell of the visible table. The same is available for displaying a row or a column of a matrix, where "view" is 'r' or 'c'. If a single period of a matrix is displayed, where "view" is 'y', then the two arguments specify the cell to display in the top left corner.

If the title files that are specified in the Vam file are not found, then generic row and column titles will be created and displayed. If an alternative set of titles should be displayed, and those titles are stored in text files, then the file names for these alternative row and column titles may be specified at the end of the show command; "-rt" or "rowtitle" (or "-ct" or "columntitle") and the filename indicate the appropriate file.

The Options menu item allows the user to control the number of decimal places, fonts, and colors of the display. The display can be copied to the clipboard in the usual way for Windows programs. The contents of the clipboard then can be copied into a spreadsheet such as Excel or into a table in a word processor. On the Copy menu, you may specify how much you want to be copied - just the numbers or also the frame. It is also possible to go in the other direction, from the spread sheet into the show window and thus into the Vam file.

Related Topics: *gridtype*

sma <top> <first> <last> <order> [f]

To easily estimate a polynomial distributed lag, in which the lag coefficients are softly constrained to lie along a certain degree of polynomial, use the *sma* command, described here.

The *sma* command imposes a series of constraints that "softly" require the regression coefficients between <first> and <last> to lie on a polynomial of degree <order>. If the 'f' is present at the end, the polynomial is "free" at the end; without the f, the polynomial will be assumed to be zero for the coefficient after <last>.

Examples:

```
sma 100 a4 a9 3 f
sma 100 a8 a16 1
```

Related Topics: Distributed Lags, Soft Constraints, *con*

stack

The format for *stack* is exactly like that for *sur* (see below) except that "stack" replaces "sur". With *stack*, no attention is paid to contemporaneous covariances. The point of *stack* is solely to impose soft constraints across regressions.

Limitation of *stack* and *sur*: *G7's* limitation of 500 variables, counting dependent, intercepts,and independent variables, can quickly become binding in *sur* and *stack*, since it applies to the total variables in all equations involved in the *sur* or *stack*. This is far less restrictive than the 30-variable limit for older versions of *G7*.

Related Topics: *sur*

stochastic < y | n >

Set stochastic to 'y' if you want to save the results of regressions as stochastic regressions. See the section on stochastic regressions for the details of how to do this.

Related Topics: Stochastic Regression

store

This command stores the currently loaded vector back to the Vam file with the modifications that have been made. This store is automatic when a new load or implicit load is encountered. When a *quit* is encountered with an unstored loaded vector, the program automatically will store that vector.

Related Topics: *load*

str <name> = <rhs>

str <name> += <rhs>

Create a string named <name> with definition <rhs>. If the string <name> exists already, then <rhs> may be appended with use of the '+=' operator. The string definition <rhs> may be specified as text presented within quotation marks, given as the name of a previously-defined string, or defined as a sequence of text and/or strings linked by '+'.

For example,

```
str concept  = "Output"
str industry = "Coal Mining"
str display  = concept + "of the " + industry + " industry."
```

str save <filename>

Store all defined strings in a text file <filename>. Afterward, the strings may be loaded into memory again.

For example,

```
str save MYSTRINGS.TXT
add MYSTRINGS.TXT
```

str store args <rootname>

Store all add-file or function arguments that are in scope in a series of strings. Each string name is created from the specified root name and the position (integer) of the argument in the argument list (e.g. root1, root2, . . .).

For example,

```
# Initialize a collection of series from a list of series names. Create new series or set existing series to zero.
function Initialize{
   str store args a
   do{ f %s( a%1 ) = 0 }(1 - %NARGS)
   }
Initialize gdp consumption investment
```

str print

Print each string to the screen.

str clear [<stringname>]

Erase all defined strings from memory. If a string name is given, then that string will be removed from the list of user-defined strings.

str replace [<stringname>] <"search_text"> <"replacement_text">

This function operates on the string <stringname>, if provided, or the last line read by "str getline". Any text in this string matched by the string <search_text> is replaced by the string <replacement_text>. Matching employs regular expressions as defined in the C++ Boost library. More details for regular expressions are provided on the following page.

Note: The Visual C++ Redistributable package must be installed in order to use some of the newest features. The VC++ installer should be available at C:PDGC++Install. Run this installer (run as Administrator if using Vista or Windows 7) before attempting to employ the "str replace" .

str open <filename>

Open a file for parsing with the string .

str close

Close the file that was opened with the *str open* .

str getline [<string_name>]

Read a line of text from the file opened by *str open*. Store the text as a string named <string_name>. Store a copy of the string with the name "line;" this string may be referenced by related routines, but it does not appear in the list of user-defined strings.

str parse [<string>] ["<separators>" ["<string_name>"]]

Split the string named <string> into words separated by any one of the characters listed as separators. If no string name is provided, then the routine acts on the last line read by *str getline*. The default separators list is composed of the space and tab characters, or " \t". The words are stored as a list of strings that may be accessed with the *%w()* function, which has a syntax similar to the *%s()* function described below. If a string name is suppied, it will be used as the root name of the stored words.

Note: The Visual C++ Redistributable package must be installed in order to use some of the newest features. The VC++ installer should be available at C:PDGC++Install. Run this installer (run as Administrator if using Vista or Windows 7) before attempting to employ the str replace.

(sty)pe <series_name> [<date1>] [<date2>]

Related Topics: *format*, *print*, *save*

(spr)int

"Silent" type writes the series to the currently open "save file" without displaying data on the screen. This can speed processing for long add files. Otherwise, it is identical to the *type* command.

Related Topics: *save*, *type*

(subti)tle <text>

Provides <text> as a subtitle on subsequent graphs. Use *subti* with no following text to remove the subtitle.

Related Topics: Drawing Graphs, *graph title*, *vaxtitle*

sur

The format for the *sur* command is shown by this example:

```
sur
r y1 = x11, x12, ...
...
r yn = xn1, xn2, ...
con 10 1 = a2 + 4b5 + etc
sma 100 a6 a12 1
do
```

After the *sur* command come all of the individual regressions. Any constraint or *sma* commands must come after the regressions. In the *con* and *sma* lines, a's denote coefficients in the first regression; b's coefficients in the second, and so on. Thus, a2 denotes the second coefficient of the first regression, and b5 denotes the fifth coefficient of the second regression.

The predicted values, dependent variables, and regression coefficients of the successive equations appear in the workspace file as predic1, depvar1, rcoef1, predic2, depvar2, rcoef2, etc. The results of the individual equations can be plotted by "gr *1", "gr *2", etc. The graph commands should follow the *do* command and each should be preceeded by an appropriate *title* command.

Related Topics: *stack*

3.18 *G7* Commands: T

tdates <date1> <date2>
tdates a

The *tdates* command sets the dates used by subsequent *type* (or *print*) commands.

For example, the command

```
tdates 1980.1 2010.4
```

means that the next *type* or *print* command will display quarterly data from the first quarter of 1980 until the fourth quarter of 2010. The *tdates* are set implicitly when you issue a *type* command with date arguments. They then are saved for the next *type* commands and not changed until you give another *tdates* or *ty* with date arguments.

```
tdates a
```

Selects "automatic" dates for *graph* and *type* commands. The automatic dates are the first and last date of the values actually present in the series. Automatic dates also adjust automatically to the frequency of the series.

Related Topics: Dates in *G7*, *look*, *gdates*, *print*, *type*

time

Display the date and time.

timer <on|off>

Calculates and display the length of time that passes between *timer on* and *timer off* commands.

(ti)tle <title for regression or graph>

Provides a main title that appears on subsequent graphs. Use *ti* with no following text string to remove the title. The title may be up to 90 characters in length. However, on graphs that might be too long. Note that titles can be passed as arguments when using the *add* or *fadd* command. In this case, the title must be enclosed within double quotes. Note that on graphs a *subtitle* command also may be given.

Related Topics: *add*, *fadd*, *graph*, *regress*, *subtitle*, *vaxtitle*

titpch [-<options>] [<stub_len>] ["str1" "str2" .."strn"]

The *titpch* command defines a header line that will be provided for a regression table, and also defines the regression output that will be displayed on subsequent *tpch* commands. Before using either the *titpch* or *tpch* commands, an equation table punch file (.EQP) must first be opened using the *eqpunch* command.

The <stub_len> is a number that indicates how long the "stub" or title for the line should be. You should ensure that the width you specify in the *tpch* command is the same, so that the table will line up properly. See the example below for the *tpch* command.

The option string starts with a '-' character, and may contain one or more of the following:

b
rbar-squared

o
rho

r
r-squared

s
see

d
double-line format

f
floating point f format, instead of g format.

"str1", "str2", .., "strn"
strings to be printed to explain the coefficients. Therefore, "str1" might be "intercept", "str2" is the second explanatory variable name, and so on.

Related Topics: *eqpunch*, *tpch*

tpch [<sur which>] <sector> [<"label">] [str_len] [(coef numbers)]

The *tpch* command is the command that writes a line of an equation table file (.EQP). In order to use this command, a file first should have been opened using the *eqpunch* command and a header definition supplied by the *titpch* command. The command line arguments of this command are as follows:

[<sur which>]
is used only when an equation has been estimated with the *stack* or *sur* commands to that the coefficients are in rcoef1, rcoef2, etc.

<sector>
is the number of the sector or category for which the equation is estimated.

<"label">
is a sector or category title in quotes. If used, you should also specify the length you want printed in the <str_len> argument

<str_len>
is the length of the sector or category label in the printout. Note that this should be equal to the "stub length" specified in the *titpch* command.

[(coef numbers)]
are used when there is a superset of regression parameters possible, and each equation uses some subset of that.

Example:

[eqpunch ven.eqp]

```
# Note: 30 is stub length.  Be sure to use the same for tpch!
titpch -rsf 30 const use usedif
add vena.reg   1 "Oilseed farming"
...
eqpunch off
```

[contents of vena.reg]

```
ti %1 %2
subti Inventory Change Regression
f usedif = use%1-use%1[1]
r ven%1 = use%1, usedif
#gr *
ipch ven %1 a
tpch %1 "%2"  30
```

Related Topics: *eqpunch*, *titpch*

try{. . . } catch{. . . }

The *try* command must be followed by the *catch* command. These routines loosely mimic similar routines in C++. Within the brackets following each command may be any collection of *G7* routines, including other *try-catch* blocks. If execution of any of the commands within the *try* block produces an internal error signal, then execution of the code in the *try* block ceases and execution of the code in the *catch* block begins. Otherwise, the *catch* block is ignored. For example, consider the link series command *ls*. The routine will fail if the value of the guide series is zero in the base period. Ordinarily, this will cause *G7* to stop execution of the script and report an error. If, on the other hand, the *ls* command is nested within a *try* block, then if an error occurs the alternative block of code following the *catch* command will be executed. In the following example, suppose that we prefer to use guide series y when it is available, and when it is not we rely on guide series z. We first try to extend series x using series y beginning in 2005, and if the value of y is zero in 2005, then we repeat the exercise to extend x using series z:

```
try{ ls x y 2005 f }
catch{
  try{ ls x z 2005 f }
  catch{
    ic Sorry.  No data in y or z.
    pause
    }
  }
```

The *try-catch* routine works well with many other commands, and it has a wide variety of useful applications. Note again that *try-catch* blocks can be nested, so that several alternatives can be tried before *G7* gives up in despair.

(ty)pe <series_name> [<date1>] [<date2>]
(ty)pe (<expression>) [<date1>] [<date2>]

(pr)int

Displays values for the named series from <date1> to <date2> on the screen. The first number on each line is the date of the first observation on the line. Dates <date1> and <date2> may be omitted if they are unchanged from the previous *type* or *tdates* command. If the *save* command is on, then the displayed values are transferred to the save file where they are preceeded with an *update* command by default. The word *update* can be replaced by other commands, such as *data*, *vdata*, and various fix commands, by giving the appropriate word as the third argument of the *save* command.

If an expression is specified between parentheses, where the expression is in the standard *f* command or *vf* command syntax, then it will be evaluated. The results will be printed, but the results will not be stored to the workspace or default vam file.

If you are printing a lot of data to a file, then you might want to use the *sty* command which will save some time by not writing output to the screen.

Note that the formatting of the *ty* and *sty* commands can be changed with the *format* command; *format* controls the width, precision, and number of data points per line printed. This especially can be useful when printing data for reading with programs that read fixed width data fields.

Related Topics: *addtype*, *format*, *save*, *stype*, *tdates*

3.19 *G7* Commands: U

(up)date <series_name>

This routine works like the *data* command but instead updates an existing series. The data following the command contain only the data points to be changed. The updated series is placed in the workspace only. Note that if you skip values as in the example below, then they will be assigned as missing values.

Example:

```
update um
  1994.1  34.2 35.3
  1995.1  37.0
  1997.1  38.
```

You later can interpolate the missing values for a series by using the *@lint* function.

Related Topics: Missing Values, *data*, *vdata*

3.20 *G7* Commands: V

vam <bankname> [<location>]

This command has the same format as *bank*, and assigns a Vam file at the default location. The <location> must be a letter. If <location> is not given, then it is assumed to be 'a', which is the first slot. Up to 25 data banks of the various types may be assigned, in positions 'a' through 'z' except 'w'.

The equivalent function can also be reached via the *G7* menu by selecting Bank | Assign menu item and choosing Files of type "Vam Bank".

Related Topics: Assigned Banks, *bank*, *cbk*, *dfr*, *dvam*, *hbk*, *listbanks*, *pmfile*, *vf*, *vup*

(vamcr)eate <vam configuration file> <vam file>

This command is used to create a Vam file from a Vam configuration file the command in *G7*. For example, to create the Vam file HIST.VAM from the configuration file VAM.CFG, the command is::

```
vamcreate vam.cfg hist
```

At this point, the newly created Vam file has zeroes for all of its data.

Related Topics: *dvam*

vammode [a | s]

The simulation files from *Interdyme* models usually consist of a logical pair. The Vam file contains the matrices and vectors, and the *G7* bank contains all macro variables. If you use the *Compare* program to make tables and ask it to read data from a Vam file, then it will treat the Vam file – *G* bank pair as one logical entity. *G7* on the other hand, can treat the two as one logical bank like *Compare*, or require that you load them as separate banks. The first option, which is the default, is 'a' (advanced mode), and the alternative option is 's' (simple mode).

(vaxl)ab <in | out>

This controls the position of the vertical axes numbers, causing them be printed inside or outside of the axes. The default is "vaxl in".

Related Topics: Drawing Graphs, *graph*, *vaxtitle*, *vr*

(vaxti)tle <text string>

This provides a vertically printed title for the y (left) axis. Use *vaxti* with no following text to remove the left axis title.

Related Topics: Drawing Graphs, *graph*, *subtitle*, *title*, *vaxlab*

vc <vector_name> = <expression>

The *vc*, (vector calculate) command evaluates the expression on the right and puts the results into the named vector. Only a few matrix and vector operations are implemented with *vc*. It can add or subtract any number of vectors and multiply a matrix times a vector. It cannot yet add or subtract matrices, and parentheses are not supported yet. However, scalar values (either constants or *G7* bank variables) can be used to premultiply or postmultiply a vector or matrix. Also, a scalar value can appear alone on the right hand side of a *vc* equation, which indicates that the entire vector will be set equal to that scalar value. Between a matrix and a vector, an * means matrix-vector multiplication. Between two vectors, the * means element-by-element multiplication. Between a vector and a matrix, the / means multiplication by the transpose of the matrix or vector on the left. Between two vectors, the / means element-by-element division. The *vc* command is performed only over the interval defined by the current *fdates* command.

Example: If am and cm are matrices and out, pdm, qcu, and cprices are vectors, we could write

```
vc out = am*out + out       # Matrix multiplication, vector addition
vc qcu = out*pdm            # Element by element vector multiplication
vc out = qcu/pdm            # Element by element vector division
vc cprices = pdm/cm         # This is actually pdm' * cm.
vc out{2017-2020} = qcu/pdm # Perform calculation over years 2017-2020.
vc const = out{2017}        # Set LHS series to a single year of the RHS.
```

The <vector_name> must be a valid vector name in the Vam file, as must be all names in the expression.

Related Topics: *linv*, *mcopy*, *minv*, *mmult*, *mtrans*

vdata <series_name>

This command works just like *data* except that the introduced series goes to the default Vam file. Recall that if fd is a vector in the Vam file, fd23 will be the 23rd element of it. If am is a matrix, am12.14 is the element in row 12, column 14.

Related Topics: *data*, *update*, *vf*, *vupdate*

vf <name> = <expression>
vf <name>{<date1> [- <date2>]} = <expression>
vf <name> <operator> <expression>

This command is exactly like the *f* command except that the destination for the calculated series is the default Vam file. Therefore, the <name> must be a legal name, i.e., a vector defined in that Vam file, with a sector number within the range of elements for that vector. A range of dates also may be provided to specify the periods to carry out the calculations.

Example:

```
ba outemp a
vam hist b
dvam b
vf out1 = out1
```

This seemingly tautological example actually does something useful. It will look first in the workspace bank for a series named out1, and if it fails to find it, will look in the bank assigned in position 'a', which is OUTEMP.BNK. Let's say it finds the series there. It will then copy this series over to the default Vam file, which is HIST.VAM. The right hand side <expression> is any valid *G7* expression, and the operation is effective over the current *fdates*.

In addition to the assignment operator (i.e. "="), the *vf* command can add <expression> to the left-hand series using the "+=" operator, or it can subtract <expression> from the left-hand series with the "-=" operator, or multiply with "*=" or divide with "/=". These operators follow the syntax for the C++ programming language, though here they operate over a range of values.

Related Topics, *dvam*, *f*, *vam*, *vdata*, *vupdate*, *vup*

vmake [<startyr> [<endyr>] <target> <source> . . .
vplus [<startyr> [<endyr>] <target> <source> . . .
vminus [<startyr> [<endyr>] <target> <source> . . .

The *vmake* command is used to make a vector out of other vectors, or a matrix out of vectors. The *vplus* and *vminus* variations are identical to *vmake* except that they add or subtract the *source* quantities to the *target* quantity. The <startyr> and <endyr> parameters are optional. If not given, then the process will work over the entire range of the Vam file. Each matrix, whether it is a target or a source, is represented by:

MatrixName(r|c <lines>)

where 'r' or 'c' indicate row or column and <lines> are the selected rows or columns. If the <lines> specification is omitted, then the default is all rows or columns. Each vector in the list is represented simply by its name. If the total number of lines in the sources is different from that of the target, then *vmake* uses the smaller number. The target may be a packed matrix, in which case the packed matrix file must have already been created, and the positions of the non-zero elements already determined. In this case, *vmake* will only store elements in the non-zero positions.

Each vector, whether it is a target or a source, is represented by:

VectorName(v <lines>)

where 'v' indicate that a vector is specified and <lines> are the selected elements. If the <lines> specification is omitted, then the default is all elements. Each vector in the list is represented simply by its name. If the total number of lines in the sources is different from that of the target, then *vmake* uses the smaller number. The target may be a packed matrix, in which case the packed matrix file must have already been created, and the positions of the non-zero elements already determined. In this case, *vmake* will only store elements in the non-zero positions.

The first example puts the vectors a, b and c in rows 1, 2 and 3 of matrix A.

```
vmake A(r 1-5) a b c
```

The next example puts column 3 of A into the vector c.

```
vmake c A(c 3)
```

The final example puts fows of B starting from row 1 followed by the vector c into rows 2 to 100 of the matrix A.

```
vmake A(r 2-100) B(r) c
```

Related Topics: *vmake*

(vmatd)ata <r|c> <nv> <nyrs> <first> <last> [skip]
<year> <vec1> <vec2> . . . <vecnv>

<vec> <year1> <year2> . . . <yearnyrs>
[form]

The *vmatdata* command puts whole vectors into the default Vam file. It can be used to bring in data in a variety of formats commonly used by statistical agencies in releasing input-output tables. It can read a file which shows any one of the following:

- one vector in columns for several years
- one vector in rows for several years

- several vectors in columns for one year
- several vectors in rows for one year

The *vmatdat* command requires at least two lines and three if a format for reading is specified. The form of the first line is always the same, as is the form of the third line, if present. The second line has two different forms, one if several vectors are being read for one year and another if data for one vector is being read for several years.

In the above syntax, the arguments have the following meaning:

<r|c>
is 'r' or 'c' according to whether the vectors are rows or columns.

<nv>
is the number of vectors.

<nyrs>
is the number of years.

<first>
is the position in the vector in the Vam file of the first item in the data which follows.

<last>
is the position in the vector in the Vam file of the last item in the data which follows.

<skip>
is the number of spaces on each line which should be skipped before beginning to read data. If no value for <skip> is provided, then a form line (explained below) will be expected as the third line. If no spaces are to be skipped and no form line provided, then give skip a value of 0.

<year>
is the year of the vectors or the first year if nyrs > 1.

<vec1>
is the name of the first vector, vec2 is the name of the second vector, etc.

<form>
If no value for skip is given, then this form line must be provided. It should have the letter r in every position which is to be read in the following table. This device allows direct reading of tables that have columns of data separated by | or other symbols or which have subtotals which are not to be read.

A typical form line and a matching data line are:

```
form:                rrrrrrrrrrrrrrrrrrrrr       rrrrr     rrrr

data:Agriculture     | 3450  3718  249   0  | 6780|  8102 |**| 1598
```

Columns which do not have an r are simply totally ignored. Consequently, they cannot be used to separate data fields. Thus, reading the following

```
rrrrrrrr    rrrr
  851       2971
```

would produce one number, 8512971, not two numbers, 851 and 2971.

The first version of the second line is used if there are several vectors for one year; the second is used when giving data for one vector in several years. In both cases, the item of which there is only one is specified first and then the items of which there are several.

Any line beginning with a # is skipped completely. For example, the lines

```
# Example of several vectors in columns for one year:
vmatdat c 6 1 1 57 3
1986   cons   gov   inv   stk   exp   imp
1        21     0     2    -3     6     7
...     ...   ...   ...   ...   ...   ...
57       38   401     0     0    17    15
```

would read the 6 columns of data into positions 1 through 57 of the named vectors (cons, gov, . . .) for the year 1986. Positions 1 - 3 of each line, which contain the sector number for easy visual reference, are skipped.

Here is another example which reads the vectors as rows, as may often be the case when reading printed tables of primary inputs.

```
# Example of several vectors in rows for one year:
vmatdat  r 6 1 1 6  16
1986 labinc propinc deprec inter profit tax
#                    1     2     3     4     5     6
labor income      7303    21   368  1203  1302   820
proprietor inc    1215     9   100   107   303    92
depreciation       950     3    82   104   121    73
interest income    845     7    83    54    95    52
profits             50     3    25   217   337    38
indirect taxes     121     1    32   178   294    29
```

Note the use of the line beginning with a # to provide labels for the columns.

The second form of the second line is illustrated by the following example, which reads one vector, cons, for five years.

```
# Example of a single vector in columns for several years:
vmatdat c 1 5  1 57 2
cons 1985 1986 1987 1988 1989 1990
 1    31.8 37.2 32.5 38.7 41.2 43.1
...
57    1.2  1.7  1.8  2.0  2.1  2.2
```

Finally, here is an example which reads a single vector in columns for several years.

```
# Example of reading a single vector in columns for many years.
vmatdat r 1 28 1 8  12
demog 1989 1990 1991 1992 1993 1994 1995 1996 1997
#  Date  ncent south  west college  twoy  fs1   fs2   fs5   head1 head3
 89.000  0.243 0.345 0.208  0.220  0.469 0.243 0.323 0.106  0.274 0.350
 90.000  0.241 0.346 0.209  0.222  0.476 0.250 0.320 0.105  0.267 0.350
 91.000  0.240 0.347 0.211  0.224  0.482 0.247 0.320 0.105  0.263 0.348
 92.000  0.238 0.348 0.212  0.226  0.487 0.245 0.319 0.104  0.260 0.346
```

It must be emphasized that, precisely because it can read in free format, the *vmatdat* command cannot interpret blanks as zero entries. There must be a numerical value for all cells in the tables, especially the 0's.

The flexibility of *vmatdat* often makes it possible to read in final demand columns or primary inputs as they appear in printed tables provided by statistical offices.

Related Topics: *matin*

vp <vector_name> [r|c] [field_width] [decimal_points]
vp <matrix_name><view><line> [field_width] [decimal_points]

The *vp* command writes the named matrix to the currently open *save* file. As you can see, this command has a format and options like those of the *show* command.

Example:

```
save am.dat
vp am y 2000  9  6
save off
```

Related Topics: *pmpunch*, *punch*, *punchvec*

(vr)ange <bottom> [top]
vr <bottom> [<line1>] [<line2>] ... [<line8>] <top>
vr off

The *graph* command normally sets the vertical range so the series extend from the bottom to the top of the graph. Sometimes, it is desirable to set the range independently. This is done with the *vrange* command. For example "vr 0 100" will make all subsequent graphs have 0 at the bottom and 100 at the top until the "vr off" command is encountered, restoring *G7* to its normal mode. If the <top> is omitted, then only the bottom of the graph is set and the program finds the top so that the graph fits on the screen.

If one or more of the optional [line] entries are present, horizontal lines will be marked at those levels. The lighted pixels which mark these lines are located above the long marks on the horizontal axis, so they also serve as meaningful vertical lines.

Related Topics: Drawing Graphs, *graph*, *hrange*, *line*, *vaxlab*

vtitle <title>

This command allows you to record a title for the default Vam file. This title is displayed when using *Compare*.

vupdate

Related Topics: *vdata*

vup

This command works just like *update* except that the series goes into the default Vam file. For this to work, a Vam file must be assigned with *vam*, and it must be set to the default with *dvam*.

Related Topics: *dvam*, *vam*, *vf*

3.21 *G7* Commands: W

(wri)ndex <y | n>

G7 normally writes the entire index file of the workspace as each variable is added. When many series are added, this feature lengthens processing time. To defer writing the index until the entire add file is processed, “wri n” turns off writing and “wri y” turns it on again and writes the index. A “q” with writing off will cause the index to be written before quitting, so there is no danger of losing everything by forgetting “wri y”.

Related Topics: Tips and Tricks, Workspace Banks, *wsbank*

(wsb)ank <bank_name>

This command makes the named bank the workspace bank. Make sure that you have a backup of the bank that you assign as a workspace, since it will be modified if you create any data series during the current *G7* session. In addition, a “zap” command will destroy the bank.

In traditional *G7* usage, the workspace bank is called WS.BNK, and is in the current directory. However, by editing the first two lines of the G.CFG file you can change the default location and name of the workspace bank to be any bank you like.

Related Topics: The G.CFG File, Workspace Bank, *bank*, *wrindex*, *wsinfo*, *zap*

(wsi)info

This command prints information to the screen about the current workspace bank, including its bank title (if any), the number of series in the bank, the maximum number of bservations per series, and the default starting year (base year) and period.

Related Topics: *bank*, *wsbank*, *zap*

wsreset [<path>]
wsreset [<path>\g.cfg]

This command allows the workspace to be cleared and a new workspace established with new parameters and possibly in a new location. If “g.cfg” is included in the path, then *G7* automatically will attempt to open and read the file; otherwise, the configuration window will prompt the user to find and/or accept the settings in the optionally-specified directory.

3.22 *G7* Commands: X

xl

The *xl* command actually is a family of two or three word commands that enable the reading and writing of Excel worksheets from within *G7*. Actually, *G7* does not do the reading and writing itself, but communicates with the Excel program and provides instructions for Excel to execute. Therefore, Excel already must be installed on your machine before using these *xl* commands.

Next, some examples will be provided that show how to read from and write to Excel files. For details on each member of this family of commands, see the corresponding entries in this Reference Manual.

Example 1: In this example, the file "XLTEST.XLS" is created in the current directory. The time series are written to the spreadsheet by giving the starting cell, the direction to write the data (down or right), the name of the series, and the range of periods to write.

```
fdate 1975 2010                  # Create sample data.
f Year = 1974 + @cum(t, 1.0, 0.0)
f Data = 1975 / (t - 1975)

xl create xltest.xls            # Start Excel server, create xltest.xls workbook.
xl open worksheet 1               # Open worksheet 1.
xl write A 1 "Writing text to file:"
xl write A 3 "Year"               # Record label for the year
xl write A 4 down Year 1976 2010 # Record the year
xl write B 3 "Data"               # Record series name
xl write B 4 down Data 1976 2010 # Write series 'Data'
xl close                        # Close the workbook.
```

Example 2: In this example, we re-open the same file, and read one of the series back into *G7* with a different name.

```
xl open xltest.xls               # Open the workbook.
xl open worksheet 1                # Open worksheet 3.
xl read A 1 ""                    # Read the string in position A1; string
                               # will be printed on screen.
xl read B 4 right Year2 1976 2010 # Read data into workspace.
xl exit                          # Close workbook, close connection to
                               # Excel server.
```

Example 3: In this example, we must be working with a Vam file, and it must be opened and declared to be the default. The data read from the spreadsheet are stored as vector elements in the Vam file. Writing to the Vam file, instead of to the *G7* workspace bank, is forced by providing a bank leter ('c') corresponding to the open Vam bank in front of the name of the vector. Note that columns may be specified either by the Excel column letters or by the column number.

```
xl open C0301e.xls
xl open worksheet 1
do{
  xl read %1 27 down c.gdpN%2 1990 1990
  }(3-4 6-7 9-10)(1-6)m
xl exit
```

Example 4: In this final example, we show the use of the "xl matread" command. The command must all be on one line, even though it seems to span two lines in this example. First the blocks of data in the Excel spreadsheet are specified by listing their rows and columns. Next, the name of the matrix is given in which the data should be sotred, along with the matrix rows and columns. Finally, the year is given for which the matrix should be stored.

```
xl open C0319e.xls
xl open worksheet 1
xl matread c(2-18) r(14-17, 19-20, 22-24,26-29,31-32, 34-35)...
   c.AM c(1-17) r(1-17) 2000
```

Related Topics: 123toG, *p123*

xl open <filename>

This command opens an Excel file, either for writing or reading.

xl open worksheet <worksheet>

xl open chart <chart>

This command opens a worksheet or chart within the Excel file for reading or writing. The number to be used is the order of the worksheets within the file, where the first tab is ‘1’. The list of charts is handled separately, and the first chart also begins with ‘1’. Alternatively, the name of the worksheet or chart sheet may be specified. “worksheet” may be abbreviated “ws” and “chart” may be abbreviated “ch”.

xl create

xl create workbook [<filename> [<filetype>]]

xl create [before|after] worksheet [<worksheet>]

xl create [before|after] chart [<chart>]

An *xl create* command with no arguments launches the Excel server and opens a new workbook with one worksheet.

An *xl create workbook* command may provide the name for a new Excel filename. If no name is provided, then a new workbook will be created with the default name. The option “workbook” may be abbreviated “wb”.

The default filetype is XLS. Available file types include

File Types	
AddIn (.xlam)	WorkbookNormal (.xls)
CSV (.csv)	SYLK (.slk)
CSVMac (.csv)	Template (.xltx)
CSVMSDOS (.csv)	TextMac (.txt)
CSVWindows (.csv)	TextMSDOS (.txt)
DBF2 (.dbf)	TextPrinter (.txt)
DBF3 (.dbf)	TextWindows (.txt)
DBF4 (.dbf)	WK1 (.wk1)
DIF (.dif)	WK1ALL (.wk1)
Excel2 (.xls)	WK1FMT (.wk1)
Excel2FarEast (.xls)	WK3 (.wk3)
Excel3 (.xls)	WK4 (.wk4)
Excel4 (.xls)	WK3FM3 (.wk3)
Excel5 (.xls)	WKS (.wks)
Excel7 (.xls)	WQ1 (.wq1)
Excel9795 (.xls)	UnicodeText (.txt)
Excel4Workbook (.xls)	Html (.html)
IntlAddIn (.xla)	XLS (.xls)
IntlMacro (.xlsm)	

If no extension is provided with the filename, then the appropriate extension will be determined according to the file type and will be appended to the file name.

An *xl create worksheet* command may provide a name for a new worksheet to be added to the open workbook. The instruction "worksheet" may be replaced with "ws". An *xl create chart* command may provide a name for a new chart sheet to be added to the open workbook. The instruction "chart" may be replaced with "ch". If "before" or "after" is specified, then the new sheet will be inserted to the left or right of the currently-active sheet. See also the *xl name* command.

xl background <color>

Set the background color. Available colors are listed in the *xl font <G7RMxlfont>* section.

xl border <FC> <FR> <LC> <LR> <option 1> [<option2>[...]]

Set the cell border, where the cell range is given by <first column>, <first row>, <last column>, and <last row>, and options are

off
remove border.

color
border color, chosen from the list of Excel colors.

weight
border weight, chosen from hairline, thin, medium, or thick.

position
border position, chosen from border[default], left, right, top, bottom, horizontal, vertical, diagonalup, or diagonaldown.

linestyle
border linestyle, chosen from continuous, dash, dashdot, dashdotdot, dot, double, or slantdashdot.

xl freeze <[row]||[column]||[cell]||[off]> [[r | c | rc] [rc]]>

Freeze a spreadsheet at a specified location, where first the <[row]||[column]||[cell]> for freezing is given, then the <[upper-row]||[leftmost-column]||[top-left-cell]> is given for the frozen pane, and finally the the top-left cell for the lower-right scrolling pane is specified. Alternatively, freezing may be turned off with the "off" option.

xl graph <row 1><column1>. . . <row N><column N> <direction> <start date><end date> <graph style>

xl graph title ["<title>" [font <style>]]

Create a graph sheet in the current workbook. The graph will appear in a sheet to the left of the active worksheet. Specify the first cell of each series to be graphed. Each series must extend in the same direction, either to the right or down the sheet. Each series should have the same number of observations. The first series will appear on the horizontal axis. Current *G7* settings for title, subtitle, and vertical axis title will be employed.

The command also may be used to recover the title of an existing graph; if no arguments are given, then the graph title subsequently is available using the %xls keyword. If a title is given, then the title will be added to the active chart; the title must be surrounded by quotation marks. If the title is followed by the "font" keyword, then font options may be specified including color, typeface, size, single or double underline, bold, and italic.

Available graph styles include:

Graph Styles	
ColumnClustered	Bubble3DEffect
ColumnStacked	StockHLC
ColumnStacked100	StockOHLC
3DColumnClustered	StockVHLC
3DColumnStacked	StockVOHLC
3DColumnStacked100	CylinderColClustered
BarClustered	CylinderColStacked
BarStacked	CylinderColStacked100
BarStacked100	CylinderBarClustered
3DBarClustered	CylinderBarStacked
3DBarStacked	CylinderBarStacked100
3DBarStacked100	CylinderCol
LineStacked	ConeColClustered
LineStacked100	ConeColStacked
LineMarkers	ConeColStacked100
LineMarkersStacked	ConeBarClustered
LineMarkersStacked100	ConeBarStacked
PieOfPie	ConeBarStacked100
PieExploded	ConeCol
3DPieExploded	PyramidColClustered
BarOfPie	PyramidColStacked
XYScatterSmooth	PyramidColStacked100
XYScatterSmoothNoMarkers	PyramidBarClustered
XYScatterLines	PyramidBarStacked
XYScatterLinesNoMarkers	PyramidBarStacked100
AreaStacked	PyramidCol

continues on next page

Table 1 – continued from previous page

AreaStacked100	3DColumn
3DAreaStacked	Line
3DAreaStacked100	3DLine
DoughnutExploded	3DPie
RadarMarkers	Pie
RadarFilled	XYScatter
Surface	3DArea
SurfaceWireframe	Area
SurfaceTopView	Doughnut
SurfaceTopViewWireframe	Radar
Bubble	

xl cf <FC> <FR> <LC> <LR> <number of conditions>

<type1><operator><condition1>[<operator><condition2>][font <opts>][border <opts>][background <opts>]

[<type2><operator><condition1>[<operator><condition2>][font <opts>][border <opts>][background <opts>]]

[<type3><operator><condition1>[<operator><condition2>][font <opts>][border <opts>][background <opts>]]

Set conditional formatting for the specified range of cells. Up to three conditions may be specified, and so the number of conditions and the number of specification lines must be between 1 and 3.

Each row of conditions must begin with "value"; "formula" will be offered later. Operators must be <=, <, ==, !=, >, or >=. If a second condition is specified and the first operator is < or <=, then the second operator must be > or >= (not between), or if the first operator is > or >=, then the second operator must be < or <= (between); otherwise, any second operator and condition will be ignored. A condition may be a number, string, or cell address. Available format settings include font and background; the "font" or "background" keyword must preceed the format specification, where specifications may be chosen from the relevant list. Font options include color, bold, italic, and single or double underline; border is not implemented yet; background options include color. The number of conditions must match the number of rows.

xl gridlines [<on|off>] [<color>]

Turn on or off the display of gridlines on the selected worksheet. A color may be specified from the list of Excel colors.

xl merge <FC> <FR> <LC> <LR> ["off"]

Merge cells in the rectangle from column <FC> and row <FR> to column <LR> and row <LR>. Separate cells by adding "off".

xl printer <option 1> [<option2>[. . .]]

Set print options, where the options are

orientation
set the page orientation to landscape or portrait.

area
set the print area with “area <fr><fc><lr><lc>”, or turn off the print area with “off”.

print
print the current page.

xl save [<namefile> [filetype]]

An *xl save* command must provide a name for the open workbook, if the workbook has not been named already.

If a file type is specified that is different than the current setting, then the spreadsheet will be saved as the new file type. The default filetype is .XLS. Available file types include

File Types	
AddIn (.xlam)	WorkbookNormal (.xls)
CSV (.csv)	SYLK (.slk)
CSVMac (.csv)	Template (.xltx)
CSVMSDOS (.csv)	TextMac (.txt)
CSVWindows (.csv)	TextMSDOS (.txt)
DBF2 (.dbf)	TextPrinter (.txt)
DBF3 (.dbf)	TextWindows (.txt)
DBF4 (.dbf)	WK1 (.wk1)
DIF (.dif)	WK1ALL (.wk1)
Excel2 (.xls)	WK1FMT (.wk1)
Excel2FarEast (.xls)	WK3 (.wk3)
Excel3 (.xls)	WK4 (.wk4)
Excel4 (.xls)	WK3FM3 (.wk3)
Excel5 (.xls)	WKS (.wks)
Excel7 (.xls)	WQ1 (.wq1)
Excel9795 (.xls)	UnicodeText (.txt)
Excel4Workbook (.xls)	Html (.html)
IntlAddIn (.xla)	XLS (.xls)
IntlMacro (.xlsm)	

If no extension is provided with the filename, then the appropriate extension will be determined according to the file type and will be appended to the file name.

xl subscript <C> <R> “<text>”
xl superscript <C> <R> “<text>”

Both append “<text>” to the current contents of the cell at column c and row <R>, either as subscript or superscript.

xl write <column letter> <row number> “<text>”

xl write <column_letter> <row_number> <down|right|up|left> <series_name> [<start_date> <end_date>]

The first specification writes a string of text to a cell in the worksheet.

The first specification will print floating point values or integers to the spreadsheet when numbers are provided as the text argument. Previously, numbers were written as strings, though printing of strings still can be forced with “’<number>”.

The second specification writes a time series into the Excel worksheet, starting at the given location, going up or down a column, or across a row, for the dates given. If no dates are specified, then the current *tdates* settings will be employed.

xl vecwrite <vector> < v(index) > < c(cols) > < r(rows) > < direction > [< start > [end]]

G7 can print entire vectors of data for multiple years using a single command. The *vecwrite* command must be used with a VAM bank. This command writes text or data to an open Excel worksheet, where:

Vector
The root name of a vector stored in a vam bank. Bank letters are allowed.

Index
The range of vector elements that are to be printed.

Cols
The spreadsheet columns for the first period of data, given as a group of letters or numbers.

Rows
The spreadsheet rows for the first period of data, given as a group of numbers. Either cols or rows must have a single element, and the other must have the same number of elements as contained in index.

direction
Either d(own) or r(ight), starting with the row or column specified with the cols and row entries. NOTE: this is the direction indexed by time, so that if data for a particular series are to be written across a row, then “right” should be specified.

Start
The starting date, where the date is in *G7* format. If dates are not provided, the desired dates are assumed to be those of the current *tdates* setting.

End
The ending date, where the date is in *G7* format.

Examples:

```
xl vecwrite a.output v(1-10) c(B) r(5-14) right 2008 2010
```

xl formula <column> <row> “< formula >”

Insert an Excel formula in the specified cell. ::

```
xl formula A 5 “=sum(A1:A4)”
```

xl setfont <settings>

xl font <column1> <row1> <column2> <row2> <settings>

xl font <column1> <row1> <column2> <row2> <settings>

The *xl setfont* routine sets the font for all subsequent printing. The *xl font* command sets the font for the specified range of cells.

clear
Clear the fonts set by the *setfont* command.

bold
Set text to bold face.

italic
Set text to italic.

underline
See the list of underlining options.

justify|center|right|left
Horizontal alignment.

top|vcenter|bottom|vjustify
Vertical alignment.

color
See the list of available text colors.

type
face See the list of available type faces.

X or sizeX
Set the font size to integer X.

Examples:

```
xl setfont bold italic ~ right top "red" "arial" size14
xl font A 1 D 10 "Courier New" 10 "blue"
xl font A 6 AZ 6 bold format("yyyy")
```

textwrap [off]
Turn text wrapping on or off for the specified cells.

format(<format>|off)
This setting allows specification of general, number, currency, accounting, date, short date, long date, time, percentage, fraction, scientific, text, off, or custom settings. Complex and multiple word settings must be wrapped in “”. The “off” and custom settings must use the “format(<>)” syntax; other settings may be stated simply as “general”, “number”, or otherwise.

separator(“<separator>”|off]
Turn on or off the use of a separator for numeric cells. A custom separator may be specified, or the “(“<separator>”)” argument may be omitted to employ the default separator. The setting will be stored, but it only will be applied to cells if a format specification has been or will be given. Precision still is controlled by the G7 format command.

Underline
Underlining may be specified in several ways.

underline [-]

UnderlineSingle

-

Single underlining.

underline = UnderlineDouble =

Double underlining.

underline _ UnderlineSingleAccounting _

Single accounting underlining.

underline [~] UnderlineDoubleAccounting ~

Double accounting underlining.

underline n UnderlineNone

No underlining.

Available font colors include:

Font Colors	
“None”	“Medium Gray”
“Aqua”	“Mint green”
“Black”	“Navy blue”
“Blue”	“Olive green”
“Cream”	“Purple”
“Dark Gray”	“Red”
“Fuchsia”	“Silver”
“Gray”	“Sky blue”
“Green”	“Teal”
“Lime green”	“White”
“Light Gray”	“Yellow”
“Maroon”	“RGB(<int1>,<int2>,<int3>)”

System fonts may be specified by employing the “TF(<system_font)” function (or “typeface(<system_font>)”) in <settings>. Multiple-word font names must be surrounded by quotation marks. For example, if the Adobe Garamond Pro font is installed, then it may be specified in the *xl font* command as “TF(“Adobe Garamond Pro”)”. Available standard font types include:

Font Types	
“Agency FB”	“Gill Sans Ultra Bold Condensed”
“Agency FB Bold”	“Gloucester MT Extra Condensed”

continues on next page

Table 2 – continued from previous page

“Algerian”	“Goudy Old Style”
“Arial”	“Goudy Old Style Bold”
“Arial Black”	“Goudy Old Style Italic”
“Arial Black Italic”	“Goudy Stout”
“Arial Bold”	“Haettenschweiler”
“Arial Bold Italic”	“Harlow Solid Italic”
“Arial Italic”	“Harrington”
“Arial Narrow”	“High Tower Text”
“Arial Narrow Bold”	“High Tower Text Italic”
“Arial Narrow Bold Italic”	“Impact”
“Arial Narrow Italic”	“Imprint MT Shadow”
“Arial Rounded MT Bold”	“Informal Roman”
“Arial Unicode MS”	“Jokerman”
“Baskerville Old Face”	“Juice ITC”
“Batang”	“Kristen ITC”
“Bauhaus 93”	“Kunstler Script”
“Bell MT”	“Lucida Bright”
“Bell MT Bold”	“Lucida Bright Demibold”
“Bell MT Italic”	“Lucida Bright Demibold Italic”
“Berlin Sans FB”	“Lucida Bright Italic”
“Berlin Sans FB Bold”	“Lucida Calligraphy Italic”
“Berlin Sans FB Demi Bold”	“Lucida Fax Demibold”
“Bernard MT Condensed”	“Lucida Fax Demibold Italic”
“Blackadder ITC”	“Lucida Fax Italic”
“Bodoni MT”	“Lucida Fax Regular”
“Bodoni MT Black”	“Lucida Handwriting Italic”
“Bodoni MT Black Italic”	“Lucida Sans Demibold Italic”
“Bodoni MT Bold”	“Lucida Sans Demibold Roman”
“Bodoni MT Bold Italic”	“Lucida Sans Italic”
“Bodoni MT Condensed”	“Lucida Sans Regular”
“Bodoni MT Condensed Bold”	“Lucida Sans Typewriter Bold”
“Bodoni MT Condensed Bold Italic”	“Lucida Sans Typewriter Bold Oblique”
“Bodoni MT Condensed Italic”	“Lucida Sans Typewriter Oblique”
“Bodoni MT Italic”	“Lucida Sans Typewriter Regular”
“Bodoni MT Poster Compressed”	“Magneto Bold”
“Book Antiqua”	“Maiandra GD”
“Book Antiqua Bold”	“Map Symbols”
“Book Antiqua Bold Italic”	“Matura MT Script Capitals”
“Book Antiqua Italic”	“Mistral”
“Bookman Old Style”	“Modern No. 20”
“Bookman Old Style Bold”	“Monotype Corsiva”
“Bookman Old Style Bold Italic”	“MS Mincho”
“Bookman Old Style Italic”	“MS Outlook”
“Bradley Hand ITC”	“MT Extra”
“Britannic Bold”	“Niagara Engraved”
“Broadway”	“Niagara Solid”
“Brush Script MT Italic”	“OCR A Extended”
“Californian FB”	“Old English Text MT”
“Californian FB Bold”	“Onyx”
“Californian FB Italic”	“Palace Script MT”
“Calisto MT”	“Palatino Linotype”
“Calisto MT Bold”	“Palatino Linotype Bold”

continues on next page

Table 2 – continued from previous page

“Calisto MT Bold Italic”	“Palatino Linotype Bold Italic”
“Calisto MT Italic”	“Palatino Linotype Italic”
“Castellar”	“Papyrus”
“Centaur”	“Parchment”
“Century”	“Perpetua”
“Century Gothic”	“Perpetua Bold”
“Century Gothic Bold”	“Perpetua Bold Italic”
“Century Gothic Bold Italic”	“Perpetua Italic”
“Century Gothic Italic”	“Perpetua Titling MT Bold”
“Century Schoolbook”	“Perpetua Titling MT Light”
“Century Schoolbook Bold”	“Playbill”
“Century Schoolbook Bold Italic”	“PMingLiU”
“Century Schoolbook Italic”	“Poor Richard”
“Chiller”	“Pristina”
“Colonna MT”	“Rage Italic”
“Comic Sans MS”	“Ravie”
“Comic Sans MS Bold”	“Rockwell”
“Cooper Black”	“Rockwell Bold”
“Copperplate Gothic Bold”	“Rockwell Bold Italic”
“Copperplate Gothic Light”	“Rockwell Condensed”
“Courier New”	“Rockwell Condensed Bold”
“Courier New Bold”	“Rockwell Extra Bold”
“Courier New Bold Italic”	“Rockwell Italic”
“Courier New Italic”	“Script MT Bold”
“Curlz MT”	“Showcard Gothic”
“Edwardian Script ITC”	“SimSun”
“Elephant”	“Snap ITC”
“Elephant Italic”	“Stencil”
“Engravers MT”	“Symbol”
“Eras Bold ITC”	“Tahoma”
“Eras Demi ITC”	“Tahoma Bold”
“Eras Light ITC”	“Tempus Sans ITC”
“Eras Medium ITC”	“Times”
“Felix Titling”	“Times New Roman”
“Footlight MT Light”	“Times New Roman Bold”
“Forte”	“Times New Roman Bold Italic”
“Franklin Gothic Book”	“Times New Roman Italic”
“Franklin Gothic Book Italic”	“Trebuchet MS”
“Franklin Gothic Demi”	“Trebuchet MS Bold”
“Franklin Gothic Demi Cond”	“Trebuchet MS Bold Italic”
“Franklin Gothic Demi Italic”	“Trebuchet MS Italic”
“Franklin Gothic Heavy”	“Tw Cen MT”
“Franklin Gothic Heavy Italic”	“Tw Cen MT Bold”
“Franklin Gothic Medium”	“Tw Cen MT Bold Italic”
“Franklin Gothic Medium Cond”	“Tw Cen MT Condensed”
“Franklin Gothic Medium Italic”	“Tw Cen MT Condensed Bold”
“Freestyle Script”	“Tw Cen MT Condensed Extra Bold”
“French Script MT”	“Tw Cen MT Italic”
“Garamond”	“Verdana”
“Garamond Bold”	“Verdana Bold”
“Garamond Italic”	“Verdana Bold Italic”
“Gigi”	“Verdana Italic”

continues on next page

Table 2 – continued from previous page

“Gill Sans MT”	“Viner Hand ITC”
“Gill Sans MT Bold”	“Vivaldi Italic”
“Gill Sans MT Bold Italic”	“Vladimir Script”
“Gill Sans MT Condensed”	“Wide Latin”
“Gill Sans MT Ext Condensed Bold”	“Wingdings”
“Gill Sans MT Italic”	“Wingdings 2”
“Gill Sans Ultra Bold”	“Wingdings 3”

xl read <column letter> <row number> “”

xl read <column letter> <row number> <direction > <series name> <starting date> <ending date>

xl read <col1> <row1> date [<col2> <row2>]

The first specification will read the text in the cell at the specified column and row. The string of text will be printed to the screen. It also may be recovered by using the *%xls* keyword.

The second specification reads a time series from the Excel worksheet, starting at the given location and proceeding in the direction specified, for the dates given. <direction> may be right, down, left, or up.

The third specification introduces the ability to read a range of dates and then to recover the date and its components with keywords and to read a corresponding range of data with possibly non-contiguous dates. The set of dates are stored in Excel date format, where a single date may be recovered with the %xldate keyword. The same may be recovered in G7 date format with %xlgdate, or the date components may be recovered as %xlyear, %xlquarter, %xlmonth, and %xlday. If a range of cells is specified, then the range must contain either a single row or column. Before reading a subsequent data set using this date range, the frequency should be specified with the *xl setfrequency* command to prevent possible ambiguity, particularly with quarterly data. Finally, data may be read with a second *xl read* command that employs the stored date range.

Example:

```
xl read A 1 date H 1
xl setfrequency 4
xl read A 2 right gdp "xldates"
```

xl vecread <c(columns)> <r(rows)> <direction> <vector> <v(elements)> [<start date> [<end date>]]

G7 can read entire vectors of data for multiple years using a single command. The *vecread* command must be used with a VAM bank. This command reads text or data from an open Excel worksheet, where:

Vector
The root name of a vector stored in a vam bank. Bank letters are allowed.

Elements
The range of vector elements that are to be printed.

Cols
The spreadsheet columns for the first period of data, given as a group of letters or numbers.

Rows
The spreadsheet rows for the first period of data, given as a group of numbers. Either cols or rows must have a single element, and the other must have the same number of elements as contained in index.

direction
Either d(own) or r(ight), starting with the row or column specified with the cols and row entries. NOTE: this is the direction indexed by time, so that if data for a particular series are to be written across a row, then "right" should be specified.

Start
The starting date, where the date is in *G7* format. If dates are not provided, the desired dates are assumed to be those of the current *tdates* setting.

End
The ending date, where the date is in *G7* format.

Examples:

```
xl vecread c(B) r(7 9 11-18 20 21 23-29 31-43 45 47-50 52-58 60-64 66 68-77 79-153 155-157 159-236␣
↪238-268) right cr v(1-247) 2000 2008
```

xl setfrequency <frequency>

The *xl setfrequency* command clarifies the intended frequency of the dates read with the *xl read command*. Note potential ambiguity of quarterly frequencies in particular when dates are specified in the Excel date format; the dates probably were recorded as the first day of the first month of the quarter. The *frequency* argument should be '1' for annual, '2' for semiannual, '4' for quarterly, or '12' for monthly data frequencies.

xl matread c(column index grp) r(row index grp) <matname> c(column index grp) r(row index grp) <date>

The *xl matread* command reads a data matrix from a spreadsheet and records it as a matrix in a Vam file. Use the first "c" and "r" expressions to indicate the numbers of the columns and rows in which the matrix is contained in the worksheet. Then give the name of the matrix where you want to place the data, and then the column and row group expressions where you want to put the data.

xl missing [symbol]

This sets missing value symbols for the spreadsheet. When *G7* is reading the spreadsheet file, a "missing value" entry is recorded in the *G7* data bank for any spreadsheet cell containing this symbol. The symbol may be a word, number, or a string, where strings must be specified in quotes in the *xl missing* command. Up to 10 missing value codes may be stored, but each must be entered separately. If the command is given without arguments, then previous entries are reported. For example, the command allows *G7* to recognize 0.0, NA, and _N/A_ as missing value codes when they are read from a spreadsheet file. See also the *xl replace* command.

xl print missing "<value>"

This command provides a character or string to represent a missing value when *G7* writes data to a spreadsheet. ::

xl print missing “N/A”

xl clear missing

This command clears missing value codes specified in *xl missing*. It also resets the replacement value to the default setting, where replacement values are specified with the *xl replace* command.

xl replace <value>

This command sets a replacement value for missing values in the spreadsheet file. If *G7* reads a cell that matches an entry set with *xl missing*, then the value is replaced by value. By default, this replacement value is the *G7* missing value code (-0.0000001). <value> must be a number. See also the *xl missing* command.

xl visible

xl invisible

This command makes visible (invisible) the Excel program, provided that Excel is running. Making Excel visible (invisible) may decrease (increase) the execution speed of your script. By default, the Excel window is not visible when *G7* launches the Excel server and opens a spreadsheet file.

xl name worksheet <sheetname>

This command names the worksheet that currently is open.

xl column width <column> <width>

This command sets the column <column> in the selected worksheet to width <width>.

xl column delete <column>

This command deletes the column <column> in the selected worksheet.

xl row height <row> <height>

The *xl row* command controls the specified row. Current controls include specification of the row height, given as an scalar.

xl row delete <row>

This command deletes the row <row> in the selected worksheet.

xl close

This command closes the Excel file.

xl exit

This command closes an open workbook and disconnects *G7* from the Excel server that is running in the background. If the Excel server was started by *G7*, then it will be stopped. Otherwise, the Excel server may continue running in the background; it can be closed by opening the Task Manager, selecting "Excel," and terminating the process.

Deprecated Features

xl mkseries <frequency> [text] <column> <row> [text]

Read a text string from a worksheet cell, and optionally attach additional text to the string. A new series with the created name is added to the workspace. If a following *xl read* command is given, and if the series name is omitted in the read command, then the series created by the previous *xl mkseries* command will be assumed.

3.23 *G7* Commands: Y

yearformat(yf) <format>

G7 solved the "year 2000" problem long ago, by representing the year 2000 as "100", 2010 as "110" and so on. However, more recent versions of *G7* are even smarter, in that they can take either a 2-digit or 4-digit date, and convert it internally into a "G Date".

You can provide control over how you would like these dates to print out in *print*, *matpr* or *graph* commands by using the *yearformat* (*yf*) command.

The possible values for <format> are:

4

All dates 4-digit, i.e. 1977, 2000, 2005

3

2 or 3 digit dates, 77, 100, 105

2

All dates 2-digit, i.e. 77, 00, 05

Related Topics: Dates in *G7*, *save*

3.24 *G7* Commands: Z

zap [<baseyr>] [<starting period>] [<nobservations>]

The *zap* command is used to delete the old workspace bank and create a new bank with the name specified in G.CFG. The three arguments are optional. If they are given, then they override the corresponding values in G.CFG.

If <baseyr> is given, it will override the line labeled "Default base year of workspace file. If <starting period> is given, it will override the line labeled "First month covered", or "First period covered". If <nobservations> is given, it will override the line labeled "Default maximum number of observations per series in workspace bank".

Related Topics: G.CFG file, Workspace Bank, *maxobs*, *wsbank*, *wsinfo*

zip

After the *zip* command has been given, the program will not pause after regression commands. It particularly is useful for rapid re-estimation of an entire model when data have been updated. The command to turn off *zip* is "zip off".

The *zip* command is used in production mode when you are sure that your add files of regressions or other procedures is correct and you want to run a large job in batch mode with no pauses or interruptions. For example, when creating many graphics files, you may turn on *autoprint* and use *zip*. It also is used when creating large equation parameter files using the *punch* and *ipch* command for *Interdyme* models.

Related Topics: *autoprint*, *catch*, *hesitate*, *ipch*, *punch*, *recursive*, *save*

CHAPTER

FOUR

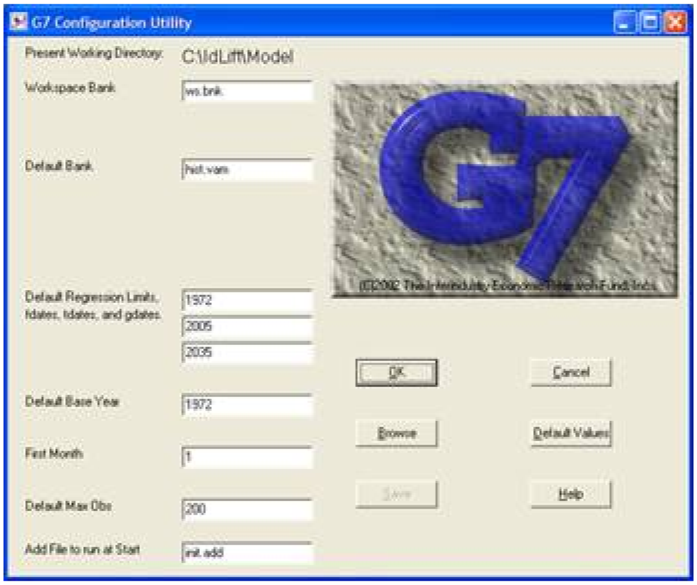
G7 TUTORIAL

4.1 *G7* Tutorial

The *G7* program is a powerful and free program developed by Inforum at the University of Maryland. It can be used to build or access databanks containing millions of time series. Data can be viewed in a grid or a graph or saved to a spreadsheet for further manipulation. Tables of historical data or model forecasts easily can be created from the data in a *G* databank. Many databanks that can be used with *G7* also are available for free on the EconData (inforumweb.inforumecon.com/econdata/econdatacontents.html) portion of the Inforum web site. Included there are the National Income and Product Accounts (NIPA), Annual Survey of Manufacturing, Annual Retail Trade Survey, and many more.

To obtain *G7* directly from the Inforum web site, go to inforumweb.inforumecon.com/software/g7.html. Follow the "Get *G7* Here" link and save the file g7exe.exe to a temporary folder. Next double click on that file icon in Windows Explorer. It will ask you into what folder you would like to extract the *G7* installation files. Specify C:Temp or something similar. Finally, go to the C:Temp directory in Explorer and double-click the G7.msi icon. Follow the steps of the installation program.

Navigate to the EconData page at inforumweb.inforumecon.com/econdata/econdatacontents.html and retrieve the quarterly NIPA bank. The filename is NIPAQ.ZIP. We recommend that you create a new folder named gbanks, and then create a subfolder for this bank called nipaq. Extract the contents of the NIPAQ.ZIP file into C:GBANKSNIPAQ. For the first several activities in *G7*, this folder will be the home directory.



G7 Configuration Utility

Present Working Directory: C:\IdLift\Model

Workspace Bank: ws.bnk

Default Bank: hist.vam

Default Regression Limits, fdates, tdates, and gdates: 1972 / 2005 / 2035

Default Base Year: 1972

First Month: 1

Default Max Obs: 200

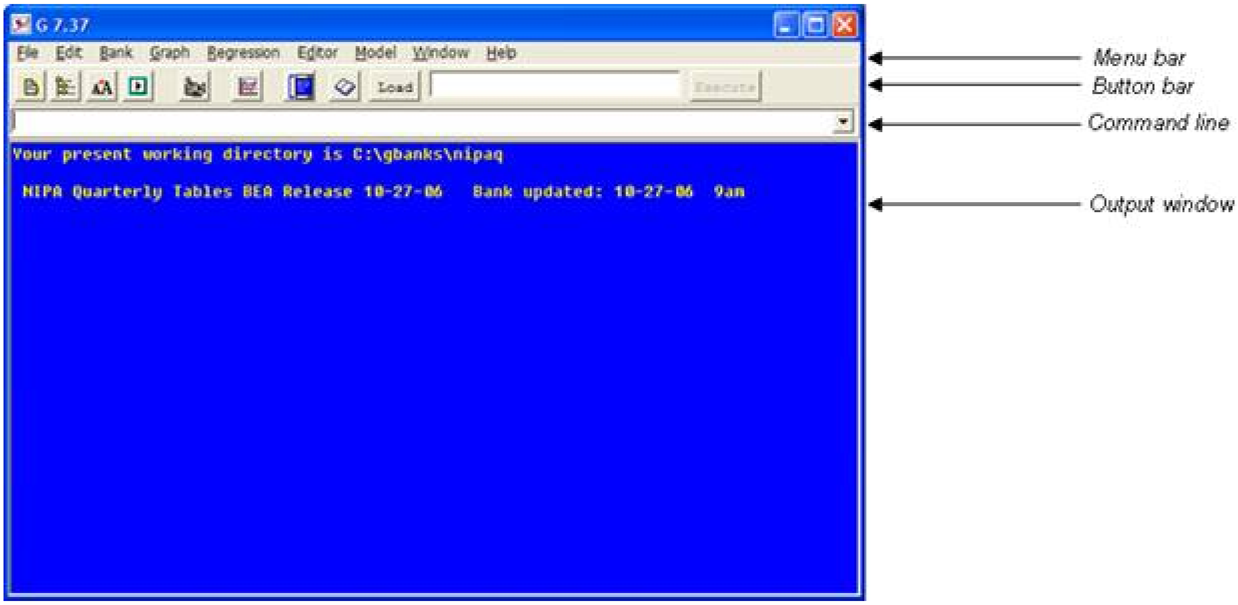
Add File to run at Start: init.add

Fig. 1: **G7 Configuration Dialog**

Now start *G7* by double-clicking the shortcut you just installed on the desktop. You first will see a dialog window similar to the one above. Click the 'Browse' button in the middle of the dialog, or if you prefer keyboard shortcuts, use Alt-B. Navigate in the explorer window to C:GBANKSNIPAQ, and double click on the file G.CFG. This establishes C:GBANKSNIPAQ as the current directory for *G7* and also reads some configuration information specific to the nipaq databank, such as the default databank (nipa), the default frequency (quarterly), and the default range of dates for many commands. Now click the 'OK' button, or just hit [Enter]. The *G7* Main Window will appear.

4.2 The *G7* Main Window and Some Basic *G7* Commands

The *G7* Main Window is shown below, and consists of a menu bar, a button bar, a command box for typing commands, and an output window. At any time, you may click Help, or the little book button on the button bar, and read an online tutorial on using *G7*.



```
Your present working directory is C:\gbanks\nipaq

 NIPA Quarterly Tables BEA Release 10-27-06   Bank updated: 10-27-06   9am
```

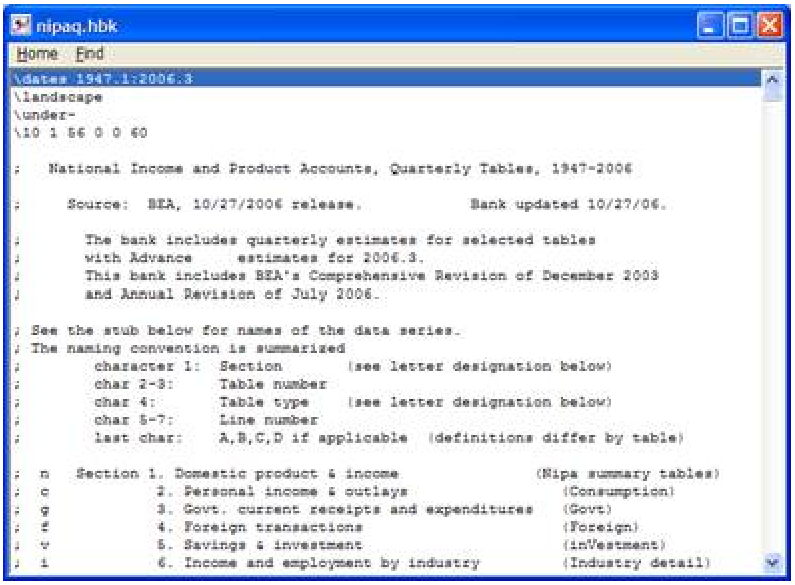
Fig. 2: **The G7 Main Window**

Many of the features of *G7* can be used either from the menu or from the command box. Where this is the case, we will show both methods. However, the real power of *G7* comes from its command language which can be used to build programs or scripts, called add files, that automate data manipulation and regression.

To use the command box, click the pointer in the white box. The first command we will use is called *look*, which shows the contents of the currently assigned G bank.

Type "look" in the command box, and then hit the [Enter] key. If you want to use the menu, then choose the menu option Bank | Look. A small dialog window will open with a list of the currently-assigned banks. Only one bank initially is assigned, which is nipaq. Click on nipaq, and click OK. This is equivalent to giving the *look* command from the command line.

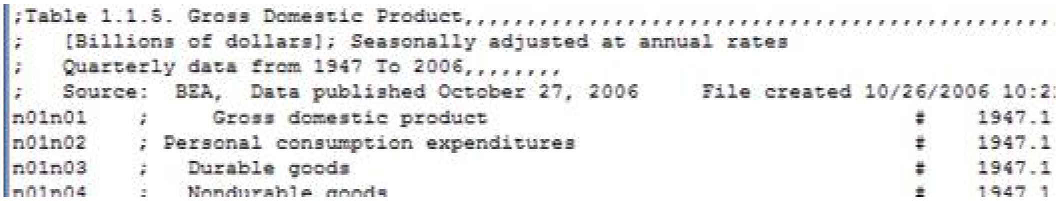
A window like the one shown below will appear. This *look* window provides a convenient way to navigate through the databank and look at graphs and printouts of the NIPA quarterly time series. The window is showing the contents of a *G7* stub file. The stub file simply is a text file. Comment lines in this file begin with a semicolon (';'). Lines with time series have the name of the series to the left of the semicolon and descriptive text to the right of the semicolon. You can scroll through the stub file in the *look* window by using the [PgUp] and [PgDn] keys, the arrow keys, or the scroll bars. You can also use the Find command to search for text in the stub file.



```
\dates 1947.1:2006.3
\landscape
\under-
\10 1 56 0 0 60

;   National Income and Product Accounts, Quarterly Tables, 1947-2006

;     Source:  BEA, 10/27/2006 release.            Bank updated 10/27/06.

;       The bank includes quarterly estimates for selected tables
;       with Advance     estimates for 2006.3.
;       This bank includes BEA's Comprehensive Revision of December 2003
;       and Annual Revision of July 2006.

; See the stub below for names of the data series.
; The naming convention is summarized
;        character 1:  Section       (see letter designation below)
;        char 2-3:     Table number
;        char 4:       Table type    (see letter designation below)
;        char 5-7:     Line number
;        last char:    A,B,C,D if applicable  (definitions differ by table)

;  n   Section 1. Domestic product & income                (Nipa summary tables)
;  c           2. Personal income & outlays                  (Consumption)
;  g           3. Govt. current receipts and expenditures    (Govt)
;  f           4. Foreign transactions                       (Foreign)
;  v           5. Savings & investment                       (inVestment)
;  i           6. Income and employment by industry          (Industry detail)
```

Fig. 3: **The Look Window**

Use the Find command to find the series name “n01n01”. This is the *G7* series name for Gross Domestic Product in the NIPA bank. The data are found in NIPA table 1.1.5, as shown below.



```
;Table 1.1.5. Gross Domestic Product,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
;    [Billions of dollars]; Seasonally adjusted at annual rates
;    Quarterly data from 1947 To 2006,,,,,,,,
;    Source:  BEA,  Data published October 27, 2006     File created 10/26/2006 10:2
n01n01    ;      Gross domestic product                              #    1947.1
n01n02    ; Personal consumption expenditures                        #    1947.1
n01n03    ;    Durable goods                                         #    1947.1
n01n04    ;    Nondurable goods                                      #    1947.1
```

Fig. 4: **A NIPA Table**

Now double-click on the line for Gross Domestic Product. In the *G7* output window you will see a printout of the values of n01n01 (GDP) for the values of the current dates for printing. You can scroll back through the *G7* output window using the scroll bar. If needed, text in that window can be selected and copied to the clipboard for pasting into other applications. A graph window (shown below) also will appear, showing a graph of the series for the current graph date interval.

You can resize the graph as you would any normal window. You can print the graph, in either landscape (full page) or portrait (half-page) mode, by picking Graph | Print. You can save the graph (in Windows metafile format, or *.WMF) so that you can include it in another document by selecting the menu item Graph | Save and then giving a file name.

Now, click in the command box again. Type the following command in the box, and hit the [Enter key]:

```
ty n01n01
```

You should see the same display in the output window that you got from double-clicking on the series line in the *look* window.

Now, graph the series manually by typing:

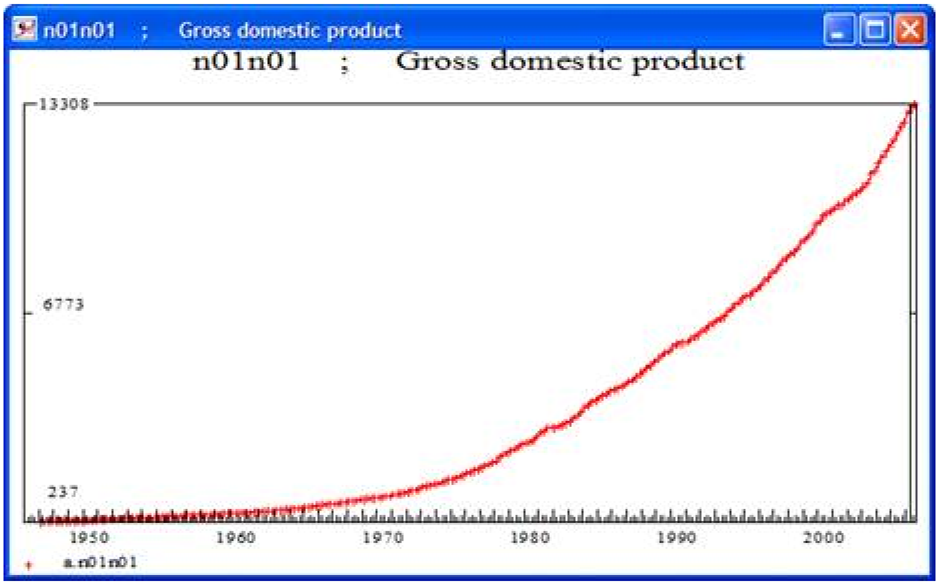

n01n01 ; Gross domestic product
n01n01 ; Gross domestic product
13308
6773
237
1950
1960
1970
1980
1990
2000
a.n01n01

Fig. 5: **A Graph of GDP**

```
gr n01n01
```

You may see in the *look* window (if it still is open) that the series n01n02 is identified as Personal Consumption Expenditures. We will make a new graph of GDP and Personal Consumption, this time giving our own title. Type the lines below in the command box, each time hitting [Enter] at the end of the line:

```
ti GDP and Personal Consumption
gr n01n01 n01n02
```

You now should see the line for consumption appear in blue below the line for GDP. The *title* command (abbreviated *ti*) is for supplying a title for the graph. This graph is very busy, showing quarterly data for a fairly long period. Perhaps you would like to graph the series for a shorter interval. To graph the same series from the first quarter of 1990 to the third quarter of 2011, give the command:

```
gr n01n01 n01n02 1990.1 2011.3
```

This now will be the default starting and ending date for other graphs we make in *G7*. We can change the dates of the *ty* command in the same way:

```
ty n01n02 1995.1 2011.3
```

As with the graph dates, these dates will be remembered for the next *ty* command. In case you were wondering if you could set these dates up front, you can. Set the graph dates using the *gdates* command, and specify the type dates using the *tdates* command. For example, the following lines would set the same dates we had used in the above commands:

```
gdates 1990.1 2011.3
tdates 1995.1 2011.3
```

The series names that Inforum has developed for the NIPA bank are logical and correspond directly to table and line numbers in the NIPA. However, they are hard to remember. You can create a copy of a series in the bank in the *G7* workspace bank by using the *f* (form) command. For example, we can make copies of

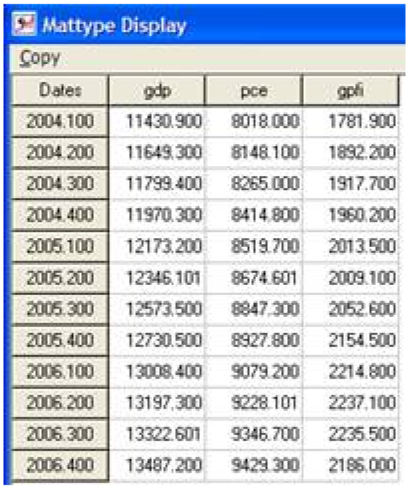
GDP, Personal Consumption, and Gross Private Fixed Investment in the workspace bank, and then graph them, with the following commands:

```
f gdp = n01n01
f pce = n01n02
f gpfi = n01n06
ti GDP, PCE and GPFI
gr gdp pce gpfi
```

Another way of viewing the data is in grid format with the *gridty* (grid type) command. Give the following command and you will see a small spreadsheet window appear:

```
gridty gdp pce gpfi
```

This grid will display data for the current *tdates* setting.



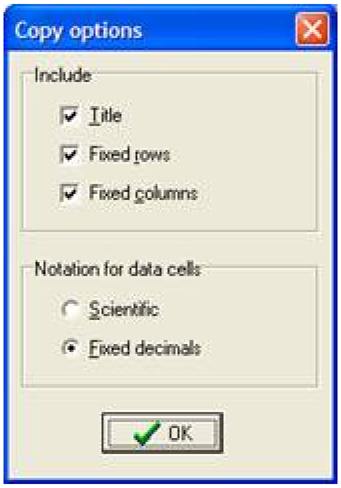
Dates	gdp	pce	gpfi
2004.100	11430.900	8018.000	1781.900
2004.200	11649.300	8148.100	1892.200
2004.300	11799.400	8265.000	1917.700
2004.400	11970.300	8414.800	1960.200
2005.100	12173.200	8519.700	2013.500
2005.200	12346.101	8674.601	2009.100
2005.300	12573.500	8847.300	2052.600
2005.400	12730.500	8927.800	2154.500
2006.100	13008.400	9079.200	2214.800
2006.200	13197.300	9228.101	2237.100
2006.300	13322.601	9346.700	2235.500
2006.400	13487.200	9429.300	2186.000

Fig. 6: **The GridType Window**

If you would like to copy data from the grid to an Excel spreadsheet or to another application, scroll to the bottom right of the grid window and click in the lower, rightmost cell. Then click the Copy menu item at the top of the window. The following dialog will appear:

The “Title”, “Fixed rows”, and “Fixed columns” check boxes allow you to specify if you would like the row and column headers (you probably do) and the title of the bank in your clipboard copy. The “Notation for data cells” choices allows you to copy the data either in scientific notation or with a fixed number of decimals. Now, open an Excel or OpenOffice Calc window and choose Edit | Paste, or Ctrl-V. The data from the *G7* grid window now will be in your spreadsheet.

These are some introductory commands that enable you to browse the contents of a databank, print and graph timeseries data, and display and copy data in a spreadsheet format. We will explore some more *G7* commands in a minute. First we will discuss the *G7* command box in more depth, together with the button bar that appears just above the command box.



Copy options

Include

☑ Title

☑ Fixed rows

☑ Fixed columns

Notation for data cells

○ Scientific

● Fixed decimals

OK

Fig. 7: **GridType Copy Options**

4.3 The *G7* Command Box and Button Bar

The command box remembers commands that you have typed previously. You can scroll through a list of previous commands, select one, and execute it by clicking the ▼ at the right and hitting [Enter]. You also can cycle through previous commands with the up or down arrow cursor keys. In addition, text can be copied and pasted into the command box from other applications (such as this document), or from text in the blue output window; to paste text, right-click on the command box and select Paste from the pop-up menu.

By typing just the first few letters of a previously given command, *G7* will try to guess what you want, and autocomplete the command. If you don't accept *G7's* suggestion, just keep typing the command you really want. If the new command is shorter than the one *G7* suggests, type a few spaces after it.

Many commands also can be executed by using the menus or the buttons just above the command line. The figure below shows the available buttons.

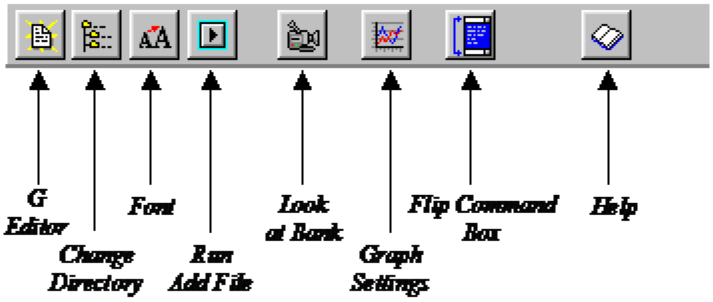


Fig. 8: **Buttons Available on the G7 Console**

For example, click the button that looks like a video camera. This is equivalent to choosing the Bank | Look menu item. *G7* has its own integrated editor, which can be used as a general text editor or for running add files within *G7*. This is the function of the button on the far left. If you have a databank or model in a different directory than the current one, you can change to that directory with the Change Directory button. Use the Font button to change the current display Font in the *G7* output window. "Run Add File" will be discussed below, but this will open an explorer window showing possible *G7* add file scripts that can be run in the current directory. "Graph Settings" allows you to change colors, line widths and styles, and other features of the *G7* graphs. "Flip Command Box" lets you choose whether to have the command box at the top or bottom of the screen. Finally, the "Help" button opens the *G7* command reference, which explains many features of *G7* beyond the scope of this document.

4.4 More on the *f* Command

One of the strengths of *G7* is its capacity to create new data series as transformations of already-defined series. You create a new series in *G7* by using the "f" command that was introduced above. The "f" command also can be used to create new timeseries where new series are the on the left side of an equal sign and functions of previously-defined variables are on the right. For example, to form the ratio of government consumption and investment to nominal GDP, you would:

- Decide on a name for the new variable. We'll call this one "govshare".
- Find the names for government consumption and investment and GDP, and create aliases in the workspace bank if you like. You can get these names by using *look*. Copy n01n01 as gdp and n01n20 as govt.
- Type the following equation:

```
f gdp = n01n01
f govt = n01n20
f govshare = govt/gdp
```

Now that govshare has been calculated, look at the results. Type:

```
ty govshare
```

To graph govshare, simply type:

```
ti Government share of GDP
gr govshare
```

To create a vertical axis title for your graph, use:

```
vaxti Billions of Dollars
```

4.5 Running *G7* Commands in Batch (Add files)

A powerful feature of *G7* is the ability to do a long sequence of commands in batch mode, much like a program. The commands are stored in text files we call add files. *G7* add files typically have certain file extensions to indicate their purpose. The ".ADD" extension is the most common, and ti indicates a file containing a sequence of *G7* commands for data manipulation or preparation. The ".SH" extension is a show file for presenting a visual show of data for analysis. Finally, the ".REG" extension commonly is used for regression files. However, you are free to name the files with any extension you like.

The nipaq bank comes with a show file called NIPAQ.SH. You can run it in *G7* using the *add* command.

```
add nipaq.sh
```

Alternative methods for executing add files include:

- Pick the menu item File | Execute, and then choose the file NIPAQ.SH.

- Click on the “Run Add File” button ▶, and then pick the file NIPAQ.SH.

This NIPAQ.SH file demos the capabilities of *G7* and also plots some of the series in the nipaq bank.

If an add file has many graphs, after each graph in the add file the add file control panel will appear. This panel is shown below.

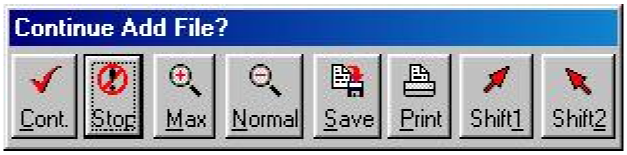


Fig. 9: **Add File Control Panel**

You continue to execute the add file by clicking the Cont. button, and you stop the add file clicking the Stop button. Here are explanations for the other buttons.

Max
maximizes the graph to full screen.

Normal
puts it in its place in the bottom right hand corner.

Save
puts the graph into a file that can be included in other documents.

Print
sends a copy of the graph to your printer.

Shift1
Shift2
buttons allow you to view more than one graph at a time. There are two positions above the default graph position that the graph can be copied to if you click Shift1 or Shift2.

Zip
continue to process the file without stoping to display each graph.

The default button on the add file control panel always is the last button clicked, and the default button can be clicked by pressing the space bar or the [Enter] key. Once you have clicked Cont. the first time, you can move through the entire add file by tapping the space bar.

In most cases, an add file will direct *G7* to act as if you had typed the commands in line by line in the command box, but there are a few multi-line commands that must be executed from within an add file. See the *G7* Reference Manual and the *G7* Help file for more details.

4.6 Using the *G7* Editor

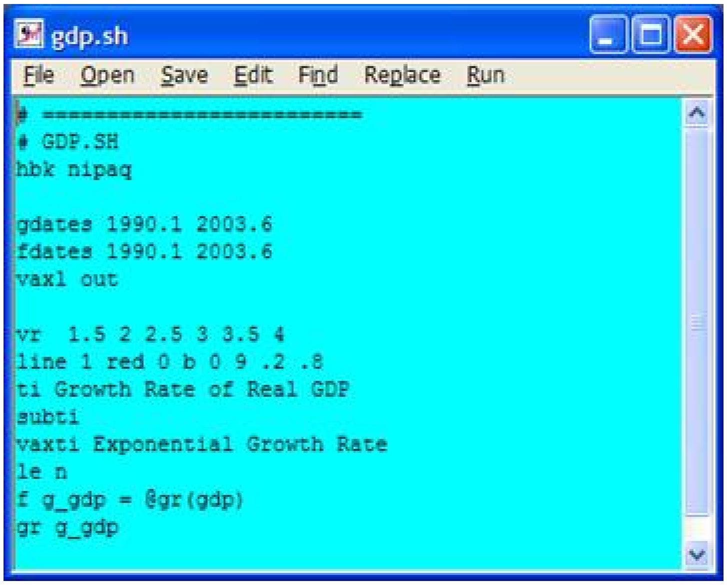
Add files simply are text files. You can look at one, change it, or create a new one within *G7* by using the *edit* command, which may be abbreviated as *ed*. For example, if you had a show file called GDP.SH that showed various components of gdp, you could edit it with:

```
ed gdp.sh
```

in the command box. This will start the *G7* editor, opening a window at the upper right of the screen as shown in the figure below. The *G7* editor is a complete text editor, allowing you to examine and modify any text file, create new text files, and save your results.

You can use this editor to build a *G7* add file by typing in lines of commands. To save your work, click Save on the menu bar and give the file a name. You then can execute the commands in the file in *G7*, without closing the editor, by clicking on Run on the editor's menu bar or by hitting the F9 function key while the cursor is in the editor.

Clicking the 🗎 button on the button bar in the main *G7* window will start the editor, just as if you typed "ed filename" in the command box.



```
# ========================
# GDP.SH
hbk nipaq

gdates 1990.1 2003.6
fdates 1990.1 2003.6
vaxl out

vr  1.5 2 2.5 3 3.5 4
line 1 red 0 b 0 9 .2 .8
ti Growth Rate of Real GDP
subti
vaxti Exponential Growth Rate
le n
f g_gdp = @gr(gdp)
gr g_gdp
```

Fig. 10: **G7 Editor, with GDP.SH File**

4.7 Using *Compare* to Make Tables

Compare is a program that works with *G7* databanks to create tables for printing, or for creating nicely labeled spreadsheets, or producing tables in other formats. You can create your own table definition files (called stub files) to develop a tables and reports of your own design.

In a typical *Compare* table, the values in each column coincide with a given dates or ranges of dates. Rows of the table display data for a given variable or expression involving matrices, vectors, or macrovariables, with an optional title. *Compare* reads a stub file to learn the desired structure and format for the table, which the user defines. *Compare* reads the data series from one or more databanks.

A stub file consists of a list of commands or variables, one per line. The next example is of a small table stub file, which will print a summary of federal receipts, spending, and the surplus. It show some of the rules for creating a valid line.

- Comments follow a hash sign (#) and may be included anywhere in the file.
- Most *Compare* commands begin with a backslash (\), which is a required part of the command. Commands are available for table formatting, doing data transformations, and other tasks.
- To include a variable in the table, type the variable name or expression, followed by a semicolon, followed by the title or label for that series.
- A \dates command indicates the set of periods or growth rates that will define the columns of the table. The \dates command in the example tells the program to print quarterly data columns from

the first quarter of 2005 to the third quarter of 2006, and then to show 2005 and 2006 annual averages (using only the three quarters available for 2006).

- The *&* command prints the current dates as a line in the table.
- A * causes the table to move to the next page.
- The *\announce* tells *Compare* to print the text following the command to the screen to tell users what table is being created.
- The *\ti* command creates the title that is printed on the top of the table.

The other lines in the sample stub file are the names of the variables for which *Compare* will print data, and a label that will appear next to the data in the completed table.

Sample Stub File:

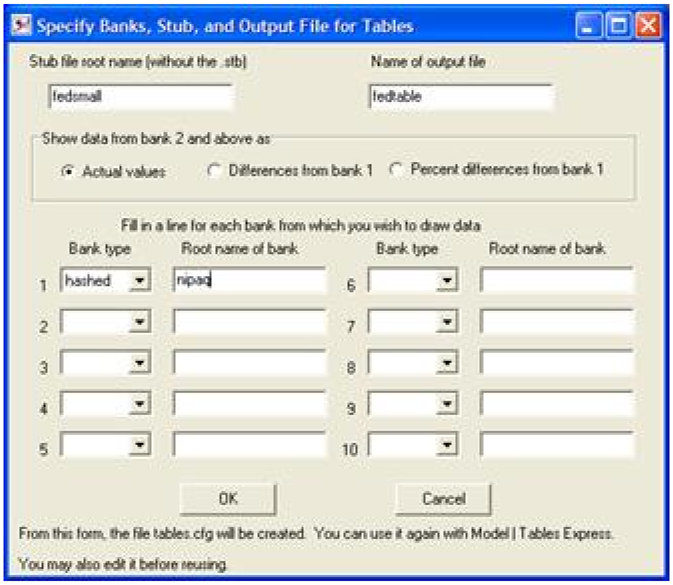
```
# FEDSMALL.STB - This is a simplified version of the
# Federal table, to illustrate features of Compare.
\dates 05.1 05.2 05.3 05.4 06.1 06.2 06.3 05 06
\announce FEDERAL GOVERNMENT RECEIPTS & EXPENDITURES
\ti FEDERAL GOVERNMENT RECEIPTS & EXPENDITURES
*
&
g0201   ; CURRENT RECEIPTS
        ;
g0202   ;  Current tax receipts
g0211   ;  Contributions for social insurance
g0212   ;  Income receipts on assets
g0215   ;  Current transfer receipts
g0218   ;  Current surplus of government enterprises
        ;
g0219   ; CURRENT EXPENDITURES
        ;
g0220   ;  Consumption expenditures
g0221   ;  Current transfer payments
g0228   ;  Interest payments
g0231   ;  Subsidies
g0232   ;  Less: Wage accruals less disbursements
g0233   ;  Net federal government saving
```

Once the stub file is complete and saved, you can run the *Compare* program to create the table. To run *Compare*, select Model | Tables item from the *G7* menu. For the FEDSMALL.STB table above, we fill in the form as follows:

The name of the stub file is FEDSMALL (without the ".STB" extension), and the name of the output table will be "fedtable". The radio buttons in the middle ("Show data from bank 2 and above as") only apply when you are comparing data from two or more banks, so it does not matter which button is pressed, but we select the first "Actual values". The grid at the bottom allows the user to specify up to 10 alternative banks (usually model simulations) to be compared. Each has a drop-down box, "Bank type", and a place to type the root name of the bank (without its file extension). In our case, the bank type is "hashed" and the bank name is "nipaq". Click the OK button, and you should see the following window appear on your screen:

This window is a command window that shows *Compare* running and creating the table "fedtable". Hit a key to continue, and the Federal government receipts and expenditures table will be created. The table can be viewed in the *G7* editor by typing "ed fedtable". The table below was copied and pasted from the editor.

Sample Output Table from *Compare*:



Specify Banks, Stub, and Output File for Tables

Stub file root name (without the .stb): fedsmall

Name of output file: fedtable

Show data from bank 2 and above as

● Actual values ○ Differences from bank 1 ○ Percent differences from bank 1

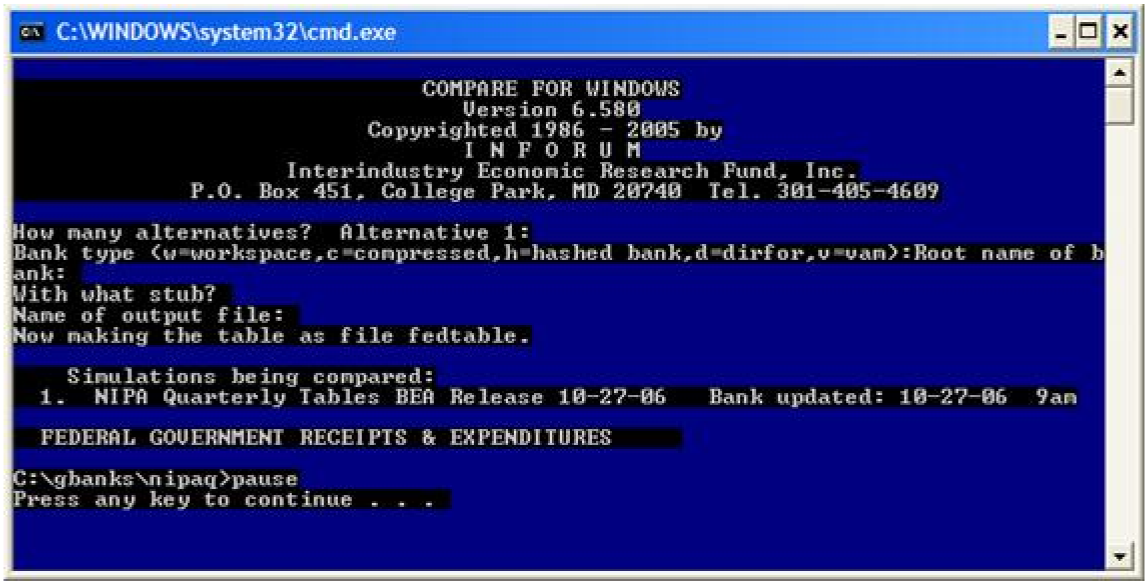
Fill in a line for each bank from which you wish to draw data

	Bank type	Root name of bank		Bank type	Root name of bank
1	hashed	nipaq	6		
2			7		
3			8		
4			9		
5			10		

From this form, the file tables.cfg will be created. You can use it again with Model | Tables Express.

You may also edit it before reusing.

Fig. 11: **The Compare Window**



C:\WINDOWS\system32\cmd.exe

```
                              COMPARE FOR WINDOWS
                                 Version 6.580
                          Copyrighted 1986 - 2005 by
                                 I N F O R U M
                    Interindustry Economic Research Fund, Inc.
             P.O. Box 451, College Park, MD 20740  Tel. 301-405-4609

How many alternatives?  Alternative 1:
Bank type (w=workspace,c=compressed,h=hashed bank,d=dirfor,v=vam):Root name of b
ank:
With what stub?
Name of output file:
Now making the table as file fedtable.

    Simulations being compared:
  1.  NIPA Quarterly Tables BEA Release 10-27-06   Bank updated: 10-27-06  9am

  FEDERAL GOVERNMENT RECEIPTS & EXPENDITURES

C:\gbanks\nipaq>pause
Press any key to continue . . .
```

Fig. 12: **Running Compare**

```
                    FEDERAL GOVERNMENT RECEIPTS & EXPENDITURES

                              2005.1 2005.2 2005.3 2005.4 2006.1 2006.2 2006.3     2005    2006
CURRENT RECEIPTS                        2214.5 2240.3 2182.4 2349.8 2490.9 2554.7   -0.0  2246.8 2522.8

 Current tax receipts            1328.0 1344.3 1364.2 1428.4 1524.9 1570.7   -0.0  1366.2 1547.8
 Contributions for social insur  838.3  846.1  863.2  873.8  911.9  928.1  938.1   855.3  926.0
 Income receipts on assets        22.8   23.8   22.8   22.3   23.3   24.2   25.1    22.9   24.2
 Current transfer receipts        29.1   30.5  -61.7   30.6   32.2   32.8   33.6     7.1   32.9
 Current surplus of government    -3.7   -4.5   -6.0   -5.4   -1.4   -1.1   -1.5    -4.9   -1.3

CURRENT EXPENDITURES                  2502.0 2529.9 2578.5 2613.3 2637.9 2686.2 2732.1  2555.9 2685.4

 Consumption expenditures        758.0  760.8  784.3  771.1  803.6  802.3  808.9   768.5  804.9
 Current transfer payments      1461.2 1461.8 1481.3 1502.4 1522.0 1546.6 1566.7  1476.7 1545.1
 Interest payments               230.9  252.1  255.2  277.1  257.5  285.4  304.9   253.8  282.6
 Subsidies                        51.9   55.2   57.7   62.7   54.7   51.9   51.6    56.9   52.7
 Less: Wage accruals less disbu    0.0    0.0    0.0    0.0    0.0    0.0    0.0     0.0    0.0
 Net federal government saving  -287.6 -289.6 -396.0 -263.6 -147.0 -131.5   -0.0  -309.2 -139.2
```

Printing Tables Created With Compare

Compare creates nicely formatted tables in two basic formats: a text format and a worksheet (spreadsheet) format. The procedures for printing these two types of formats are different and the precise instructions will depend on your individual circumstances. The following steps usually will allow you to print useful tables.

Printing Text Format Tables The procedures for printing text format tables depend on whether or not you can print directly from DOS. If you are on a network, check with you network administrator to find the name of the command that will allow you to print files directly from DOS to your network printer. If your system has such a command, then simply execute that command, referring to the file you created from running *Compare.* For example, if your network DOS print command were called netprint you might type

```
netprint table.out
```

If you cannot print directly to your network printer using DOS, we suggest importing the output from *Compare* into your favorite word processor. If you choose this route, then you also must change the font of the tables to a fixed-width font, like Courier or Line Printer, to preserve the table formatting. You also should choose an 8 or 9-point font.

Further, you also may need to check that the page breaks divide the individual tables in suitable places. If they do not, then changing the top and bottom margins often will allow the tables to print nicely. Some word processors have an auto-fit option that automatically will adjust the size of the tables to fit on the page.

Here are some suggested instructions for Word:

- Choose File | Open from the menu and select your file.
- Choose Edit | Select All (or hit CRTL+A).
- Choose the Line Printer or Courier font in the font selection box.
- To preview the document, choose File | Print Preview.

If you choose to import the text tables into a word processor, be sure that the *\noformat* option is turned on the stub file. That option controls the printing characteristics for DOS printing, but it might interfere with the formatting when a word processor is used.

Printing Spreadsheet Format Tables If you choose the spreadsheet format by putting the command *\xls* or *\wk1* near the top of your stub file, then the output will go into an .XLS or .WK1 file, which can be read by your spreadsheet program. However, you also should preview your spreadsheets to see that they are formatted nicely.

4.8 Where to Go to Learn More

A fairly complete reference book on *G7* can be found at inforumweb.inforumecon.com/papers/inforum/software/GREF.pdf.

A somewhat outdated, though still useful, FAQ on *G7* can be found at inforumweb.inforumecon.com/software/gfaq.html.

The Craft of Economic Modeling, Volumes 1-3, by Clopper Almon are available online in PDF form at inforumweb.inforumecon.com/papers/TheCraft.html.

Please see the G7 page on the Inforum web site for software updates and the latest documentation. In addition to the *Help* files, the *G7 Reference Manual,* and the *G7 Tutorial,* please also see and the demonstration programs offered there.

You can find contact information for software support on the Inforum web site. Please send questions and concerns by email to the Inforum webmaster.

CHAPTER

FIVE

COMPARE MANUAL

5.1 Introduction to *Compare*

The *Compare* program is a general table making program that can be used to make tables from various source databases that are constructed in Inforum data bank formats. As the name suggests, it particularly is suited to comparing results of several data banks, but it also can list the contents of a single data bank or the results of a single run of a model. When being used to show a base case and several alternatives, it can show the alternatives as actual values, or as deviations from the base, or as percentage deviations from the base. The results are written to a file that can be printed, viewed as text, viewed in a spreadsheet program, or read by other software programs. The Inforum databank formats that *Compare* supports include normal G databanks ("workspace banks"), *G* compressed banks, *G* hashed banks, *Interdyme* Vam files, and SLIMFORP Dirfor files. Up to 10 banks of different types can be compared using *Compare*.

Compare contains several features for working with the *Interdyme* system for building input-output models. For example, *Compare* can do matrix listings for an *Interdyme* model, showing intermediate and final demand flows that comprise the total output for each industry. Matrix listings can be done either for buyers or for sellers, and either in flow or coefficient form. The dates for a table or a section of a table should be specified in a *\dates* command. If no *\dates* command is given, dates will be requested from the user at runtime. By using the *\dates* command, the same table can have different dates in its various sections. Also, multi-period growth rates can be displayed.

This document first describes the construction of the stub file, which is the key to specifying the contents of a table and how they will be displayed. Next, the operation of the program is reviewed. Finally, special uses for *Compare* are described. These cases include making a matrix listing for *Interdyme* using *Compare* and how to make tables using Dirfor files.

5.2 Preparing the *Compare* Stub File

To use Compare, one first prepares a "stub" file, which is a text file that provides a description of the series to be printed, the format for printing, titles, dates and other information. Our stub files usually have the extension ".STB", but this is not necessary.

The following is a sample of a basic stub file. All commands in the sample, and many others, will be explained in this section.

```
\dates 2007.0 2008.0 2007.1 2007.2 2007.3 2007.4 2008.1 2008.2 2008.3 2008.4
;
;              THE AMI MODEL
;
&
```

(continues on next page)

(continued from previous page)

```
;
gnp$                   ;Gross National Product
c$                     ;Personal Consumption
pibg$                  ;Personal Income w/o Govt
pidis$                 ;Personal Disposable Income
taxrate*100            ;taxrate*100
gtnis$                 ;Govt Trans, Net Int, Sub
cpc$                   ;Consumption per Capita
pop*1000               ;Population (Millions)
100*(gnp$-gnp$[1])/gnp$[1] ;GNP Growth Rate
```

5.3 Choosing a File Type

Compare can print tables in spreadsheets, word processor files, or as plain text files. Some formatting commands are unique to a particular file type. Enter one of the following commands near the top of a stub file to instruct *Compare* on what type of file to create. If a file type is not specified, *Compare* will create an output file with special codes that are interpreted by the printer.

Printer Format

The default file type is printer format. These files should be sent directly to a printer. As an example, consider the following batch file that will send a file to an HP Laserjet 4200 printer from the windows command line:

```
@echo off
if "%1"=="" goto noarg
echo put %1 port1 > c:\temp\hpscript.bat
echo bye >> c:\temp\hpscript.bat
ftp -A -s:c:\temp\hpscript.bat <insert printer IP address here>
del c:\temp\hpscript.bat
goto end
:noarg
cls
echo :---------------------------------------------:
echo : php4200.bat:  print utility for text files. :
echo : usage:  php4200 [file]                      :
echo :---------------------------------------------:
echo :---------------------------------------------:
:end
```

If no argument is provided, then the batch file will print the required syntax. Otherwise, the filename of the *Compare* output file should be included. For example, if the batch file is named PHP4200.BAT and the *Compare* output file is TABLES.OUT, then the following should be entered into the Windows command prompt:

```
C:> PHP4200.BAT TABLES.OUT
```

For use with other printers, such as Brother models, first change the printer default setting to HP mode (where available). Next, download and install the *prfile32.exe* program from http://www.lerup.com/printfile. Launch the program and change Text File Settings | Text File Action to Send To Printer. Then from the command line, the command

"C:\Program Files\PrintFile\prfile32.exe" <filename>

will bring up a GUI. (You can put this on the path, either in a batch file that will set the path and print the file or as a default.) Select the desired printer and proceed.

See the *printer control commands* for details on the direct control of a printer.

\xls

This option makes all output go to a Microsoft Excel spreadsheet file. The default file extension and file type is .XLS, though other options are available. The Microsoft Excel program must be installed for this feature to work. Supported versions include Excel 2000, 2003, and 2007. Unlike the following *\wk1* feature, printing to multiple worksheets is supported, with new sheets created as with new pages for other formats. See the *\wstitle* command for specifying worksheet names. Special formatting commands for Excel spreadsheets are presented in the "Formatting Commands for Excel Documents" section.

\wk1

This option makes all output go directly to a Lotus .WK1 file. Note that no matter what you give for the name of the output file, it will have the file extension .WK1. You still are limited to the maximum rows and columns (4096 x 256) when using a .WK1 file. To create larger spreadsheets and modern formats, use the *\xls* option.

\prn <y|n>

This option sets printing to a .PRN file, where all text will be in quotes. A .PRN file can be opened by any text editor or Microsoft Excel.

\rtf

This command will instruct *Compare* to create a Rich Text Format (.RTF) file. This file type can be opened with Microsoft Word, OpenOffice Writer, and most other document editing programs. Support for this file type is limited.

\gdata [<ObsPerLine>]

This command is used to generate output as a *G7* "data card" file. For all series names or expressions in the file, output is written in a format usable as input for *G7*. The optional parameter <ObsPerLine> specifies how many observations will be written on each line. Note that the *\field* and *\decs* commands will determine the field width and number of decimal points to be printed. By default, series will be printed to *G7* data cards at 5 observations per line, using the current field width and precision. Also, titles, subtitles, and comment lines (with a ';' and text, but no series) will be printed to the file as *G7*-style comments (i.e., lines starting with a '#'). If you want to shorten the file and include only data cards, use the *\nocomment* option. Note that series that entirely are zeroes will not be printed.

\oneperline

This is a command like *\gdata* that is used to specify that output is not to be a table but rather a data dump. This command prints series to the output file with one data observation per line, with whatever is specified on the title field to the left of the data. Formatting is controlled by current width, precision, and line title width settings. Often this command could be used to print the data with series codes in the line title, with the codes passed as arguments to *\fadd* commands.

5.4 The Most Commonly Used *Compare* Commands

*

Go to the top of a new page and print the titles of the alternatives runs as given in the ".FIX" file. In Excel spreadsheets, a new worksheet will be created.

* <m>

If there is not room for <m> more items on the page, then go to a new page and print the dates across the top.

* <m> <n>

If there is not room on the page for <m> more items plus <n> lines, go to a new page and print the names of the base and alternative runs.

;

Print the line just as it stands.

&

Print the dates across the page above the appropriate columns. Note that this has changed from the '@' used in older versions! This is so that the '@' can be used for function names.

#

Any line beginning with this character as the first non-blank character will be treated as a comment. This allows you to selectively remove portions of a table by commenting them and then include them again later by removing the comment characters.

\dates

Provide the dates for the run. Use either 4-digit (prefered) digit or 2-digit (deprecated) dates; *Compare* can understand either. To specify the number of digits with which dates are printed, see the *\yearformat* command below. Annual dates are printed as integers, or with a '.0'. Quarterly dates are printed with 1 decimal point, and monthly dates with 3 decimal points. Examples: 1992 or 92 means 1992 at an annual rate. 1992.2 means 1992, 2nd quarter. 1993.011 means November 1993.

Growth rates also can be specified with the *\dates* command. To request growth rates, enter two numbers separated with a dash, such as "1992-1995". Growth rates and levels can be mixed in the same table.

There also is a code that can be given in the *\dates* line of a stub file to provide finer control over the column layout of the table. The "s" code is followed by an integer. The integer tells how many columns to skip before printing the next column, if the file format is text or printer mode, or it provides the column width if the *\xls* option is employed. This may be useful in quarterly or monthly printouts, where you want to clearly mark out where the year begins and ends. For example:

```
\dates 1993.1 1993.2 1993.3 1993.4 s4 1994.1 1994.2 1994.3 1994.4 s4 1995.1 1995.2
```

Other uses of the *\dates* command are for calculating sums or averages of variables over a number of periods. If you specify the column heading "1995+1997", then *Compare* will calculate the sum over the periods from 1995 to 1997 and print it in that column. If you specify a column heading such as "1995|1997", then *Compare* will calculate the average over that interval and print it in that column. Finally, you should note that a shorthand version for specifying a continuous interval of dates exists. For example, if you want to make a large table with all series from 1950 to 1997 quarterly, you could specify your dates as:

```
\\dates 1950.1:1997.4
```

\ti <title>

This command can be used to specify a title, which will be centered on each subsequent page. It is positioned as many lines down from the top as is specified by the top margin.

\head <header>

The header is different than a title. The header is put at the top left margin, on the first line of each page. It is directly to the left of the page number. This is useful as the header for an entire set of tables, where each table has its own title given by the *\ti* command. Note: if the <header> is the word "title", the title of the first bank will be used as the header.

\center <text to center>

This command inserts centered text into the table. The centering is done not with respect to the current page width but rather to the "right end" of the table, which is defined as the end of the last column of data.

\line <linechar>

This command inserts a line of <linechar> characters, out to the "right end" of the table. (The right end is the end of the last column of data.) If no character is supplied, the default character will be employed: the dash ('-') character.

\add <stubfilename> [<arg1>] [<arg2>] ...

This command allows you to include the contents of another stub file into the current stub file. This is convenient when you want to make a large table which is an agglomeration of other tables. For example, with a large model such as LIFT, you may keep separate special-interest tables around, but then you may want to add them together for one large table. Note that command line arguments work just as in *G7,* where arguments are replaced by "%1" for the first argument, "%2" for the second, etc.

\fadd <stubfile> <argfile>

This command is also patterned after the corresponding *G7* command. The <stubfile> is a stub file with arguments that could be used with the *\add* command. The <argfile> is a file containing a list of arguments. The <stubfile> is called once for each line of the <argfile>, just as in *G7.*

\mode

This command allows you to change the mode of comparison in selected areas of the table. "\mode a" means that data are to be compared as levels ("actuals"). "\mode d" means that data are to be compared as first differences. "\mode p" means that data are to be compared in terms of percentage differences from the base. Note that this same information is requested from the user in the interactive running of the *Compare* program. The *\mode* command in the stub file overrides this default mode of comparison.

\printbase <true | false>

Set *\printbase* to "true" (or "1" or "on" or "yes") to print the baseline levels. Set *\printbase* to "false" (or "0" or "off" or "no") to print only deviations from baseline levels. Deviations may be presented in differences or percentage differences, and this is controlled with the *\mode* command.

\gt

\growthtype

This command only is relevant if you have multi-period growth rates, such as 2005-2008. The *\gt* command can take three possible arguments. "\gt e" specifies exponential growth, "\gt c" specifies a compound growth rate, and "\gt p" specifies percentage growth. The relevant formulas used are displayed in the table below:

Growth Type	Formula
e	$g = \frac{100}{T} \times \log\left(\frac{x_T}{x_0}\right)$
p	$g = \frac{100}{T} \times \left(\frac{x_T - x_0}{x_0}\right)$
c	$g = 100 \times \left(exp\left(\frac{\log(\frac{x_T}{x_0})}{T}\right) - 1\right)$

where:

g
: The growth rate

x_T
: The ending period data point

x_0
: The starting period data point

T
: The number of periods

exp()
: The exponential function

log()
: The natural logarithm function

Note that the default growth rate type is exponential ('e'), and this will be applied if no alternative is specified.

\g

Show the growth rates in percent per year for the variables named on the following lines. Annual dates with a quarterly file will give growth of one year over the previous year. Quarterly dates with quarterly data will give quarter to quarter growth and this will be at an annual rate if the $\backslash ar$ command is in effect. Be careful not to use this command if you already have growth rates (such as "2002-2005") specified in your $\backslash dates$ command.

\n

Cancels a previous $\backslash g$ command.

\div <number>

This divides every table entry in levels following this command by the specified factor. This is helpful if you would like to convert an entire table from millions to billions, for example. To turn off the effect of this command, merely issue the command "\div 1".

5.5 *Compare* Commands That Control The Table Format

\ <fw> <dp> <pl> <tm> <bm> <tw>

This is the general formatting command, where <fw> <dp> <pl> <tm> <bm> and <tw> all are numbers. This is an line to set the field width of each printed number to <fw>, the number of decimal places to be printed to <dp>, the page length to <pl> lines, the top margin to <tm> lines, the bottom margin to <bm> lines, and the width of the titles to <tw>. The last three may be omitted. The default values are 7, 1, 60, 3, 9, and 32; these will be in effect if this command is omitted. Note that the title width will establish the column width for the first row of a spreadsheet when the $\backslash xls$ command is employed.

\decs <number>

This command changes only the number of decimal places printed, and so is simpler to use than the general formatting command described above. Note that this command overrides the second argument of the general formatting command. Note also that this setting controls the precision when the $\backslash xls$ option is employed.

\gd <number>

\growthdecs <number>

Use this command to override the number of decimal points specified in the *\fw* command, so that growth rates have their own number of decimal points displayed. For example, you may want level variables to have one decimal point, but growth rates to have three. In this case, issue the command "\gd 3".

\growthwidth (gw) <number>

This command allows a different width for the growth rate columns than for the other columns. Note that the width must be at least as big as the date expression ("2000-2009") plus 1, so that 10 would be the smallest width allowed. This feature is handy when printing a lot of growth rates and you need to conserve space.

\field <number>

This command can be used to specify the column width of the data in the table. Give the column width as the argument. Note that this overrides the first argument in the general formatting command.

\pw <number>

\pagewidth <number>

Use this command to set the page width of the table to something other than the default of 132. This controls how the title is centered and where the page numbers will be printed. You may create a table that is wider than this page width, put the page number will not be at the right margin of the data.

\yf

\yearformat

This command lets you specify how you want dates printed. "\yf 4" means to make all dates with four digits. "\yf 2" means to print all dates with two digits; use of two-digit dates is not recommended. Finally "\yf m" means "mixed", i.e., to print levels as four-digit dates and growth rates as two-digit dates. Note that mixed format is the default if none is specified.

\pp

\pageprefix

The default leading text for page numbers is "Page ". If you would like to change it to something like "F -", give the command "\pp F -"

\under <character>

This command specifies the underlining character for dates. For example, for single-line underlining, give the command "\under -". For double-line underlining, give the command "\under =". Any character is legal for underlining except 'n', which turns off the feature.

\pages <on|off>

This command is used to turn page numbering on or off.

\bz <y|n>

\blankzeroes <y|n>

This command allows you to print zeroes as blank fields. This sometimes makes a table less busy and easier to read. The default is 'n', which means that zeroes will be printed. To enable this option, include "\bz y" in the stub file.

\nzr
\nozerorows

The idea for this command is similar to the previous one. Sometimes we want to avoid the clutter of many zeros. Even with zero fields blank, we do not want to waste paper by printing out blank space. After issuing this command in the stub file, series that entirely are zero will be skipped. In this way, you can create a stub file for, say, all 360 sectors of the *Iliad* model for Producers' Durable Equipment and be confident that only the relevant PDE sectors will show up in the table.

\thresh <number>
\threshhold <number>

This is one more wrinkle on the same general idea. If you not only want to avoid looking at zeroes but also want to avoid looking at tiny numbers, then you can set a threshhold value. Any lines of data with no numbers above this threshhold then will not be printed.

\commas <on | off>

Giving this command with the "on" argument, or with no argument, tells *Compare* to print numbers with commas separating thousands, millions, etc. It takes effect at the point in the table where the *\comma* command is given. "\comma off" turns commas back off again. This command helps with the formatting of very large numbers. However, remember that *Compare* and *G7* work in single-precision floating point arithmetic. Therefore, only about seven digits are significant.

\sp <number>
\spacing <number>

This command changes the default spacing between variables or expressions in the table. The notation is similar to that of word processors, where "1" means single spacing. "2" means to insert one blank line between variables or expressions. If you are comparing multiple banks or simulations, the spaces are not inserted between each simulation, but instead only between separate variables. The purpose of this command is to allow space for notes in "scratch" tables, or to make lines of the table stand out when doing multiple simulations.

\missing <text>

This command supplies text to be printed in the table for missing values of variables, or for expressions that default to missing values, because they are based on variables which have missing data. For example: "\miss N/A".

5.6 Commands That Control The Printer

The following commands relate to the control of the printer. They perform their functions by inserting various printer control strings into the *Compare* table file:

\landscape

Issue this command if you want your table to print in landscape mode on a laser printer. You can return to portrait mode in the same table file by giving the command "\landscape off". Note that you might want to increase the pagewidth with the "\pagewidth" command. For example:

```
\pw 170      # set up for wide printing
\landscape
```

You also might want to set the page length to a shorter page. Landscape printing makes it possible to print up to 15 or 16 columns on the same page.

\portrait

Issue this command if you already are in landscape mode and want the rest of the pages of the table to be in portrait mode.

\lm <columns>

This command is used to prevent the left margin of tables from being too close to the side of the page. The number that you supply as <columns> will be the number of columns on the left margin. This makes it easier to put the tables into books. Note that spaces are not inserted into the table, but rather a printer code is added at the top of each page that specifies how many columns to skip before printing. This left margin can be changed dynamically within the document.

\lpp <lines>

This command allows you to specify the number of printer lines per page. This command causes an HP LaserJet printer code to be printed at the top of each page of the table to control the number of vertical lines per inch. Standard settings would be "\lpp 72" or "\lpp 77".

\pc <control string or name>

\pcontrol <control string or name>

This command allows you to send a printer control string directly to the *Compare* table. Printer control strings are used in the Hewlett Packard printer control language (PCL5) to control the formatting and appearance of printouts. Many of the printer functions that can be accessed from the buttons on the printer can also be used by sending printer control strings. In fact, the *Compare* commands "\landscape", "\lpp", "\lm" work by sending a printer control string in the header line of each page. Printer control strings appear somewhat strange to the unititiated. They always begin with the {ESC} character (ASCII 27). For example, the string for setting the printer to portrait mode is {ESC}&l1O (That's "el" "one" "capital O"). To type an {ESC} in the SEE editor, while in Edit mode type "\", then "27". You can refer to the Laser Jet manual for other printer control strings. Since these strings are not easy to remember, we also have created names for commonly used features.

These are listed below. More information may be found at http://h20000.www2.hp.com/bizsupport/TechSupport/Document.jsp?objectID=bpl02705. Note that many of these routines have not been tested; please report trouble.

Command	Description
\pc normal	Reset Primary Style to Upright (Solid)

Job control commands

	Reset
\pc unitofmeasure	Unit of Measure: # = Number of units per inch

	Simplex/Duplex operation NOTE: This is only applicable to duplex capable printers.
\pc duplex or \pc duplexlong	Duplex Print: Long-Edge Binding:
\pc duplexshort	Duplex Print: Short-Edge Binding
\pc duplexlongoffset #	Long-Edge (left) Offset Registration: # of Decipoints (1/720”)
\pc duplexshortoffset #	Short-Edge (Top) Offset Registration: # of Decipoints (1/720”)
\pc pagesidenext	Page Side Selection: Next Side
\pc pagesidefront	Page Side Selection: Front Side
\pc pagesideback	Page Side Selection: Back Side

Page control commands

	Paper destination
\pc destinationauto	Auto Select
\pc destinationtop	Top Output Bin
\pc destinationleft	Left Output Bin

	Optional multi-bin mailbox (HP LaserJet 5Si, 5Si MX, and 5Si NX)
\pc destinationleftup	Left Bin Face Up
\pc destination1down	Bin 1 Face Down
\pc destination2down	Bin 2 Face Down
\pc destination3down	Bin 3 Face Down
\pc destination4down	Bin 4 Face Down
\pc destination5down	Bin 5 Face Down
\pc destination6down	Bin 6 Face Down
\pc destination7down	Bin 7 Face Down
\pc destination8down	Bin 8 Face Down

	Paper source, length, and size CAUTION: Some products require different Paper Source and Page Size commands than those that are contained in this document. This information may be found in documentation specific to those products.
\pc ejectpage	Paper Source: Eject Page
\pc sourcepapercassette	Paper Source: Paper Cassette
\pc sourcepapercassette2	Paper Source: Paper Cassette / tray 2
\pc sourcemanualfeedpaper	Paper Source: Manual Feed Paper
\pc sourcemanualfeedenvelope	Paper Source: Manual Feed Envelope
\pc source1	Paper Source: Tray 1
\pc sourcelowercassette	Paper Source: Optional 500/ 2000 Sheet Lower Cassette
\pc source3	Paper Source: MP Tray/ Tray 3
\pc sourceenvelopefeeder	Paper Source: Envelope Feeder
\pc pagesizeexecutive	Page Size: Executive
\pc pagesizejisexec	Page Size: JISEXEC (8.5 X 13)
\pc pagesizeletter	Page Size: Letter
\pc pagesizelegal	Page Size: Legal
\pc pagesizetabloid	Page Size: Tabloid (11 X 17)
\pc pagesizea4	Page Size: A4
\pc pagesizea3	Page Size: A3
\pc pagesizea5	Page Size: A5
\pc pagesizejisb5	Page Size: JIS B5
\pc pagesizejisb4	Page Size: JIS B4
\pc pagesizejpost	Page Size: JPOST
\pc pagesizejpostd	Page Size: JPOSTD
\pc pagesizemonarch	Page Size: Monarch
\pc pagesizecom10	Page Size: COM 10
\pc pagesizeb5	Page Size: B5

	Orientation
\pc portrait	Orientation: Portrait
\pc landscape	Orientation: Landscape
\pc reverseportrait	Orientation: Reverse Portrait
\pc reverselandscape	Orientation: Reverse Landscape
\pc printdirection	Print Direction: Number of Degrees of Rotation counter- clockwise (90 degree increments only)

	Margins and text length
\pc topmargin	Top Margin: Number of Lines
\pc textlength	Text Length: Number of Lines
\pc leftmargin	Left Margin: Number of Columns
\pc rightmargin	Right Margin: Number of Columns
\pc clearhorizontalmargins	Clear Horizontal Margins

	Perforation skip mode
\pc perforationskipdisable	Perforation Skip: Disable
\pc perforationskipenable	Perforation Skip: Enable

	Horizontal column spacing The Horizontal Motion Index (HMI) command designates the distance between columns in .0083-inch increments. When fixed pitch fonts are selected, all printable characters, including the space and backspace characters, are affected by HMI. When proportional fonts are selected, the HMI affects only the control code space character. The default HMI is equal to the pitch value in the font header. The printer escape sequence to be sent is as follows: Ec&k#H # is equal to a variable that is derived from the following formula: Horizontal Printable Area X 120 = # Desired characters per line. If the desired pitch or number of characters per horizontal inch is known, use the following formula: 120/Desired characters per inch = #
\pc hmi #	Horizontal Motion Index (HMI): Number of .0083-inch Increments

	Vertical line spacing The Vertical Motion Index (VMI) command designates the distance between rows in .0208-inch increments (the vertical distance that the cursor will move for a line feed operation). This command affects the line feed and half-line feed spacing. The factory default VMI is eight, which corresponds to six lines per inch. VMI can be selected from the printer control panel or by sending a printer escape sequence: Ec&#C . # is equal to a variable that is derived from the following formula: Vertical Printable Area X 48 = # Desired Lines Per Page Note: Use either VMI (most precise method of line spacing) or Lines per inch, as only the last command will execute.
\pc vmi #	Vertical Motion Index (VMI): # of .0208-inch Increments

	Line Spacing (Lines per inch)
\pci 1	Line Spacing (Lines per inch): 1 line/inch
\pci 2	Line Spacing (Lines per inch): 2 lines/inch
\pci 3	Line Spacing (Lines per inch): 3 lines/inch
\pci 4	Line Spacing (Lines per inch): 4 lines/inch
\pci 6	Line Spacing (Lines per inch): 6 lines/inch
\pci 8	Line Spacing (Lines per inch): 8 lines/inch
\pci 12	Line Spacing (Lines per inch): 12 lines/inch
\pci 16	Line Spacing (Lines per inch): 16 lines/inch
\pci 24	Line Spacing (Lines per inch): 24 lines/inch
\pci 48	Line Spacing (Lines per inch): 48 lines/inch

	Vertical and horizontal
\pc vposition	Vertical Position: # of Rows
\pc superscript	Superscript: Up $\frac{1}{4}$ Row
\pc subscript	Subscript: Down $\frac{1}{4}$ Row
\pc vpositiondots	........: # of Dots
\pc vpositiondecipoints	.........: # of Decipoints
\pc hposition	Horizontal Position: # of Columns
\pc vpositiondots	........: # of Dots
\pc vpositiondecipoints	.........: # of Decipoints
\pc halflinefeed	Half Line Feed: N/A

	End-of-line termination
\pc linetermination1	Line Termination: CR=CR;LF=LF; FF=FF
\pc linetermination2	Line Termination: CR=CR+LF;LF=LF FF=FF
\pc linetermination3	Line Termination: CR=CR; LF=CR+LF; FF=CR+FF
\pc linetermination4	Line Termination: CR=CR+LF; LF=CR+LF; FF=CR+FF

	Symbol set selection Refer to the PCL-5 Comparison Guide for additional supported symbol
\pc symbolnorwegian	Primary Symbol Set: ISO Norwegian
\pc symboluk	Primary Symbol Set: ISO 4: United Kingdom
\pc symbolwindowslatin2	Primary Symbol Set: Windows 3.1 Latin 2
\pc symbolfrench	Primary Symbol Set: ISO 69: French
\pc symbolgerman	Primary Symbol Set: ISO 21: German
\pc symbolitalian	Primary Symbol Set: ISO 15: Italian
\pc symbolmicrosoft	Primary Symbol Set: Microsoft (R) Publishing
\pc symboldeskp or sym_desktop	Primary Symbol Set: DeskTop
\pc symbolpstext	Primary Symbol Set: PS Text
\pc symbolmctext	Primary Symbol Set: MC Text
\pc symbolventurainternational	Primary Symbol Set: Ventura International
\pc symbolventuraus	Primary Symbol Set: Ventura US
\pc symbolwingdings	Primary Symbol Set: Wingdings
\pc symbolpsmath	Primary Symbol Set: PS Math
\pc symbolventuramath	Primary Symbol Set: Ventura Math
\pc symbolmath8	Primary Symbol Set: Math-8
\pc symbolsymbol	Primary Symbol Set: Symbol
\pc symbollatin1	Primary Symbol Set: ISO 8859-1 (ECMA-94) Latin 1
\pc symbollatin2	Primary Symbol Set: ISO 8859-2: Latin 2
\pc symbollatin5	Primary Symbol Set: ISO 8859-9: Latin 5
\pc symbolswedish	Primary Symbol Set: ISO 11: Swedish
\pc symbolspanish	Primary Symbol Set: ISO 17: Spanish
\pc symbolwindowslatin5	Primary Symbol Set: Windows 3.1 Latin 5
\pc symbolturkish	Primary Symbol Set: PC Turkish
\pc symbolascii	Primary Symbol Set: ISO 6: ASCII
\pc symbollegal	Primary Symbol Set: Legal
\pc symbolrom8 or sym_roman8	Primary Symbol Set: Roman-8
\pc symbolwindowslatin1	Primary Symbol Set: Windows 3.0 Latin 1
\pc symbol8 or sym_437	Primary Symbol Set: PC-8
\pc symbolpcn or sym_437n	Primary Symbol Set: PC-8 D/N
\pc symbolpc850	Primary Symbol Set: PC 850
\pc symbolpi	Primary Symbol Set: Pi Font
\pc symbolpc852	Primary Symbol Set: PC-852
\pc symbolwindowslat1 or \pc sym_winlatin1	Primary Symbol Set: Windows 3.1 Latin 1 (ANSI)

	Spacing
\pc fixed	Primary Spacing: Fixed
\pc proportional	Primary Spacing: Proportional

	Pitch
\pc pitch #	Primary Pitch: Number Characters per inch
\pc pitch10 or \pc large	Set Pitch Mode: 10.0
\pc pitchcompressed or \pc small	Set Pitch Mode: Compressed (16.5-16.7)
\pc pitchelite or \pc med	Set Pitch Mode: Elite (12.0)

	Point size
\pc point #	Primary Height: # Points

	Style Additional style values may be obtained from the related documentation provided with HP's font products. PCL 5 HP LaserJet printers allow complex structures (contours, outlines, shading, etc.) and widths, as well as posture to be specified. Refer to the HP PCL 5 Printer Language Technical Reference Manual.
\pc upright	Primary Style: Upright (Solid)
\pc italic	Primary Style: Italic
\pc condensed	Primary Style: Condensed
\pc compressed italic	Primary Style: Condensed Italic
\pc compressed	Primary Style: Compressed (Extra Condensed)
\pc expanded	Primary Style: Expanded
\pc outline	Primary Style: Outline
\pc inline	Primary Style: Inline
\pc shadowed	Primary Style: Shadowed
\pc outline-shadowed	Primary Style: Outline Shadowed

	Stroke weight
\pc ultrathin	Primary Font Stroke Weight: Ultra Thin
\pc extrathin	Primary Font Stroke Weight: Extra Thin
\pc thin	Primary Font Stroke Weight: Thin
\pc extralight	Primary Font Stroke Weight: Extra Light
\pc light	Primary Font Stroke Weight: Light
\pc demilight	Primary Font Stroke Weight: Demi Light
\pc semilight	Primary Font Stroke Weight: Semi Light
\pc medium	Primary Font Stroke Weight: Medium (book or text)
\pc semibold	Primary Font Stroke Weight: Semi Bold
\pc demibold	Primary Font Stroke Weight: Demi Bold
\pc bold	Primary Font Stroke Weight: Bold
\pc extrabold	Primary Font Stroke Weight: Extra Bold
\pc black	Primary Font Stroke Weight: Black
\pc extrablack	Primary Font Stroke Weight: Extra Black
\pc ultrablack	Primary Font Stroke Weight: Ultra Black

	Primary typeface family
\pc lineprinter	Typeface Family: Lineprinter
\pc albertus	Typeface Family: Albertus
\pc antiqueolive	Typeface Family: Antique Olive
\pc clarendon	Typeface Family: Clarendon
\pc coronet	Typeface Family: Coronet
\pc courier	Typeface Family: Courier
\pc garamondantiqua	Typeface Family: Garamond Antiqua
\pc lettergothic	Typeface Family: Letter Gothic
\pc marigold	Typeface Family: Marigold
\pc cgomega	Typeface Family: CG Omega
\pc cgtimes	Typeface Family: CG Times
\pc univers	Typeface Family: Univers
\pc arial	Typeface Family: Arial (R)
\pc timesnewroman	Typeface Family: Times New Roman
\pc symbol	Typeface Family: Symbol
\pc wingdings	Typeface Family: Wingdings
\pc times	
\pc timesbold	
\pc timesitalic	
\pc timesbolditalic	
\pc courier	
\pc courierbold	
\pc courieritalic	
\pc courierbolditalic	
\pc arialbold	
\pc arialitalic	
\pc arialbolditalic	
\pc garamond	

Underline	
\pc underlinefixed	Underline: Enable Fixed
\pc underlinefloating	Underline: Enable Floating
\pc underlinedisable	Underline: Disable

Note that any printer control string that works on your printer can be inserted into the file with this command.

\noformat

Compare achieves features such as "\lpp" (lines per page), "\landscape" (landscape mode), and "\lm" (left margin) by insertting HP LaserJet printer codes in the top of the table file. Sometimes, such as when printing to a 9-pin printer, you want to force these codes off. Use the "\noformat" command to do this.

\nocomment

Issue this command when using the *\gdata* command to inhibit printing of comment lines in a file.

5.7 Commands to Build Tables in Excel Format

Many commands in this document apply regardless of the file format that has been selected for the document that is to be printed. Several commands apply only to the creation of spreadsheets with the *\xls* command. Use of these features requires that Microsoft Excel is installed on the user's machine.

\xls [<filename>]

Issue this command to create spreadsheet documents. If a filename is specified, then it will be employed to name the document. If no name is given, then the name will be specified according to the *.IN configuration file.

\filetype <type>

The default file type is XLS. Available file types include

File Types	
AddIn (.xlam)	WorkbookNormal (.xls)
CSV (.csv)	SYLK (.slk)
CSVMac (.csv)	Template (.xltx)
CSVMSDOS (.csv)	TextMac (.txt)
CSVWindows (.csv)	TextMSDOS (.txt)
DBF2 (.dbf)	TextPrinter (.txt)
DBF3 (.dbf)	TextWindows (.txt)
DBF4 (.dbf)	WK1 (.wk1)
DIF (.dif)	WK1ALL (.wk1)
Excel2 (.xls)	WK1FMT (.wk1)
Excel2FarEast (.xls)	WK3 (.wk3)
Excel3 (.xls)	WK4 (.wk4)
Excel4 (.xls)	WK3FM3 (.wk3)
Excel5 (.xls)	WKS (.wks)
Excel7 (.xls)	WQ1 (.wq1)
Excel9795 (.xls)	UnicodeText (.txt)
Excel4Workbook (.xls)	Html (.html)
IntlAddIn (.xla)	XLS (.xls)
IntlMacro (.xlsm)	

If no extension is provided with the filename, then the appropriate extension will be determined according to the file type and will be appended to the file name.

\wstitle <worksheet name>

This command names the current worksheet when printing to a Microsoft Excel spreadsheet. New worksheets are created in the standard fashion; that is, the '*' command creates a new worksheet.

\xlgraph <start date> <end date> <graph style>
\xlgraph title "<title>"
\xlgraph subtitle "<subtitle>"
\xlgraph vaxtitle "<vaxtitle>"
<series 1> ;Series 1 title
. . .
<series N> ;Series N title
\xlgraph

Create a graph sheet in the current workbook. The graph will appear in a sheet to the left of the active worksheet. Specify the the date range, in *G7*-style dates, for the data to be graphed. Time will appear on the horizontal axis. Text may be given for the graph title, subtitle, and vertical axis title; text must be wrapped in quotation marks. Up to 29 series may be specified between the opening "\xlgraph" and the closing "\xlgraph" commands, including any alternative series if more than one bank is loaded. Available graph styles include:

Graph Styles	
ColumnClustered	Bubble3DEffect
ColumnStacked	StockHLC
ColumnStacked100	StockOHLC
3DColumnClustered	StockVHLC
3DColumnStacked	StockVOHLC
3DColumnStacked100	CylinderColClustered
BarClustered	CylinderColStacked
BarStacked	CylinderColStacked100
BarStacked100	CylinderBarClustered
3DBarClustered	CylinderBarStacked
3DBarStacked	CylinderBarStacked100
3DBarStacked100	CylinderCol
LineStacked	ConeColClustered
LineStacked100	ConeColStacked
LineMarkers	ConeColStacked100
LineMarkersStacked	ConeBarClustered
LineMarkersStacked100	ConeBarStacked
PieOfPie	ConeBarStacked100
PieExploded	ConeCol
3DPieExploded	PyramidColClustered
BarOfPie	PyramidColStacked
XYScatterSmooth	PyramidColStacked100
XYScatterSmoothNoMarkers	PyramidBarClustered
XYScatterLines	PyramidBarStacked
XYScatterLinesNoMarkers	PyramidBarStacked100
AreaStacked	PyramidCol
AreaStacked100	3DColumn

continues on next page

Table 2 – continued from previous page

3DAreaStacked	Line
3DAreaStacked100	3DLine
DoughnutExploded	3DPie
RadarMarkers	Pie
RadarFilled	XYScatter
Surface	3DArea
SurfaceWireframe	Area
SurfaceTopView	Doughnut
SurfaceTopViewWireframe	Radar
Bubble	

Compare allows the user to format specific parts of an Excel spreadsheet by inserting commands into a stub file. The font in the header, title, dates, banks, series names, and data can changed. Specifically, the type face, size, color, vertical and horizontal alignment, bold status, italic status, and underline status and type can be controlled. The number and order of settings does not matter. The type face is specified by giving the name surrounded by double quotes. Size is adjusted by entering "size" followed by an integer. Horizontal alignment is controlled by entering "justify", "center", "right", or "left" for the horizontal alignment. Vertical alignment can be specified by "top", "vcenter", "bottom", or "vjustify". A complete listing of Excel font settings are listed in the appendix at the end of this page. To remove a formatting option for any area of an Excel spreadsheet, enter "clear" as the setting.

\fonthead <settings>

This command will control formatting for the \head output.

For example,

```
\fonthead "arial" size32 bold underline
```

\fonttitle <settings>

This command will control formatting for the \title output.

For example,

```
\fonttitle "times new roman" size24 bold
```

\fontdate <settings>

This command will control formatting for \date output.

For example,

```
\fontdate "times new roman" size14 underline center
```

\fontbank <settings>

This command will control formatting for the printed names of data banks.

For example,

```
\fontbank  "courier new"   size10
```

\fontname <settings>

This command will control formatting for the data series names.

For example,

```
\fontname "verdana" size12 italic left
```

\fontdata <settings>

This command will control formatting for the printing of numerical data.

For example,

```
\fontdata  "arial"  size11  right
```

\font <settings>

This command is used to set the default font formatting and will override all other font commands.

For example,

```
\font  "courier new"  size12   left
```

Appendix: Excel Font Format Settings

Underline

Underlining may be specified in several ways.

underline [-]
UnderlineSingle
-

Single underlining.

underline =
UnderlineDouble
=

Double underlining.

underline _
UnderlineSingleAccounting
_

Single accounting underlining.

underline [~]
UnderlineDoubleAccounting
~

Double accounting underlining.

underline n
UnderlineNone

No underlining.

Font Colors

Available font colors include:

Font Colors	
“None”	“Medium Gray”
“Aqua”	“Mint green”
“Black”	“Navy blue”
“Blue”	“Olive green”
“Cream”	“Purple”
“Dark Gray”	“Red”
“Fuchsia”	“Silver”
“Gray”	“Sky blue”
“Green”	“Teal”
“Lime green”	“White”
“Light Gray”	“Yellow”
“Maroon”	

Font Types

Available font types include:

Font Types	
“Agency FB”	“Gill Sans Ultra Bold Condensed”
“Agency FB Bold”	“Gloucester MT Extra Condensed”
“Algerian”	“Goudy Old Style”
“Arial”	“Goudy Old Style Bold”
“Arial Black”	“Goudy Old Style Italic”
“Arial Black Italic”	“Goudy Stout”
“Arial Bold”	“Haettenschweiler”
“Arial Bold Italic”	“Harlow Solid Italic”
“Arial Italic”	“Harrington”
“Arial Narrow”	“High Tower Text”
“Arial Narrow Bold”	“High Tower Text Italic”
“Arial Narrow Bold Italic”	“Impact”
“Arial Narrow Italic”	“Imprint MT Shadow”
“Arial Rounded MT Bold”	“Informal Roman”
“Arial Unicode MS”	“Jokerman”
“Baskerville Old Face”	“Juice ITC”
“Batang”	“Kristen ITC”
“Bauhaus 93”	“Kunstler Script”
“Bell MT”	“Lucida Bright”
“Bell MT Bold”	“Lucida Bright Demibold”
“Bell MT Italic”	“Lucida Bright Demibold Italic”
“Berlin Sans FB”	“Lucida Bright Italic”
“Berlin Sans FB Bold”	“Lucida Calligraphy Italic”
“Berlin Sans FB Demi Bold”	“Lucida Fax Demibold”
“Bernard MT Condensed”	“Lucida Fax Demibold Italic”
“Blackadder ITC”	“Lucida Fax Italic”
“Bodoni MT”	“Lucida Fax Regular”
“Bodoni MT Black”	“Lucida Handwriting Italic”
“Bodoni MT Black Italic”	“Lucida Sans Demibold Italic”
“Bodoni MT Bold”	“Lucida Sans Demibold Roman”
“Bodoni MT Bold Italic”	“Lucida Sans Italic”

continues on next page

Table 3 – continued from previous page

“Bodoni MT Condensed”	“Lucida Sans Regular”
“Bodoni MT Condensed Bold”	“Lucida Sans Typewriter Bold”
“Bodoni MT Condensed Bold Italic”	“Lucida Sans Typewriter Bold Oblique”
“Bodoni MT Condensed Italic”	“Lucida Sans Typewriter Oblique”
“Bodoni MT Italic”	“Lucida Sans Typewriter Regular”
“Bodoni MT Poster Compressed”	“Magneto Bold”
“Book Antiqua”	“Maiandra GD”
“Book Antiqua Bold”	“Map Symbols”
“Book Antiqua Bold Italic”	“Matura MT Script Capitals”
“Book Antiqua Italic”	“Mistral”
“Bookman Old Style”	“Modern No. 20”
“Bookman Old Style Bold”	“Monotype Corsiva”
“Bookman Old Style Bold Italic”	“MS Mincho”
“Bookman Old Style Italic”	“MS Outlook”
“Bradley Hand ITC”	“MT Extra”
“Britannic Bold”	“Niagara Engraved”
“Broadway”	“Niagara Solid”
“Brush Script MT Italic”	“OCR A Extended”
“Californian FB”	“Old English Text MT”
“Californian FB Bold”	“Onyx”
“Californian FB Italic”	“Palace Script MT”
“Calisto MT”	“Palatino Linotype”
“Calisto MT Bold”	“Palatino Linotype Bold”
“Calisto MT Bold Italic”	“Palatino Linotype Bold Italic”
“Calisto MT Italic”	“Palatino Linotype Italic”
“Castellar”	“Papyrus”
“Centaur”	“Parchment”
“Century”	“Perpetua”
“Century Gothic”	“Perpetua Bold”
“Century Gothic Bold”	“Perpetua Bold Italic”
“Century Gothic Bold Italic”	“Perpetua Italic”
“Century Gothic Italic”	“Perpetua Titling MT Bold”
“Century Schoolbook”	“Perpetua Titling MT Light”
“Century Schoolbook Bold”	“Playbill”
“Century Schoolbook Bold Italic”	“PMingLiU”
“Century Schoolbook Italic”	“Poor Richard”
“Chiller”	“Pristina”
“Colonna MT”	“Rage Italic”
“Comic Sans MS”	“Ravie”
“Comic Sans MS Bold”	“Rockwell”
“Cooper Black”	“Rockwell Bold”
“Copperplate Gothic Bold”	“Rockwell Bold Italic”
“Copperplate Gothic Light”	“Rockwell Condensed”
“Courier New”	“Rockwell Condensed Bold”
“Courier New Bold”	“Rockwell Extra Bold”
“Courier New Bold Italic”	“Rockwell Italic”
“Courier New Italic”	“Script MT Bold”
“Curlz MT”	“Showcard Gothic”
“Edwardian Script ITC”	“SimSun”
“Elephant”	“Snap ITC”
“Elephant Italic”	“Stencil”
“Engravers MT”	“Symbol”

continues on next page

Table 3 – continued from previous page

"Eras Bold ITC"	"Tahoma"
"Eras Demi ITC"	"Tahoma Bold"
"Eras Light ITC"	"Tempus Sans ITC"
"Eras Medium ITC"	"Times"
"Felix Titling"	"Times New Roman"
"Footlight MT Light"	"Times New Roman Bold"
"Forte"	"Times New Roman Bold Italic"
"Franklin Gothic Book"	"Times New Roman Italic"
"Franklin Gothic Book Italic"	"Trebuchet MS"
"Franklin Gothic Demi"	"Trebuchet MS Bold"
"Franklin Gothic Demi Cond"	"Trebuchet MS Bold Italic"
"Franklin Gothic Demi Italic"	"Trebuchet MS Italic"
"Franklin Gothic Heavy"	"Tw Cen MT"
"Franklin Gothic Heavy Italic"	"Tw Cen MT Bold"
"Franklin Gothic Medium"	"Tw Cen MT Bold Italic"
"Franklin Gothic Medium Cond"	"Tw Cen MT Condensed"
"Franklin Gothic Medium Italic"	"Tw Cen MT Condensed Bold"
"Freestyle Script"	"Tw Cen MT Condensed Extra Bold"
"French Script MT"	"Tw Cen MT Italic"
"Garamond"	"Verdana"
"Garamond Bold"	"Verdana Bold"
"Garamond Italic"	"Verdana Bold Italic"
"Gigi"	"Verdana Italic"
"Gill Sans MT"	"Viner Hand ITC"
"Gill Sans MT Bold"	"Vivaldi Italic"
"Gill Sans MT Bold Italic"	"Vladimir Script"
"Gill Sans MT Condensed"	"Wide Latin"
"Gill Sans MT Ext Condensed Bold"	"Wingdings"
"Gill Sans MT Italic"	"Wingdings 2"
"Gill Sans Ultra Bold"	"Wingdings 3"

5.8 Calculation of Data Within *Compare*

The *\f* command works like the *f* command in *G7*. The original motivation for the *\f* command was to create variables that could be used later in the *Compare* table, or to create intermediate variables to be used in further *\f* calculations.

\f <name> = <expression>

This command is parallel to the *f* command in *G7*. The format is exactly the same as in *G7*. The *\f* command works by creating a "*Compare* workspace bank" for each simulation or databank that is being compared. The name of each bank is of the form "CWSx.BNK", where "x" is the number of the simulation, starting at zero. Therefore, after making a *Compare* table with simulations and using the *\f* command in the process, there should be banks CWS0.BNK, CWS1.BNK, CWS2.BNK, and CWS3.BNK in your directory. They will contain any series created with the *\f* command. When *Compare* is looking for a series, it checks first in the *Compare* workspace bank to see if the series has been put there by a *\f* command. Then it checks the corresponding simulation bank.

At present, the *Compare* workspace banks are not created until the first use of the *\f* command, and the banks remain in use after *Compare* is finished. The banks can be assigned within *G7* just like any other workspace bank, so you may find further uses for them beyond what originally was intended.

5.9 Calculation and Display of Running Totals

The next three commands work together and are used for the calculation and display of running totals.

\st
\starttotal

This and the two following commands are used together to calculate running totals. The *\st* command should be placed before the first variable that is to be included in the running total.

\et
\endtotal

This command should be placed after the last variable that is to be included in the running total.

\pt <series title text>
\printtotal <series title text>

To print the total in the table, issue the *\pt* command. The <series title text> is whatever text you want to appear to the left of the data series, such as “Total for All Industries”.

5.10 Commands for Sorting or Ranking

These three commands work together to enable rankings to be made with *Compare*. They enable rankings by levels, sums, growth rates, or averages. The *\startsort* command is given before the range of lines that is to be sorted, and *\endsort* is given after that range. Printing of those lines will be suppressed until the first *\printsort* command, and printing of the rest of the table should resume after *\printsort*. Note that for these features to work, all three commands must be in the same file. In other words, do not put *\add* or *\fadd* commands between the *\startsort* and *\endsort*

\ss <date expression> [<size to allocate>]
\startsort <date expression> [<size to allocate>]

Put the *\startsort* command before the lines in the stub file that you would like to sort. The date expression represents the number that you will use as the criteria for the sort. This can be a value for single year, a range of years indicating a growth rate (e.g. 1990-1997), a sum (1990+1997), or an average (1990|1997). Note that the date expression used for the sorting need not be a date expression actually used in the table, but it can be any valid date expression that makes sense for the bank that you are using. Note that if you are comparing several banks, the sorting will be based on the value of the variables for this date expression in the first simulation bank.

The last argument for *\startsort* is optional. By default, enough space is set up to sort up to 500 lines. If you want to save memory, you may reduce this number, or if you need more than 500 you need to increase it. Note that *Compare* will sort only lines that normally would print values, i.e., either variables or expressions. All other lines such as *\center*, *\line*, *;*, etc. will be skipped.

Example:

```
\startsort 90-95 1000
```

\es
\endsort

The *\endsort* command marks the end of the range of the table that will be sorted. Note that all table lines between the *\startsort* and *\endsort* commands will not be printed until a *\printsort* command is given.

\ps [<direction>] [<numtoprint>]
\printsort [<direction>] [<numtoprint>]

The *\printsort* command is the place where the work is done. Both arguments are optional. For the first, the sort direction is 'd' (descending) by default. If you want the sort to be in ascending order, give an 'a'. The second option is the number of items you would like to print.

For example, if you want to print the 10 largest items, you would use:

```
\printsort d 10
```

5.11 Miscellaneous Commands

\toc This command turns on table of contents generation. The table of contents will be generated as the last page of the table. The page heading will be the title of the entire table, as given in the *\head* command. Each entry will be triggered by the presence of a *\ti* command either in the main stub file or in any stub files called by *\add*. If table of contents generation is turned on, then each table title will be preceeded by the text: "TABLE 1.", "TABLE 2.", etc. Page numbers in the table of contents will have the page prefix tacked on, as given by the *\pageprefix* command.

\ts
\timestamp or \ts

This command will specify the printing of a date and time, centered at the bottom of each page. This is the date and time that the table was generated.

\mem

This command reports the amount of memory that currently is available. If you are working with large *Interdyme* matrices or long *@csum()* commands, you might bump into memory limits.

\tell

This command tells the time and date on the screen to aid in timing the creation of tables.

\announce <announcement>

This command is useful if you want to announce when a certain portion of a stub file is being processed, either for debugging reasons or just to monitor how far you are in the process. If you are using a lot of *\add* files, you might want to put an *\announce* command near the top of each one so you can trace when each one is done.

\at (or \aggtype)

\at (or \aggtype)

This command changes the way that data are calculated “at annual rates”. This is a subject that is clouded by the fact that people do not agree on what is meant by “annual rates.” The Inforum approach, which is the default (”\at i”), yields annualized growth rates that exactly are the frequency of the data times the period growth rates. This internally is consistent and gives annualized growth rates that yield the same numbers as the corresponding annual growth rates. The BEA approach, which seems to have been adopted by just about everyone else (the CEA, the Fed and the Washington Post, to name a few), is to take the period-to-period ratio to the *n*-th power (where *n* is the frequency) and then subtract 1.0 and multiply by 100. In other words, where *xt* is the current quarter and *x0* is the previous quarter, this method is calculated by the following formula:

$$100.0 \times \left(\exp\left(4.0 \times \log\left(\frac{xt}{x0}\right)\right) - 1.0\right)$$

It is not clear what is meant by “annual rates” in this sense, but this is approach often is employed. To use the BEA approach, give the command “\at b”.

\ar

Use if monthly or quarterly data is given at annual rates (the default). “ar” indicates “annual rates.”

\pr

Use if monthly or quarterly data is given at monthly or quarterly rates, where “pr” signifies “period rates.” Note that it only makes sense to compare period and annual rates if the growth type is exponential (‘e’).

5.12 Groups in *Compare*

Compare is been able to use group expressions in *@csum()* functions, including the named groups that may be used in *G7*. Named groups are created by running *G7* or the *Fixer* program, which creates a groups file called GROUPS.BIN. If a GROUPS.BIN file is in the current directory, then *Compare* will read it and store the group definitions in that file. For example, if the *Fixer* input file was VECFIX.VFX, and it had as input:

```
group Manufacturing
9-58
```

then the group called “Manufacturing” would be created and stored in GROUPS.BIN. This then could be used in the *Compare* *@csum()* function in a stub file:

```
@csum(emp,:Manufacturing) ; Manufacturing employment
```

Note that a colon (‘:’) is given as a prefix for the group name.

Some other commands that can be used for checking the groups defined in the GROUPS.BIN file are useful for working with Vam files.

\listgroups

This will print a list of the currently-defined groups to the screen.

\glist <groupname>

If <groupname> is one of the groups printed in the list above, then this command will print those sectors or categories comprising the group.

Note that neither of these commands affects the output of the table, but are used only to provide infomation to the screen.

5.13 Commands Used With Vam Files

Note that when printing a table of series from *Interdyme*, it usually is advantageous to issue the *\load* command before printing the vector. This loads the data for the entire vector into memory and allows much faster processing. Also, note that you can print time series of matrix elements in *Compare*. To print the values of the A-matrix for row 1, column 1, you would insert the series name "am1.1", where "am" is the name given to the A-matrix in the *Interdyme* VAM.CFG file. Note the dot that separates the row and the column.

\load <vectorname>

This command can be useful when using *Compare* to print tables of vector variables, for example if you are printing a table of many industries or sectors for the same vector. Use of this command might allow a significant increase in speed.

\matlist <groupdef>

This command tells *Compare* to print a matrix listing. Note that this command only works on a vam file, and the program will complain about any other type of file. The group definition can be a simple list of sectors, or it can include ranges of sectors specified by two sector numbers separated by a dash. Sectors or intervals in parentheses will be excluded.

Example:

```
\matlist 20 24 30-50 (41 45 46-48)
```

would perform a matrix listing for sectors 20,24, 30 through 50, with 41, 45, and 46 through 48 excluded.

\row

This command is to be given before requesting the matrix listing. It specifies that a row listing is to be printed.

\column

This command specifies that a column listing is to be printed, and also must be given before the *\matlist* command.

\cutoff <cutoff ratio> [<year>]

This command specifies the cutoff, in as a ratio in terms of output, for what flows to print. For example "\cutoff 0.005" says to print all flows that are greater than 0.5% of output. You can specify the cutoff year (optional), and then the data for that year will be used to determine if the line should be printed in the matrix listing or not.

\cd <number>

\coefdecs <number>

This command really is relevant only for matrix listings, as it allows you to specify how many decimal places to use for the display of I-O coefficients.

\nst

This command is used to turn off "seller titles" in matrix listings. In some cases, the matrix listing is used to print detailed flows from a matrix identity other than the typical buyer or seller listing. An example would be a table of flows of employment by occupation.

\nosubtot

This command, also for matrix listings for *Interdyme*, will suppress the printing of subtotals at the bottom of a matrix listing section. This is useful when the items of the matrix already include aggregates, where calculating a subtotal would imply double counting.

\mc <filename>

\matcfg <filename>

This option allows you to specify a different configuration file for a matrix listing besides the normal "MATLIST.CFG". The matlist configuration file specifies the matrix identity that the matrix listing will summarize. In a complicated model with many matrix identities, such as a regional model, you may need many alternative configuration files. This command allows you to switch easily between these files.

5.14 Variables, Expressions, and Functions

A line beginning in any other way will be presumed to begin with a variable name, such as gnp$, or an expression. For the expressions, most of the functions available in *G7* also are available in *Compare*. In particular, *Compare* has a useful function for sectoral models, called *@csum()*, which is used to sum up a specified group of industries for a given data concept. For example, suppose you wanted a certain line in your table to contain the sum of exports ("exp") for sectors 1 to 10. Then the formula to use in the name section of the stub file would be:

```
@csum(exp,1-10)
```

The expression:

```
@csum(exp,1-12 (4-7) 15 18)
```

would include exports of sectors 1 to 12 inclusive, except for sectors 4 to 7 inclusive, and then sectors 15 and 18 in addition.

Note that when working with Vam files, a series name is followed by the vector name and the sector. Therefore, output of sector 10 would be "out10", if the output vector is named "out". To print a matrix element, use the matrix name, plus the row index, then a period ("."), and finally the column index. For example, "am1.1" is am(1,1).

After the name or expression comes a ';'and then a line title that will be printed at the left side of the line. The length of this line title is specified by the formatting command described above. Leading blanks – blanks between the ";" and the first visible letter on the line – will be printed and can be used to add indentation for the titles.

5.15 Running the *Compare* Program

Once the ".STB" file is ready, one can run *Compare* by typing:

```
compare
```

at the DOS prompt.

As soon as *Compare* starts, it asks you how many alternatives you want to see in the table (see the sample session in the box below). Of course, respond with '1' if you are just going to make a table from one bank. You may have up to 10 alternatives. Next, for each alternative, you are asked to specify from what type of bank this alternative should be read and then the rootname of the data bank file. Answer this first question with one character, and then hit {ENTER}. Type 'w' if this is a normal G bank or workspace-type bank (.BNK); 'c' if this is a compressed G bank (.CBK); 'h' if this is a hashed G bank (.HBK); 'd' if this is a dirfor file (.DFR); and 'v' if this is a vam file (.VAM). Note that if you want to use a dirfor file, you must have the DIRFOR.DAT corresponding to that file in the current directory in which you are running *Compare*. Make sure that the MACRONAM.BIN specified in that file points to a valid location. (See the Display chapter of the *LIFT* manual for more details on DIRFOR.DAT.) If you are assigning a vam file, note that *Compare* automatically assumes that each Vam file has a "sister" G bank, with the same root name and in the same directory. It will try to open this file to find series such as macrovariables that might not be in the Vam file. See the *G7* or *InterDyme* manual for more details on Vam files. Remember with all databank file names to give only the root name of the file (with no file extension).

Next, if you are printing from more than one bank, you are asked whether you would like to see the alternatives in actual values ('a'), as differences ('d') from the base run, or as percentage differences ('p') from the base. Type the desired letter and press [Enter] to show your choice. (If you are listing only one data bank or a single run of a model, it does not matter how you answer this question.)

You then will be asked for the name of the stub file, and you should reply with the name of the stub file that you have prepared. (Use the full name, including the ".STB"). Finally, *Compare* will ask for the name for the output file. We commonly use the ".OUT" extension for these files, but any name is OK. When *Compare* has finished and has given the DOS prompt again, you can use the DOS Print command to obtain a printed copy.

As an alternative to answering the individual questions asked by *Compare*, you can put the answers into a file such as "COMPARE.IN" and start the program with

```
compare compare.in
```

Compare assumes that a filename given on the command line is a file containing input responses. The box below shows the response file that would correspond to the example on the previous page.

```
1
v
dyme
mast.stb
tabpf.out
```

A configuration file format also is available, which can be used by specifying the "-f" option on the command line. For example, if your configuration file is named COMPARE.CFG, you can specify using this with: "compare -f compare.cfg". Note the space between the "-f" and the file name. An example config file is in the box below.

```
Number of Simulations; 1
1st bank type;          h
```

(continues on next page)

(continued from previous page)

```
1st bank name;         c:\ami\quip
Stub file;           c:\ami\quip.stb
Output File;          quip.out
```

Note that for the *IdLift* model, a stub file called DYME.STB already has been prepared which contains most of the macrovariables and many of the sectoral variables in the model. The *QUEST* model has stub files called LONGTERM.STB and SHORTTRM.STB that produce tables similar to those in the *QUEST* section of the meeting book. If you would like to make tables containing only portions of these file, you can cut and paste to create new stub files. These files also can be used to print or graph data in *G7* using the *look* command (see the *G7* documentation).

5.16 Special Uses with *Compare*

This section describes how to make a matrix listing for *Interdyme* using *Compare*, and how to make tables using Dirfor files.

Using *Compare* as a Matrix Lister for Interdyme

A "matrix listing" displays all of the cells in a row or column of an input-output table. The command in the stub file simply is the following.

\matlist <sectors>

For example, it could be

```
\matlist 1 5 7 15-30 (21 23-25)
```

where we have used the now-familiar system of indicating a list of sectors.

Two related commands should the *\matlist* command: *\row* or *\column* and *\cutoff*. Here is a complete example of a stub file for a matrix listing:

```
\date 1987 1988 1989 1991 1993 1995 2000 1991-1995 1990-2000 1995-2000
\ti MUDAN MATRIX LISTING
\9 1 66 1 3 33
\row
\cutoff 0.02
\matlist 18-25
```

This will produce a row listing; a cell of a matrix must account for at least 0.02 of the total (two percent of the row total) in order to be listed. Rows 18 - 25 will be listed.

The *\matlist* command causes *Compare* to look for a file which at present must be called "MATLIST.CFG". Here is an example from Mudan.

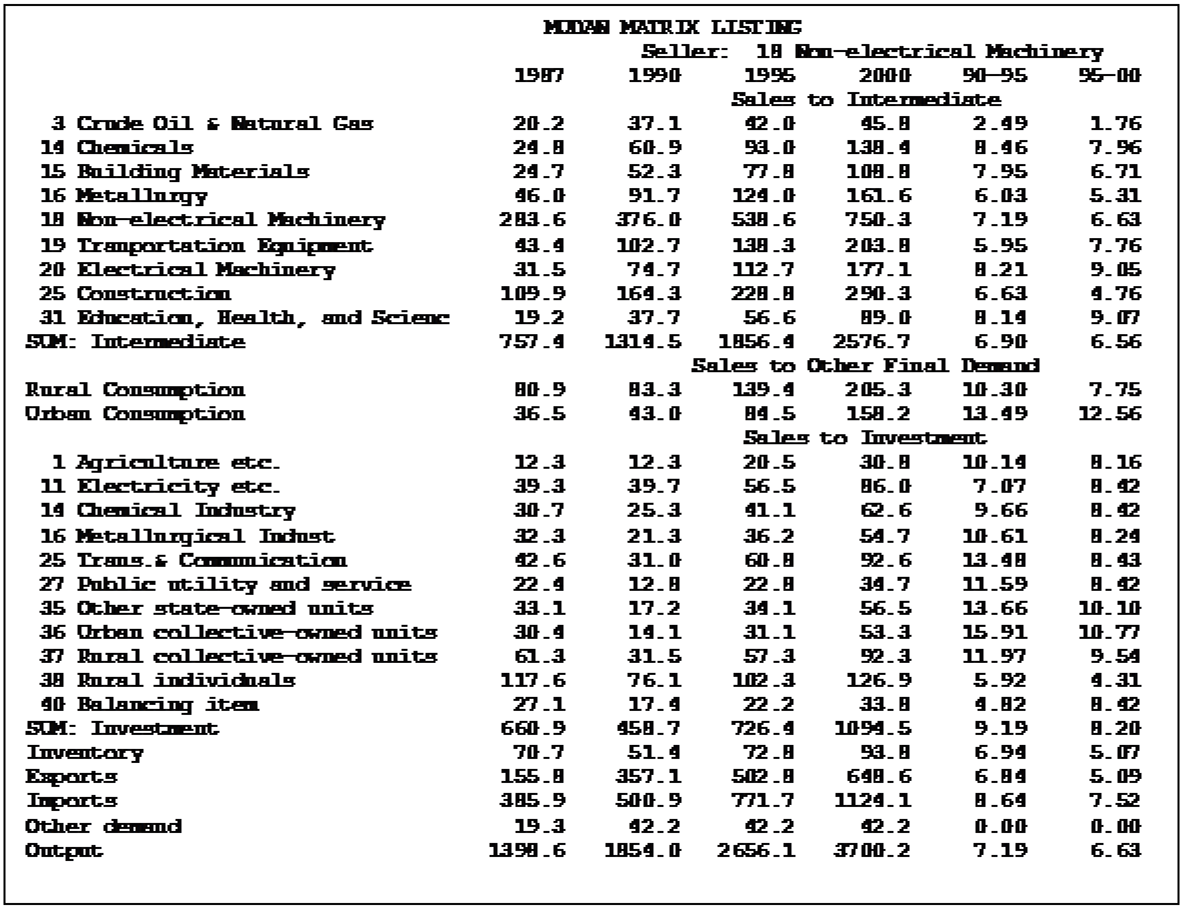
In this file, all lines beginning with a '#' are comments, and anything before a ';' on a line is also a comment. The first line that does not begin with a '#' must be the matrix listing identity. It specifies an input-output identity in terms of the particular model. Usually this identity says, in effect, total output = intermediate demand plus final demand, but other identites are possible. The identity must have a single vector on the left and then on the right an expression that is the sum of a number of terms. Each term may be either a single vector, like cr and cu in the example, or may be a matrix*vector product, such as am*out and bmv*capital. The terms should be joined by + or - signs. The matrix*vector terms result in a display of all flows obtained by multiplying each column of the matrix by the corresponding element of the vector.

Following the identity, the next non-comment line must provide the title file name for the vector on the left hand side of the identity. The file name must be enclosed in quotation marks (""). These title files can be the same one used with *G7* but *Compare* skips the 10-letter abbreviation at the beginning of each line which is used by the *G7 show* command. Rather, *Compare* looks for a string within " marks, and uses that. Other material may be in front of or after the quoted string. Here are the first few lines of the sectors.ttl file used in the above example.

```
Agricul  ;1 e "Agriculture"
Coal     ;2 e "Coal"
CrudeOil ;3 e "Crude Oil & Natural Gas"
```

There also must be file names for the titles of the columns of every matrix in a row listing or the row of every matrix in a column listing. In the example they are SECTORS.TTL and BMV.TTL. These files should be of the form as just explained with the titles between quotes.

Finally, for each term in the identity, MATLIST.CFG must show the name to be listed with the term. These term names also are shown as strings within quotes. For a vector term, this name will be listed on the left. For a matrix*vector term, it will be centered above the display of the cells of the matrix. The box below shows a sample of the matrix listing produced by our example.



```
                                        MUDAN MATRIX LISTING
                                               Seller:  18 Non-electrical Machinery
                                   1987     1990      1995      2000     90-95    95-00
                                                   Sales to Intermediate
   3 Crude Oil & Natural Gas       20.2     37.1      42.0      45.8      2.49     1.76
  14 Chemicals                     24.8     60.9      93.0     138.4      8.46     7.96
  15 Building Materials            24.7     52.3      77.8     108.8      7.95     6.71
  16 Metallurgy                    46.0     91.7     124.0     161.6      6.03     5.31
  18 Non-electrical Machinery     283.6    376.0     538.6     750.3      7.19     6.63
  19 Tranportation Equipment       43.4    102.7     138.3     203.8      5.95     7.76
  20 Electrical Machinery          31.5     74.7     112.7     177.1      8.21     9.05
  25 Construction                 109.9    164.3     228.8     290.3      6.63     4.76
  31 Education, Health, and Scienc 19.2     37.7      56.6      89.0      8.14     9.07
SUM: Intermediate                 757.4   1314.5    1856.4    2576.7      6.90     6.56
                                                 Sales to Other Final Demand
Rural Consumption                  80.9     83.3     139.4     205.3     10.30     7.75
Urban Consumption                  36.5     43.0      84.5     158.2     13.49    12.56
                                                    Sales to Investment
   1 Agriculture etc.              12.3     12.3      20.5      30.8     10.14     8.16
  11 Electricity etc.              39.3     39.7      56.5      86.0      7.07     8.42
  14 Chemical Industry             30.7     25.3      41.1      62.6      9.66     8.42
  16 Metallurgical Indust          32.3     21.3      36.2      54.7     10.61     8.24
  25 Trans.& Communication         42.6     31.0      60.8      92.6     13.48     8.43
  27 Public utility and service    22.4     12.8      22.8      34.7     11.59     8.42
  35 Other state-owned units       33.1     17.2      34.1      56.5     13.66    10.10
  36 Urban collective-owned units  30.4     14.1      31.1      53.3     15.91    10.77
  37 Rural collective-owned units  61.3     31.5      57.3      92.3     11.97     9.54
  38 Rural individuals            117.6     76.1     102.3     126.9      5.92     4.31
  40 Balancing item                27.1     17.4      22.2      33.8      4.82     8.42
SUM: Investment                   660.9    458.7     726.4    1094.5      9.19     8.20
Inventory                          70.7     51.4      72.8      93.8      6.94     5.07
Exports                           155.8    357.1     502.8     648.6      6.84     5.09
Imports                           385.9    500.9     771.7    1124.1      8.64     7.52
Other demand                       19.3     42.2      42.2      42.2      0.00     0.00
Output                           1398.6   1854.0    2656.1    3700.2      7.19     6.63
```

Using Dirfor Files with Compare

Although *Compare* primarily was developed for use with *G7* and *Interdyme*, it also can be used with the DIRFOR files that traditionally are created by the LIFT and *SLIMFORP* software. This section will illustrate what needed to be done to get the Dirfor files for the Japanese model into *Compare*, but any *SLIMFORP* model that uses a Dirfor file structure can be adapted for use with the *Compare* program.

The key to the Dirfor file structure is the DIRFOR.CRD file that is found in almost every *SLIMFORP* model (in LIFT, this is called DIRFOR.DAT). Those of you who work with LIFT or *SLIMFORP* already are familiar with this file, so it will not be described here in complete detail. However, there is a subtle difference in the format of the LIFT and *SLIMFORP* formats of this file. The *Compare* program works with the LIFT format as is, but minor changes must be made to the *SLIMFORP* versions.

```
daf\DIRHIS,
TOTAL NUMBER OF RECORDS
3064          /*    =  1 + 22*67 + 5*25 + 10*26 + 2*67 +10*67 + 1*400
NUMBER OF RECORDS PRECEDING EACH BLOCK    (IGN)
1     1475   1600   1860   1994   2664   3064
BLOCK    1 SERIES 22 SECTOR 67         IO LEVEL DATA
```

The DIRFOR.CRD for the Japanese model will be used in the example discussion below. The first change that needs to be made to the DIRFOR.CRD is in the section describing the blocks of the DIRFOR file. In the usual *SLIMFORP* format, these lines are as in the box below. The disadvantage of this traditional format is that, each time that the sector numbering or number of series in each block changes, these numbers must be hand edited, and new starting record numbers for each block must be calculated. In the LIFT format, these same lines would be written as follows:

```
daf\\DIRHIS,
 6 blocks
Sectors in block  67  25  26  67  67 400
Number of series  22   5  10   2  10   1
BLOCK    1 SERIES 22 SECTOR 67         IO LEVEL DATA
```

In this format, the number of blocks is given in the first line, in the format (I2), and the number of sectors and number of series in each block are given in the following two lines, in the format (16X,12I4). The LIFT version of Mkdirfor will calculate the starting record numbers of each block automatically, based on these data, and *Compare* does the same.

```
BLOCK    1 SERIES 22 SECTOR 67         IO LEVEL DATA
SERIES   1 RDFIL   1 RPOINT  1               CENTRAL GOVERNMENT
IHIS   1history\CEN5584.HIS,
SERIES   2         1         2          LOCAL GOVERNMENT
IHIS   1history\LOC5584.HIS,
SERIES   3         1         3          WATER & SANITATION
IHIS   1history\WAS5584.HIS,
```

The next difference is that each series in the new format has its own mnemonic, by which it can be referred to in making tables. The traditional DIRFOR series format is displayed in the box below. To change this to the new format, you need only to add mnemonics of up to four characters, starting in column 32:

```
BLOCK    1 SERIES 22 SECTOR 67         IO LEVEL DATA
SERIES   1 RDFIL   1 RPOINT  1  CEN          CENTRAL GOVERNMENT
IHIS   1history\CEN5584.HIS,
SERIES   2         1         2  LOC          LOCAL GOVERNMENT
IHIS   1history\LOC5584.HIS,
SERIES   3         1         3  WAS          WATER & SANITATION
IHIS   1history\WAS5584.HIS,
```

Once this is done, *Compare* can read the *SLIMFORP* file, using mnemonics such as cen1, loc21, etc. During the interactive running of the program, if you specify file type 'd' for Dirfor, you also will be asked to give the path of the DIRFOR.DAT file. You might want to keep a copy in the traditional format as DIRFOR.CRD and a copy in the new format as DIRFOR.DAT. Then you can use DIRFOR.DAT when running *Compare*.

CHAPTER SIX

BUILD MACRO-EQUATION PROCESSOR

Build pulls together equations estimated with *G7* into a model. It does so by writing a C++ program, HEART.CPP, which must then be compiled by a C++ compiler and linked with two object programs, RUN.OBJ and UTILITY.OBJ, to produce an executable file, RUN.EXE, that is a complete model that is ready to run. Basic documentation may be found in The Craft of Economic Modeling.

The HEART.CPP program now is written for the Borland C++ compiler. We have had success with every Borland compiler from Turbo C++ 2.0 to Borland C++ 5.5, which now is available for free. In case you have some other C++ compiler, the source code for the RUN.CPP and UTILITY.CPP files has been included, and you can recompile them with your compiler. Various other changes might need to be made in the make file or in headers, depending on your compiler. The output of *Build* is:

BWS.IND
BWS.BNK
A G data bank containing the historical values of all variables in the model.

HEART.CPP
The C++ program to compute the model. Now written for the the Borland C++ compiler.

RUN.LAG
A binary file necessary for the running of the model.

RUN.NAM
The names and some information about each variable in the model. In front of each variable name are three numbers. The first is the sequential number of the variable, the second is the maximum number of lags with with it occurs, and the third is the number of times that it is defined in the model. If this last is zero, the variable is exogenous. Watch out for variables that are defined more than once! While not necessarily an error, this condition can indicate that the same name has been used for more than one concept.

RUN.GR
A file for graphing all of the variables in the model. It also indicates whether the variable is exogenous or endogenous.

When *Build* starts, it reads the BUILD.CFG file to get its basic configuration. Read this file and ensure that it is appropriate for your setup. Similarly, RUN.EXE, the model itself, will read RUN.CFG.

Please visit the Software pages of the Inforum web site for more details and to download the *Build* software: http://inforumweb.inforumecon.com/software/software.html.

CHAPTER

SEVEN

BANKER : THE G-BANK MAKER

7.1 What is Banker?

Banker is a G data bank maker. It creates G data banks without going through *G7*. It particularly is suited for making large G data banks, since compressed *Banker*-generated data banks are NOT bounded by the rather limited number of series allowed with any *G7*-generated workspace bank. With compressed banks that are built by *Banker*, your G-bank literally can include millions of series and the size of your hard disk becomes the constraint.

Banker makes both hashed (.HBK, .HIN) and compressed (.CBK, .CIN) G banks. However, users strongly are encouraged to produce the hashed bank, as it now is the standard data bank that Inforum creates and maintains.

7.2 Usage

Syntax: banker [option] <data> [<bank>]

Options are one or more of the following characters preceded by "-" or "/". If no option specified, then hashed bank will be produced.

Examples:

```
banker lift.dat              :produces default bank (LIFT.HBK, LIFT.HIN)
banker -h lift.dat           :produces LIFT.HBK, LIFT.HIN
banker -b101 lift.dat          :makes the hashed bank with 101 bins
banker -d lift.dat           :make hashed bank, debugging only
banker -c lift.dat           :produces LIFT.CBK, LIFT.CIN
banker -cv lift.dat          : and displays series names while processing
banker -cvtinforum lift.dat  : and makes the bank_title "inforum"
banker -ctinforum-d lift.dat :debugging, with bank_title "inforum"
banker -cdtinforum lift.dat  :debugging, and bank_title = "inforum"
banker -cd lift.dat          :make compressed bank, debugging only
```

Note that normally only the first option switch is required to be preceded with '-' or '/', but there is a caveat with the -t switch. If it is chosen and a bank title is attached next to it, then the next chosen option must be preceded with '-' or '/'. If your bank title consists of more than one word, then connect these words with '+' so that there will not be any space in between the words. In light of this, you may decide to ignore -t switch altogether, in which case you will be prompted to provide a bank title once *Banker* starts to run. Yet another way to provide the bank title is to bury it in brackets "{}" and put it at the very beginning of your formatted data file. In a way, brackets "{}" indicate comments to *Banker*.

A file named BANKER.MSG automatically will be produced for each run of *Banker*. It contains information on data compression status for each series as well as some other related messages. The most commom messages on data compression includes:

- "No compression on xxx: FIRST DIFFERENCE TOO LARGE or all-zero series."
- "No compression on yyy: HUGE NUMBER ENCOUNTERED."

The first simply means that series XXX is an all-zero series or that its largest first difference is greater than 32767. Note that for *G*-style compression purposes, the first difference is taken AFTER all of the decimal places have been moved to the right until the series is all integers. This point also might help you to understand the second message, in which case the largest first difference so calculated exceeds LONG_MAX (or 2,147,483,647), a condition that we call "HUGE NUMBER ENCOUNTERED", even though the largest first difference may not exceed LONG_MAX (or 2,147,483,647) if we had not first slid the decimal to the right until the series was all integers.

7.3 How to Format Input Data for Banker

Banker handles most conventional *G7 data* and *matdat* input formats, which are assumed to be known to the user. There are detailed discussion in *G7* Help files and in the Appendix of Clopper Almon's The Craft of Economic Modeling.

Banker also accepts the one-series-per-line format, which initially was developed to process very large data banks more efficiently. The general idea of this format is to stock all of the information about a series on one line (record), including, of course, all of its observations.

Here is an example of the one-series-per-line format:

```
gdp$ 1985 Q 1 0 5025 5130.5 5.255e3 ...
```

where

gdp$
series_name

1985
baseyear (can also be written as 85 [deprecated])

Q
frequency (M = Monthly, Q = Quarterly, A = Annual)

1
starting_period

0
decimal point left-shift factor

5025
observation #1

5130.5
observation #2

5.255e3
observation #3

All seems clear except, perhaps, the fifth item under this format – the decimal point left-shift factor. Thus some explanation (or justification for it) is in order. Simply put, this left-shift factor enables you to manipulate (add) the decimal places in your data. For example, BLS data sometimes requires some additional decimal places, and this left-shift factor can help you to do just that. If, on the other hand, there is no need to modify decimal places in your data, then use zero as the left-shift factor.

Please take note that each item must be separated by at least one space (as shown in the example above).

It should be mentioned in passing that *Banker* will keep intact the series names exactly as in the formatted input file. That is, if a series name (or part of it) is in upper (lower) case, it will remain that way in the bank created by *Banker*.

Lastly, *Banker* allows *G7*-style commenting.

Examples:

```
# This is a one line comment.
: This comment, however, is longer.
Clearly, it is a much more important comment,
For it takes three lines.
: Or four.
```

7.4 Notes for Programmers

As indicated earlier, *Banker* can make both hashed and compressed G banks. In the compressed bank, each series has its own starting date and number of observations and most series have been compressed. Compression involves these steps:

1. Find and record the number of decimal points in the series.
2. Slide the decimal to the right until the series is all integers.
3. Record the first observation as a 4-byte integer.
4. Record the first differences of the series as 2-byte integers.
5. If a series cannot accurately be recorded in this compressed form, it is declared to have 255 decimal places (just a flag), and the observations are recorded as four-byte floating point numbers.
6. Missing observations are marked with a special code.
7. The data file has the extension "CBK"; and the index, "CIN".

The .CIN file contains:

Item	Size in Bytes	C type	Definition
ns	2	int	The number of series.
nc	2	unsigned int	The cumulative number of character in the series names, counting the nulls at the end of each series name.
names	nc	char	The series names.

If there were three variables in the bank with names tom, dick, and harry, *ns* would be 3, and the names vector would be

```
tom0dick0harry0
```

where 0 represents a null ('0' in C), and *nc* would be 15, the number of characters in the names vector, counting the null characters.

The .CBK file contains:

Item	Size in Bytes	C type	Definition
title	80	char	The bank title.
ns	2	int	The number of series
position	4	unsigned long	The byte number at which this “indx” array begins.
series 1	variable	see below	
. . .		. . .	
series n	variable	see below	
indx	4*ns	unsigned long	An array containing the byte numbers at which the series begin in this file.

To continue the example, suppose that the series “tom” requires 101 bytes, “dick” requires 121 bytes, and “harry” requires 81. Then “indx” is the vector (86, 187, 308). (Remember that in C a file starts with byte 0). The “position”, which is the byte number at which this “indx” array begins, will be 389.

Each compressed series has the format:

Item	Size in Bytes	C Type	Definition
BaseY	1	unsigned char	The year of the first observation, minus 1900.
FreqPeriod	1	unsigned char	16*frequency+period, where frequency is the number of observations per year (1, 4, or 12), period is the period of first observation. (For frequencies above 12, set FreqPeriod = 255. This value signals that two integers have been inserted after this byte containing the frequency and the period.)
SlashD places	1	unsigned char	16*SlashFactor + Number of Decimal places, where SlashDecplaces usually is just the number of decimal places. If it is 255, the series has not been compressed.
NDif	2	int	The number of differences, equal to the Number of observations - 1.
FirstO	4	long	The first observation, as a four-byte integer.
Differences	2*ND	int	The first differences of the series, as 2-byte integers.

If a zero occurs in the series, then it is indicated by 32767 in the differences. The following difference applies to the previous non-zero number, not to the zero. This practice was adopted because some banks have series with numerous missing observations that appear as zeroes. Also, some banks consider quarterly series to be monthly series in which only the end-of-quarter months have non-zero values. Note that SlashFactor normally is 0; the Press program, however, allows the option of dividing by a power of 2 to reduce the magnitudes of a series so that it can be compressed. The SlashFactor is the power (1, 2, 3, etc.) to be used on this series. In Press, the default maximum slash factor is 0, so the occurrence of non-zero slash factors is unusual.

If it was not possible to compress the series, then the format is:

Item	Size in Bytes	C type
BaseYear	1	unsigned char
FreqPeriod	1	unsigned char
255	1	unsigned char
nobs	2	int
Observations	4*nobs	float

Note that the 255 in the third byte is the signal that the series is not compressed. The next two bytes represent the number of observations, and then the observations follow as 4-byte floating point numbers.

The compressed form can represent a series as accurately as can an 18-foot-high graph printed with laser-printer resolution of 300 dots per inch. (All series in the US National Accounts or Industrial Production Indexes compress easily. In the Blue Pages of the Survey of Current Business, however, nearly ten percent fail to compress. In IMF data, the hyperinflation of many third-world countries produces series which fail to compress.)

Hashed banks differ from compressed banks mainly in the organization of their index files. With standard and compressed banks, *G7* keeps the names of the series in memory and simply does a linear search for a name each time one is requested. In the hashed banks, the names instead are grouped into bins on the basis of a number calculated from the letters of the name. When a name is requested, *G7* calculates the number, locates the bin in which the name has been stored, reads in the names in that bin, and does a linear search over only those names to find the desired series. The size of compressed banks is limited by the requirement that the total number of characters in the names of all series must be less than 64,000. In practice, that limit typically translates to about six or seven thousand series. Hashed banks, in contrast, can go up to several million series. The data file has the extension “HBK” and the extension for the index is “HIN”.

The precise form of the hashed bank .HIN and .HBK files are presented below. The “.HIN” file contains:

Item	Size in Bytes	C type	Definition
ns	4	long	The number of series in the bank.
nbins	2	unsigned	The number of bins in the bank.
nsb	2*nbins	unsigned	An array to contain the number of series in each bin.
ncharb	2*nbins	unsigned	The cumulative number of characters of the series names (including each ‘0’) contained in each bin.
posbin	4*nbins	unsigned	The beginning positions in the “.HIN” file of the first bytes of the binname() strings.
binname(0)	nchar[0]	char	The string binname(i) denotes the concatination (including the 0’s) of all the series names in the i-th bin.
binposts(0)	4*nnmsb[0]	long	binposts(i) is an array of beginning positions in the associated “.HBK” of the series in the the i-th bin.
binname(1)	nchar[1]	char	
binposts(1)	4*nnmsb[1]	long	
binname(2)	nchar[2]	char	
binposts(2)	4*nnmsb[2]	long	
. . .			
binname(nbi 1)	nchar[nbins-1]	char	
binposts(nbi 1)	4*nnmsb[nbi 1]	long	

The series are separated into *nbins* “bins”, where the number contained in each bin is recorded in the *nsb* array. Of course, the ordering of the series in the binname() and binposts() arrays must be the same.

Consider an example. Suppose that the third bin contains the series “joe”, “dave”, and “bill”. The string binname(3) would be

```
"joe\0dave\0bill\0"
```

Suppose that the starting positions in the “.HBK” bank for the three series are 40700008, 490987, and 3378294. The array binposts(3) then would be [40700008, 490987, 3378294], with nsb[3] = 3, and with ncharb[3] = 14. If the beginning position of binname(3) in the “.HIN” file is 4724, then posbin[3] = 4724.

To assign a bin number to a series you must use the following hashing routine. In C, the routine is:

```
unsigned hash(char *s);
hash(char *s){
  unsigned bill;
  for( bill=0; *s!='\0'; s++ ) bill = *s + 31*bill;
  bill = bill%nbins;
  return(bill);
  }
```

To continue with the example, to determine the bin which the series “joe” really belongs to you’d evaluate the function hash(“joe”).

The .HBK file layout is:

Byte	Type	Description
0 - 79	char	Name of bank (terminated with a null)
80 - 81	int	ns, number of series in the bank
82 - 85	long	psn, position in file of index
86 -		first series, as described below
*(psn+1) -		second series,
. . .		. . .
psn	long	position in file of first byte of first series
psn+4	long	position in file of first byte of second series
. . .		. . . on out to ns series

For each series, the format is:

Byte	Content
0	base year
1	frequency*16+period
2	slash*16+maxplaces or 255 if not compressed
3-4	number of observations
5-8	first observation as a long
9 -	differences as integers

If the series is not compressed, then floating-point data begin in byte 5.

CHAPTER

EIGHT

FIXER AND *MACFIXER* PROGRAMS

8.1 *MacFixer*: Fixing Macro Variables in *InterDyme* Models

Macro variable fixes apply to variables of type Tseries; these are defined using the *Idbuild* program described elsewhere. These fixes work like those of models built with the *G7-Build* combination, but they also have much in common with the vector fixes described in the next section. The program that handles the macro variable fixes is called *MacFixer*. The input to *MacFixer* is a file prepared by the user with a text editor. It should have the extension .MFX. Once this file has been created, the program *MacFixer* is run by selecting Model | *MacFixer* on the *G7* main menu. The results of this program are written to a "macro fix bank", which essentially is a *G* workspace bank (which can be read with *G7*). The root name (the part of the filename before the dot) of the macro fix *G* bank is passed to *MacFixer* through the form that the above *G7* command opens. It also must be passed to the simulation program by the form that opens on the command Model | Run Dyme.

MacFixer requires a configuration file, called MACFIXER.CFG. It is created by *G7* from the information provided on the form that is opened by the Model | *MacFixer* command. This form requires the name of the text input file, the root name of the *G* bank file used for base values for the index and growth-rate fixes (this normally would be the *G* bank created for use with the simulation program), the name of the *G* bank which will contain the values of the fixes, and the name of the output check file. This last file shows the values of each fix in each year, and serves as a check on the results in the binary file.

While it is up to the user to name files, it is good practice to give the same root name to files for the same simulation. A simulation that involves low defense expenditures, for example, could have a *G* bank file called LOWDEF.BNK and a .MFX file called LOWDEF.MFX.

There are several varieties of macrofixes that may be given, and they are described in the list below.

skip

This is the simplest type of fix. It simply skips the equation and uses the values in the model *G* bank. For example:

```
skip invn$35
```

would skip the equation for the macro variable invn$35 and instead use the value already in the model *G* bank.

ovr

This overrides the result of the equation with the value of the time series given. Values between given years are interpolated linearly. In the example below, the macro fix program would calculate and override a fix series that starts in 1992, ends in 2000, and moves in a straight line between the two points. For example,

```
ovr uincome$
1992  154.1
2000  182.3
```

would override the value of the forecast of uincome$ with the values shown for the years shown. Note that the specified years either can be presented in 2-digit format (deprecated) or 4-digit format (preferred).

mul

This multiplies the equation's forecast by a factor specified by the data series on the following line. For example,

```
mul ulfi$
1992  1.00
1995  1.05
2000  1.10
```

multiplies the forecast results for the macrovariable ulfi$ by the factors shown. Values of the multiplicative fix between the specified years shown are interpolated linearly.

cta

This performs a constant term adjustment. That is, it adds or subtracts the value of the time series to the result of the equation. The time series is provided by the fix definition. For example,

```
cta nonagincome
1992    0.0001
1995  200
2000  180
```

is a constant term adjustment for nonagricultural income from 1992 to 2000. Intermediate values are interpolated.

ind
dind

This is a variety of the override fix that specifies the time series as an index. There must be data in the bank for the item being fixed up until at least the first year of the index series specified. The value for the item in that year then is moved by the index of the time series given by the fix lines.

For example,

```
ind wag01
1982  1.00  1.03  1.08  1.12  1.15
1997  1.21  1.29  1.31  1.34
```

will move the value of wag01 in 1982 forward by the rate of change of the series given, and it will replace the calculated value of wag01 by this value when the model is run.

The *dind* version of the index fix is the "dynamic index fix". This fix can start in any year and does not rely on historical data being present in the databank. Rather, the fix calculation is based on the value of expression during the model solution for the first year of the fix.

gro
dgro

This is a type of override fix that specifies the time series by growth rates. For the growth rate fix to be legal, there must be data in the vam file up until at least the year before the first year of the growth rate fix. Missing values of the growth rates are interpolated linearly.

For example,

```
gro wag01
1993  3.1
2000  3.4
```

The *dgro* version is the "dynamic growth rate fix". This fix can start in any year and does not require data to be available in the databank for the starting year of the fix. The growth rate always is applied to the value of the variable in the previous period.

stp
dstp

This is a step-growth fix. It is like *gro* except that a given growth rate is applied until a new one is specified. A value for the final period is necessary.

For example,

```
stp wag01
1993  4.1
1995  4.5
2000  5.0
```

The *dstp* version is analogous to the *dgro* fix; the only differences is the way in which values of the fix are interpolated.

rho <depvar> <rho_value> <rho_set_date>

This is a rho-adjustment fix where

rho_value
is the value of rho.

rho_set_date
is the year in which the rho adjustment error is to be calculated. If none is provided, it is set in the first year of the run.

This type of fix finds the error made by an equation in the last year for which there is historical data. In the next year, it multiplies this error by the given <rho_value> and adds the result to the value forecast by the equation. In the following year, it multiplies what was added in the first year again by <rho_value> and adds the result to the equation's forecast, and so on.

For example:

```
rho invn$38 0.40 1995
```

tells the model to apply a rho adjustment to the variable invn$38, using the value 0.40 for rho and starting the rho adjustment in 1995.

A rho fix with a <rho_set_date> works like a *skip* fix in years before the <rho_set_date>. A variable can have a *rho* fix in conjunction with and a *cta*, *mul*, *ind*, or other type fix. The rho adjustment is applied before any other other fixes.

eqn <macroname> = <expression>
<year> <value> [<value> <value> . . .]

This an equation fix where:

<macroname>
is the name of a macrovariable

<expression>
is a legitimate expression, as described below

<year> <value>
entries are in the same format as the data for other fixes, but these indicate the years for which the equation fix is to take effect. They also represent the time series for a special variable called "fixval", which can be used within the equation expression. This "fixval" variable can be used wherever a vector element or macrovariable could be used.

This type of fix lets you dynamically introduce a new equation relationship into the model at run time. The advantage of this type of fix is that users of the model who are not programmers can introduce their own assumed relationships into the model without having to change the model program code. It also is helpful for prototyping a model, where you want to try different equation relationships quickly to see how they work before changing model code.

The equation fixes use the same expression syntax as used in the *f* command and other commands in *G7*. Most expressions that are legal in *G7* are legal for an equation fix, but only a subset of functions are implemented. Legal functions are: *@cum*, *@peak*, *@log*, *@exp*, *@sq*, *@sqrt*, *@pow*, *@fabs*, *@sin*, *@pct*, *@pos*, *@ifpos*, *@pct*, *@rand*, and *@round*.

Lagged values of any order can be used, with the constraint that they must not refer to a date before the starting year of the model *G* bank (DYME.BNK). Macrovariables are read directly from memory. Lagged values of vector variables are read from the Vam file. Therefore, you can use a lagged value of any vector element as far back as the starting date of the Vam file, and you are not limited by whether or not that vector has been declared to store lagged values in memory in VAM.CFG.

Examples:

```
# Make the T bill rate equal to the average inflation plus some percent,
#   specified in "fixval".
eqn rtb = 0.34*gnpinf + 0.33*gnpinf[1] + 0.33*gnpinf[2] + fixval
1998 1.0
2010 1.5;
...
```

fol <Macroname> = <expression>
<year> <value> [<value> <value> . . .]

This is a follow fix. The follow fix allows you to specify that a macrovariable should move like some other quantity that may be specified as a general expression involving vector elements and macrovariables, just like the equation fix. The variable "fixval" should not be used in the follow fix expression. Its purpose of the specified time series values is to specify a growth rate to add to the growth of the expression.

For example, if we would like to specify that Medicaid transfer payments grow like real disposable income per capita plus 0.1 per cent, we could write:

```
fol trhpmi = di87/pt
1997 0.1
2010 0.1
```

shr <Macroname> = <expression>
<year> <value> [<value> <value> . . .]

This is a share fix. This fixed is used to specify that the macrovariable should be a certain share of another variable or expression, with the share specified by the fix value. The *share* is just a multiplier, so it can be any number. In the share fix, the fix value is the multiplier or share to multiply by the right hand side expression.

When the MACFIXER.CFG file and the input file as described above are ready, type "macfixer" at the DOS prompt to invoke the program *MacFixer*. When the model is running, calls to the "modify" function will apply the fixes using the information in the macro fix *G* bank specified in the DYME.CFG file for that run. Note that to view the fixes in the macro fix databank, load the bank in *G7* and specify the series name as the name of the macro variable, followed by a colon (':'), followed by a one-letter code signifying the type of fix. These codes are as follows: skip ('k'), ovr ('o'), cta ('c'), ind ('i'), gro ('g'), stp ('s'), and rho ('r'). Therefore, to view a *cta* fix on the variable invn$38, issue the following command in *G7*

```
ty invn$38:c
```

Macro fixes provide an alternative way to supply values of exogenous variables. Exogenous variables, to review, should be put into the "hist" bank in the process of running *IdBuild*. If the variable appears in no .SAV file for a macro equation, then it must be included in the PSEUDO.SAV file. The standard way of providing the values of the exogenous variables then is through *update* or other commands in Vam. Another possibility for providing exogenous values is to have a special run of *G7* with the "hist" or other bank as the workspace bank. Finally, one can provide the exogenous values as macrofixes. For example, if we want disinc to be an exogenous variable, then – however we are going to provide the values – we need the statement

```
f disinc = disinc
```

in the "PSEUDO.SAV" file. To use the macrofix method of assigning values, we need in the code of the model the statements

```
depend=disinc[t];
disinc[t] = disinc.modify(depend);
```

We then could provide the values with *ovr*, *ind*, *gro*, or *stp* commands to the *MacroFix* program, for example, by

```
gro disinc
1995 3.0
2000 3.5
2005 4.0;
```

This method has the advantage of keeping all of the fixes which constitute a scenario in one place. It also allows the use of the *gro* and *stp* fixes, which may be convenient. It has the disadvantage of adding an additional series to the banks which constitute the model and an additional statement within the model.

Please visit the Software pages of the Inforum web site for more details and to download the *MacFixer* software: inforumweb.inforumecon.com/software/software.html.

8.2 *Fixer*: Fixing General Vectors and Matrices in *InterDyme* Models

Vector fixes are more complicated than macro fixes because they can apply to individual elements of a vector, to the sum of a group of elements, or to the sum of all elements in the vector. However, the format of the vector fixes is very similar to that of the macro variable fixes. Matrix fixes at present remain rather simple, with each fix being applicable to only one cell of a matrix. The preparation of the vector and matrix fixes is the work of the *Fixer* program. (*Fixer* sometimes is called *VecFixer*.)

G7, it should be noted, normally prepares vectors of exogenous variables; fixes apply to vectors of endogenous variables. However, the *Fixer* program also can be used to supply the values of exogenous variables. Also, when building a model, before all of the equations are finished, *Fixer* can be used to project the values of right hand side variables of some of the unfinished equations.

When and how are fixes applied as a model runs? Unlike the macro fixes, which automatically are applied when a macro regression equation is calculated, vector and matrix fixes are applied when and where the model builder specifies. At the point where the fixes for the vector "x" should be applied, the model builder must put into the program the line

```
x.fix(t);
```

The input to *Fixer* is a file prepared by the user in a text editor. It should have the extension .VFX . *Fixer* also reads the definitions of static groups of sectors and writes them into the GROUPS.BIN file which can be used both by the simulation program and by *G7*. To use the *Fixer* program, it is essential that the model's VAM.CFG file should have a vector called "fix" with enough rows to allow one element for each fix. As *Fixer* reads the fixes from the input file, it stores the numerical values of the fixes into this "fix" vector within the vam file. It also creates a "fix index" file, which will have the extension .FIN and tells the simulation what to do with each fix. Finally, it produces a binary file with the definitions of groups, called GROUPS.BIN. If *G7* already has produced a GROUPS.BIN file, then *Fixer* will read it and might add to it.

Fixer is started by the command Model | VexFixer on the *G7* main menu. From the information on the form which this command creates, *G7* prepares a configuration file, called FIXER.CFG, that is read by the *Fixer* program. This form specifies the root names of:

- The text input file
- The fix index file
- The vam file used for base values for the index and growth rate fixes
- The name of the output check file, which will show the sectors in each group and the values of each fix in each year. It is used only for manual checking of the program.

While it is up to the user to name files, it is good practice to give the same "root" name to files in the same simulation. A simulation that involves low defense expenditures, for example, could have a vam file called LOWDEF.VAM and a .VFX file called LOWDEF.VFX.

Vectors fixes may apply to a single element or to a group of elements. The concept of a "group", also described in the *G7* documentation, is central to the working of *Fixer*. A group basically is a set of integers, usually representing sectors in the model. Groups are useful because we often want to impose a fix on a group of elements in a vector. For example, we might want to control the total exports of the chemical manufacturing sectors. We then might create a group named "chem" which would contain the sector numbers of all sectors in question. The command for defining a group is "grp <groupname>", where the group name can be a number or a name. The sectors included in the group are then specified on the next line. For example,

```
grp 1
7 10 12
```

creates a group called 1 of the sectors 7, 10, and 12. The "-" sign means consecutive inclusion. Thus

```
group zwanzig
1 - 20
```

consists of the first twenty integers. Parentheses mean exclusion. Thus

```
group duo
:zwanzig (2 - 19)
```

makes the group “duo” consist of the integers 1 and 20.

When a group is referenced after it has been defined, a colon must precede its name, as shown when “zwanzig” was used in the definition of “duo” above. Names of groups are case sensitive. Groups need not be kept in numerical order and can be defined anywhere in the input file, so long as the group is defined before it first is employed. If you try to redefine an existing group, the program will complain unless the new group has the same or less than the number of elements as in the old group. References to other groups can be used in new group definitions only if the groups referenced already have been defined.

The way that most fixes are applied to groups is that the value of the fix is calculated first, and then the elements of the group are scaled to the fix control total using right direction scaling. For certain types of variables, such as price indexes, wage indexes, or productivity indexes, scaling to such control totals is not sensible.

For growth fixes (*gro*, *stp*, *dgro*, *dstp*), index fixes (*ind*, *dind*), and follow fixes (*fol*), the *-s* option is available. It applies the same growth rate, index, or follow pattern to each element of the group separately. The “-s” comes after the fix type and before the vector name. For example:

```
# Foreign price fixes
group impprice
 59,61,62,66,69,72,77-80,91

 # "fol -s" means that each sector in the group should follow
 # the variable forpi.
 fol -s fpi :impprice = forpi
  2013 0.0
  2040 0.0
```

In this example, the group *impprice* is defined to be a list of sectors that do not have exogenous foreign prices. They each should be indexed to move like the aggregate foreign price index *forpi*. The 0’s in the body of the fix have two purposes: they indicate that the fix is to take effect from 2013 to 2040 and that the indexes should move exactly like *forpi*, with no extra growth added or subtracted.

Interdyme provides a number of ways for a fix to work. In all of them, a time series is specified by the fix definition. The forms of the fixes differ in how they obtain and in how they apply this time series. The basic format of the input file for a vector fix is:

<command> <vectorname> <GroupOrSector>

followed on the next line by the year and value of the fix. The basic format of the input file for a matrix fix is:

<command> <matrixname> <row> <col>

Definitions of the legal commands and examples follow.

ovr

This overrides the result of the equations with the value of the time series given. Intermediate values are interpolated linearly. In the example below, the fix program would calculate and override the model predicitions using the fix series that starts in 1992, ends in 2000, and moves in a straight line between the two points:

```
ovr ex 10
1992  154.1
2000  182.3
```

overrides the value of the forecast of element 10 of the "ex" vector (probably exports) with the values shown for the years shown. Note that year either can be specified in 2-digit format (deprecated) or 4-digit format (preferred). As an example of a matrix fix,

```
ovr am 1 9
1990 0.23
1995 0.26
2000 0.28
```

would override the value of the A-matrix in the Vam file for the element in row one, column nine, from 1990 to 2000. As before, missing values are interpolated linearly.

mul

This multiplies the equation forecast by a factor specified by the data series on the following line. For example,

```
mul im 44
1992  1.00
1995  1.05
2000  1.10
```

multiplies the forecast results for imports of sector 44 by the factors shown. Values of the multiplicative fix on imports between the years shown are interpolated linearly.

cta

This performs a constant term adjustment. It adds or subtracts the value of the time series to the result of the equations. The time series is provided by the fix definition.

For example,

```
cta def :Alice
1992    0.0001
1995  200
2000  180
```

is a constant term adjustment for defense expenditures of all sectors in the Alice group. Intermediate values are interpolated.

ind

dind

This is a variety of the override fix that specifies the time series as an index. There must be data in the vam file for the item to be fixed up until at least the first year of the specified index series. The value for the item in that year then is moved by the index of the time series given by the fix lines.

For example,

```
ind pceio :zwanzig
1982  1.00  1.03  1.08  1.12  1.15
1997  1.21  1.29  1.31  1.34
```

will calculate the sum of the elements of the pceio vector included in the group “zwanzig” in 1982, will move that sum forward by the index of the series given, and will impose that total to control the those elements when the model is run.

The *dind* version of the fix can start in any year, and it indexes the series to the value of the expression in the starting year of the fix.

gro
dgro

This is a type of override fix that specifies the time series by growth rates. For the growth rate fix to be legal, there must be data in the vam file up until at least the year before the first year of the growth rate fix. Missing values of the growth rates will be replaced by interpolated values.

For example,

```
gro out 10
1983  3.1
2000  3.4;
```

The *dgro* version of the growth rate fix can start in any year and always calculates the series in the present period based on the value in the previous period.

stp
dstp

This is a step-growth fix. It is like *gro* except that a growth rate continues until a new one is provided. A value for the final period is necessary.

For example,

```
stp out 1
83  4.1
95  4.5
2000 5.0;
```

The *dstp* version is the dynamic version which can start in any year. It is just like *dgro*, except for the method of interpolation of the fix values.

eqn

The equation fix for vectors works in the same way as the version for macrovariables, with the exception that the name of the vector must be separated from the sector number by a space.

For example:

```
# Make the pce deflator for category 3 grow like the aggregate PCE deflator,
# based on the ratio in 1997, from 1998 to 2010.
eqn cprices 3 = cprices3{1997}/apc{1997} * cprices3
 1998 1
```

(continues on next page)

(continued from previous page)

```
 2010 1

# Make corporate profits in sector 1 remain a constant share of total corporate
#   profits, equal to the share in 1997:
eqn cpr 1 = cpr1{1997}/vcpr{1997} * vcpr
 1998 1
 2010 1
...
```

fol

The follow fix specifies that an element or group of a certain vector should follow the expression on the right, plus or minus a certain growth rate that can be specified in the body of the fix. It often is used to make imports of a certain commodity grow like domestic demand. For example, the following follow fix makes crude petroleum imports grow like domestic demand, plus 0.2 percent per year:

```
fol im 4 = dd4
1998 0.2
2030 0.2
```

shr

The share fix takes the value of the body of the fix ("fixval") and multiplies the right-hand expression by it before assigning the value to the left hand side variable or group. Like the follow fix, a typical use for this fix is to control the relation between imports and domestic demand. The example below specifies the share of domestic demand for imports of Radio, Television, and Video equipment:

```
shr im 42 = dd42
1998 0.90
2000 0.92
2030 1.00
```

When the the FIXER.CFG and input file as described above are ready, type "fixer" at the DOS prompt to invoke the *Fixer* program.

Please visit the Software pages of the Inforum web site for more details and to download the *Fixer* software: http://inforumweb.inforumecon.com/software/software.html.

Fixes, as used here, are ways to make a model work the way we want it to work, not necessarily the way that emerges from its equations. The power over a model that fixes provide certainly can be, and often has been, abused. Nonetheless, they have a legitimate role to play. Suppose, for example, we wish to consider the impacts of some event that the equations never dreamed of, like a natural disaster or a massive overhaul of the health care system. A fix then is the natural way to convey to the model that the equations are not to be trusted entirely.

Interdyme has three types of fixes:

- those for macro variables
- those for vectors and matrices
- and a special type for industry output.

CHAPTER

NINE

IDBUILD MACRO-EQUATION PROCESSOR FOR INTERDYME

IdBuild is an adaptation of the *Build* program, which is used for aggregate model building, to serve the construction of Interindustry Dynamic models. Like *Build*, it translates the .SAV files created by *G7* into C++ code, builds a bank of all the scalar variables, and writes several files of C++ code for use in the simulation program. Also like *Build*, it requires a BUILD.CFG file that specifies the name of the output (or workspace) bank and the initially-assigned bank. Like *Build*, it also requires a Master file, but this file merely lists each .SAV to be included in the model, where each .SAV file name is given with the command *iadd*. Unlike *Build*, the Master file should not contain identities or other code. *IdBuild* is invoked from *G7* by the command Model | *IdBuild*. This command also proceeds the compilation and linking of the model. (From DOS, IdBuild can be invoked by “idbuild master”.) The output files made by *IdBuild* are:

HIST.BNK
HIST.IND
A standard *G* bank that contains the series used or created by *IdBuild*. As in *Build*, variables on the right of *fex* commands are not included.

HEART.CPP
A compilable C++ program. Each of the *iadd* files becomes a separate, callable function with a name derived from the name of the *iadd* file from which it was created. It also contains a function (tserin) to read in all the time series from bws and make them accessible everywhere in the forecasting program. Both past and preliminary future values are available at all times.

HEART.H
Prototypes for the functions in the HEART.CPP file. This file makes it possible to call these functions from anywhere in the forecasting program.

TSERIES.INC
A file to be included in the main module of the forecasting program to declare the names of the variables in HIST.BNK as variables that can be used in the program.

CALLALL.CPP
A program to call all of the functions in HEART.CPP. It is used at the end of each year’s calculations to set or update rho adjustment factors.

The Master File for IdBuild

We will illustrate the forming of the master file with the Mudan model of China. The file for this model is:

```
bank cmdm
iadd invest.sav
iadd income.sav
iadd finance.sav
iadd pseudo.sav
q
```

Except for PSEUDO.SAV, the various .SAV files contain estimated equations. For example, INVEST.SAV begins

```
title Other State owned units investment
f sinvest = sibac$ + sirep$
r invn$35 = 40.346479*intercept + 0.110044*sinvest
```

The program recognizes PSEUDO.SAV as the name of a file with a special purpose. Namely, it puts into the model's *G* bank those time series that are not needed in any of the code-image equations but are required elsewhere, perhaps in identities or detached coefficient equations. For Mudan, the beginning and end of the PSEUDO.SAV file are:

```
f trsa = trsa
f rpop = rpop
f upop = upop
...
f rscale = 1.0
f uscale = 1.0
```

These commands put the variables named trsa, rpop (rural population), and upop (urban population) into the model's *G* bank. At the end, it initializes the rscale and uscale variables to 1.0 in all years. Most right-hand side expressions that are legal in *G7* also would be legal here.

In the master file, the final *q* signals the end of input to *IdBuild*. *IdBuild* will produce from this command the following heart.cpp file:

```
#include <stdio.h>
#include "constant.h"
#include "matrix.h"
#include "dyme.h"
#include "groups.h"
#include "databank.h"
#include "vamfile.h"
#include "fixbank.h"
#include "dyme.ext"
#include "heart.h"
#include "tseries.ext"
FILE *fmatrix;
int i,j,k,err;
extern int t;
float depend;
/* end of standard prolog */

void investf(){
   /*  Other State owned units investment */
   sinvest[t]= sibac_[t]+ sirep_[t];
   /* invn$35 */ depend = 40.346479+0.110044* sinvest[t];
   invn_35.modify(depend);
   ...
   }
void incomef(){
   ...
   }
void financef(){
```

(continues on next page)

(continued from previous page)

```
   ...
   }
void tserin(){
   sibac_.in("sibac$");
   sirep_.in("sirep$");
   sinvest.in("sinvest");
   ...
   }
```

Note that all of the *iadd* files have been turned into functions whose names are formed by adding an ‘f’ to the end of the *iadd* file name. (Any $’s in the file name have been turned to _’s; there are none in the example.) What were *f* commands have become C statements with the exception that the ‘$’ character in variable names has been changed to an ‘_’. Examples of this change are seen in both the investf() function and the tserin() function. Any *fex* commands (none in the example) have disappeared, but the variable on the left of the *fex* has been created and entered into the data bank. Regression equations appear with the regression coefficients in the code. They calculate a variable, “depend”, which is passed to the routine *modify()*, along with the identification number of the dependent variable. The *modify()* routine then looks to see if there are any macro variable fixes – add, multiply, index, growth, skip, or rho adjustment – on that variable, and then stores the variable with the appropriate modification, if any. At the end of the HEART.CPP file is the *tserin()* function. It is called at the beginning of a run of the model to read into memory all the time-series variables. Finally, note that it contains all the variables which are in the PSEUDO.SAV file, even though no function was created by this file. This is the way to put into the data bank those variables, such as labfor, that appear in no code-image equation. The name “pseudo” is a keyword for the program; files by any other name create functions.

The HEART.H file created by *IdBuild* for this example is:

```
void investf();
void incomef();
void financef();
void exdgf();
```

It simply provides the prototypes required by C++ in any program that uses the functions in the HEART.CPP file.

The TSERIES.INC file is:

```
Tseries sibac_, sirep_, sinvest, invn_35, invn_36, d88, d90, invn_37,
rni, rpindex, rpop, rincome_, trsa, trsa_, invn_38, ulfi,
...
uscale;
```

These files are “#included” in the forecasting program to declare that sibac_, sirep_, etc. are objects of the type “Tseries”. A “Tseries” is an object defined in the forecasting program that is designed to hold a time series. One of the things that a Tseries object “knows” how to do is to load itself from the assigned *G* data bank. Thus, in the *tserin()* function in the HEART.CPP program above, the command

```
ngdpc.in("ngdpc");
```

tells the ngdpc object to read in its data contents from the series called “ngdpc” in the data bank.

The final product of *IdBuild* is the CALLALL.CPP file, which for our example simply is:

```
#include "heart.h"
void callall(){
   investf();
   incomef();
   financef();
   exdgf();
   }
```

As its name suggests, *callall()* simply is a program to call all of the functions in the HEART.CPP file. Its function is in connection with rho adjustments, as will be seen in the forecasting program.

Combining Vector and Tseries Variables in a Function With IdBuild

When building interindustry macro models one usually needs to integrate the macro and the industry computations. For example, it often is necessary to form a macrovariable as a sum of components of a vector. Conversely, it may be that some sectoral variable is required on the right hand side of a macro equation. When this is the case, the *IdBuild* command *isvector* particularly is useful. This command indicates to *IdBuild* that a variable in one of the following save files is to be treated as an element of a vector, and not as a macrovariable or Tseries variable which is the default. For example, in the LIFT model of the U.S., the equation for railroad construction uses output of sector 59 (Railroads). Other equations use aggregates of output of many sectors. The included sections of files below show how this is handled. Here is part of the regression file for *G7*, CONSTR.REG:

```
save constr.sav
f outman = @csum(out,9 58)/1000.
f outbus = @csum(out,64,65,72,73,77 80)/1000.
f outtrade = @csum(out,69 71)/1000.
f outmin = @csum(out,2 6)/1000
#================================
ti 15. Railroad Construction
r cst15$ = cstoth, rpoil[2], rcbr[1], out59, doutrail, doutrail[1]
gr *
```

Here is a small MASTER file that will generate the code for this function only.

```
ba constr
isvector out,emp,pdm,cstk
iadd constr.sav
```

Here is the code for the *constrf()* function. Note that since the *isvector* command was in effect, *IdBuild* knows that "out" is a vector. Therefore, it passes "out" as one of the vectors in the argument list to *constrf()*, it writes out the *csum* function correctly as a method of type Vector, and it writes "out[59]" on the right hand side of the regression equation instead of "out59[t]".

```
void constrf(Vector& out,Vector& emp,Vector& pdm,Vector& cstk){
   outman[t]=out.csum("9 58")/1000.;
   outbus[t]=out.csum("64,65,72,73,77 80")/1000.;
   outtrade[t]=out.csum("69 71")/1000.;
   outmin[t]=out.csum("2 6")/1000;
   ...
   /*  15. Railroad Construction */
   /* cst15_ */ depend =3699.702916+ 0.405094* cstoth[t]+5.812924* rpoil[t 2]+
   17.332162* rcbr[t 1]+0.020963* out[59]+0.040612* doutrail[t]+
   0.117446* doutrail[t 1];
```

(continues on next page)

(continued from previous page)

```
  cst15_.modify(depend);
  ...
  }
```

At the present time, *IdBuild* doesn't know how to handle lagged values of vector variables. In this case, you can make the vector variable a macrovariable and include code in your model to copy the macrovariable values to the vector and back again. For example, another regression in the construction equations mentioned above uses construction capital stock of category 16 lagged once ("cstk16[1]"). The way to handle this is as follows. Before opening the save file in *G7*, first do:

```
f cstk16$ = cstk16
```

Then, include cstk16$[1] on the right-hand side of the equation. In the simulation model, remember to fill the macrovariable with the value of construction capital stock of category 16 before calling the construction function:

```
cstk16_[t] = cstk[16];
constrf(out,emp,pdm,cstk);
```

Note that in addition to the *iadd* command, *IdBuild* has one more command not found in *Build*. That is the *break* command. Its format is:

break <filename>

After the *break* command, subsequent C++ code will go to the named file, with the extension .CPP appended, rather than to HEART.CPP. With a large model, HEART.CPP can become too large and may fail to compile. Also, somtimes it is more convenient to group the functions written by *IdBuild* into smaller logically organized files. This command may help in these cases.

Macrovariables also can be used on the right-hand side of equations for vector variables, called "detached-coefficient equations". These are described in the *InterDyme* manual.

Please visit the Software pages of the Inforum web site for more details and to download the Fixer software: http://inforumweb.inforumecon.com/software/software.html.

INDEX

Symbols

.DFR Bank, 39
.OPT File, 116
.SHW File (*Show Files, see Results of Model*), 118
[], 33, 105
2-Stage Least Squares, 110
3-Stage Least Squares, 110

A

Add Files, 51
Add files, 34
Additive Errors, 117
Almon lag, 105
Annotating Graphs, 91
Arguments, 62
ARIMA, 106
ASCII Data, 68
ASCII Files, 70, 153
Assigned Data Banks, 39
Assigning Multiple Banks, 39, 118
Autocomplete, 24
autocomplete, 18
Autocorrelation, 105
Autocorrelation Function, 106

B

Balancing Entire Input-Output Tables, 150
Balancing Matrices, 148
Balancing Matrices with Uncertain Totals, 149
Balancing of Matrices, 148–151
Bank Types, 39
Banker
- How to Format Input Data, 294
- Notes for Programmers, 295
- Usage, 293
- What is Banker?, 293

Banker - The G-Bank Maker, 299
Bar Graph, 36
Basic Commands, 30
Basic G7 Commands, 246
Basic Regression, 33
Box-Jenkins, 106
Build
- Macro-Equation Processor, 291

Building a Model, 111
Building Models, 111, 122

C

C Compiler, 111
Calculation and Display of Running Totals, 281
Calculation of Data Within Compare, 281
Change Directory, 96
Choosing a File Type, 260
Chow Test, 105
Clear Results, 38
Clopper Almon, 3
Command Cache, 24
Command Files, 38, 51
Command files, 34
Command Files and the G7 Editor, 34
Command Files Within Command Files, 38
Command Reference, 29
Commands, 33
Commands for Sorting or Ranking, 282
Commands That Control The Printer, 266
Commands That Control The Table Format, 264
Commands to Build Tables in Excel Format, 275
Commands Used With Vam Files, 284
Common Coefficient, 107
Commonly Used Compare Commands, 261
Compare, 253
Compare Command
- *, 261
- ;, 261
- #, 261
- &, 261
- \add, 261
- \aggtype, 283
- \announce, 283
- \ar, 283
- \at, 283
- \blankzeroes, 264
- \bz, 264
- \cd, 284

\center, 261
\coefdecs, 284
\column, 284
\commas, 264
\cutoff, 284
\dates, 261
\decs, 264
\div, 261
\endsort, 282
\endtotal, 281
\es, 282
\et, 281
\f, 281
\fadd, 261
\field, 264
\filetype, 275
\font, 275
\fontbank, 275
\fontdata, 275
\fontdate, 275
\fonthead, 275
\fontname, 275
\fonttitle, 275
\fw dp pl tm bm tw, 264
\g, 261
\gd, 264
\glist, 284
\growthdecs, 264
\growthwidth, 264
\gt, 261
\gw, 264
\head, 261
\landscape, 266
\line, 261
\listgroups, 284
\lm, 266
\load, 284
\lpp, 266
\matcfg, 284
\matlist, 284, 288
\mc, 284
\mem, 283
\missing, 264
\mode, 261
\n, 261
\nocomment, 266
\noformat, 266
\nonzerorows, 264
\nosubtot, 284
\nst, 284
\nzr, 264
\pageprefix, 264, 283
\pages, 264
\pagewidth, 264
\pc, 266
\pcontrol, 266
\portrait, 266
\pp, 264
\pr, 283
\printbase, 261
\printsort, 282
\printtotal, 281
\ps, 282
\pt, 281
\pw, 264
\row, 284
\spacing, 264
\ss, 282
\st, 281
\startsort, 282
\starttotal, 281
\tell, 283
\thresh, 264
\threshhold, 264
\ti, 261
\timestamp, 283
\toc, 283
\ts, 283
\under, 264
\wsfreeze, 275
\wstitle, 275
\xlgraph, 275
\xls, 275
\yf, 264
Compare Funcion
    @csum(), 286
Compare Program, 111, 119
Compare program, 89
Compare Stub Files, 286
Compressed Bank, 39
Constant Term Adjustment, 113
Constraints, 103, 105
Continue command, 34
Control Totals, 145, 147
Conventions, 33
Copying a Vam File, 157
Creating Variables, 44

D

Data Banks, 39
Dates and Frequencies, 43
Default Vam File, 126
Derivatives of Regression Coefficients, 99
DH (*Durbin H Statistic*), 99
Dirfor Files, 288
Distributed Lags, 105
Do Loops, 51
DoFree (*Degrees of Freedom*), 99

drawing graphs, 91
DW (*Durbin-Watson Statistic*), 99

E

Editing, 27
Editor, 27, 34
Elasticity, 99
Email, 3
Error Handling, 60
Excel, 275
Excel Commands, 72, 87
Execute files, 34
Expressions, 44, 286

F

Fill (*Colors in Graphs*), 36
Fix File, 111, 113
Fixer, 305
    cta, 305
    dgro, 305
    dind, 305
    dstp, 305
    eqn, 305
    fol, 305
    gro, 305
    ind, 305
    mul, 305
    ovr, 305
    shr, 305
    stp, 305
Fixer - General Vector Fixing Program, 305
Fixer and MacFixer, 310
Fixes
    cta, 113
    mul, 113
    ovr, 113
    skip, 113
    update, 113
Fixing Programs for *InterDyme* Models, 310
Flow Control, 57
Folded Vector, 130
Font (*to display on screen*), 24
Forming Variables, 44
Formulas, 44
Frequency (*Vam Files*), 126
Functions, 47, 286

G

G.CFG File, 24
G7, 3
G7 Buttons, 249
G7 Command
    ", 159
    :, 159
    #, 159
    \gdata, 260
    \oneperline, 260
    \prn, 260
    \rtf, 260
    \wk1, 260
    \xls, 260
    ac, 106, 159
    add, 30, 34, 51, 159, 251
    addpause, 51
    addprint, 51, 159
    addtype, 159
    annline, 91
    anntext, 91
    autocomplete, 159
    autoprint, 91, 159
    bank, 30, 118, 161
    bj (*Box-Jenkins*), 161
    break, 57, 96, 161
    btitle, 39, 41, 161
    bupdate, 41, 161
    catch, 35, 51, 60, 96, 162, 218
    cbk, 39, 162
    cc, 122, 162
    cd, 96, 162
    chow, 162
    clear, 96, 162
    close, 39, 128, 162
    cls, 162
    cmtype, 99, 162
    coef, 152, 162
    comcoef, 107, 162
    commandcache, 162
    con, 162
    conditional formatting, 72, 228
    continue, 57
    cpress, 41
    ctrl, 44, 145, 147, 162
    data, 30, 68, 166
    del, 41, 166
    dfr, 166
    dfreq, 43, 166
    diagextract, 144, 166
    diaginsert, 144, 166
    dir, 96, 166
    do, 166
    dos, 166
    dvam, 128, 166
    ed, 170
    else, 57
    eqpunch, 154, 170
    f, 30, 33, 44, 122, 171, 246, 251
    fadd, 51, 171
    fdates, 43, 44, 171

fdup, 122, 171
fex, 122, 171
findmode, 96, 171
flow, 152, 171
format, 70, 171
freq, 43
frequency, 171
function, 51, 171
fvread, 130, 171
gdates, 43, 175, 246
getsum, 152, 175
glist, 140, 175
gname, 35, 38, 118, 175
gprint, 91, 175
gr, 30, 33, 118
gr *, 33, 99
graph, 175, 246
gridty, 70
gridtype, 175, 246
group, 140, 175
gsave, 91, 175
gtfile, 91, 175
gtitle, 91, 175
gvras, 148, 150
hbk, 179
hbk (*Hashed Bank*), 39
help, 179
hesitate, 179
hl, 179
hpress, 41
hr, 179
hrange, 91, 179
ic, 70, 181
id, 181
if, 57
if-else, 181
index, 144, 181
intvar, 99, 181
ipch, 154, 181
legend, 185
lgr, 91
lgraph, 185
limits, 30, 33, 185
line, 185
lint, 143, 185
linv, 138, 185
listbanks, 185
listgroups, 140, 185
listnames, 30, 39, 41, 185
listnamescol, 39, 185
listvecs, 126, 185
lnv, 185
load, 145, 185
Look, 26
look, 185, 246
ls, 44, 185
lv, 126, 185
lvc, 126
madd, 138, 191
matdat, 68
matdata, 191
matin, 132, 191
matin5, 132, 191
matprint, 191
matty, 70, 191
matupdate, 191
maxobs, 191
mcopy, 138, 191
mgr, 91
mgraph, 191
minv, 138, 191
missing, 44, 191
mmult, 138, 191
mode, 99, 191
monup, 68, 191
move, 191
mplot, 191
mtrans, 138, 191
mup, 191
nl, 198
nlp, 198
normality, 198
p123, 88, 200
p5, 200
pause, 51
pch, 200
plot, 175
pmatin, 132, 200
pmatin1, 132, 200
pmfile, 132, 200
pmpunch, 153, 200
print, 218
psras, 148, 149, 151, 200
punch, 153, 154, 200
punch5, 200
punchvec, 153, 200
pv, 153, 200
pwd, 200
pwd Command, 96
qpcon, 103, 204
quit, 30, 96, 204
r, 30, 33, 99
Random, 96
ras, 148, 151, 205
rdras, 148, 151, 205
recursive, 205
regress, 205
rename, 41, 110, 205

resector, 141, 205
return, 57, 205
rowscale, 205
rp123, 88, 205
rs, 141, 205
save, 35, 70, 211
scale, 147, 211
second, 110, 211
seed, 96, 211
setveclag, 211
sgraph, 211
Shift, 33
show, 135, 211
sma, 105, 211
splot, 211
sprint, 211
stack, 107, 211
stochastic, 211
store, 145, 211
str, 61, 211
sty, 122
stype, 70, 211
subtitle, 91, 211
sur, 107, 211
tdates, 43, 218, 246
text, 70
time, 96, 218
timer, 218
title, 30, 33, 91, 99, 218, 246
titpch, 154, 218
tpch, 154, 218
try, 60, 218
ts, 70
type, 30, 70, 218, 246, 251
update, 68, 221
vam, 128, 221
vamcr, 128
vamcreate, 221
vammode, 96, 221
vaxl, 91
vaxlab, 221
vaxti, 251
vaxtitle, 91, 221
vc, 136, 221
vdata, 129, 221
vf, 221
vmake, 136, 221
vmatdat, 130
vmatdata, 221
vp, 153, 221
vr, 221
vrange, 91, 221
vtitle, 221
vtitles, 154
vup, 221
vupdate, 129, 221
wrindex, 68, 228
wsbank, 228
wsinfo, 41, 228
wsreset, 228
xl, 72, 228
xl background, 228
xl border, 72, 228
xl cf, 72, 228
xl clear missing, 228
xl close, 228
xl column, 228
xl create, 72, 228
xl exit, 228
xl font, 72, 228
xl formula, 72, 228
xl freeze, 228
xl graph, 72, 228
xl gridlines, 72, 228
xl invisible, 228
xl matread, 72, 228
xl merge, 228
xl missing, 228
xl mkseries, 228
xl name, 228
xl open, 72, 228
xl print missing, 228
xl printer, 72, 228
xl read, 72, 228
xl replace, 228
xl row, 228
xl save, 228
xl setfont, 72, 228
xl setfrequency, 228
xl subscript, 228
xl vecread, 72, 228
xl vecwrite, 228
xl visible, 228
xl write, 72, 228
yearformat, 243
zap, 41, 243
zip, 243

G7 Command Box, 24
G7 Command Reference, 29
G7 Configuration, 24
G7 Console, 249
G7 Function
- @atoq, 47
- @bmk, 47
- @csum, 47
- @cum, 47
- @diff, 47
- @dlog, 47

@exp, 47
@fabs, 47
@ggr, 47
@gr, 47
@hpfilter, 47
@if, 47
@ifpos, 47
@interp, 47
@lint, 47
@log, 47
@max, 47
@mean, 47
@min, 47
@miss, 47
@mtoq, 47
@normal, 47
@pchain, 47
@pchwt, 47
@pcl, 47
@peak, 47
@pos, 47
@pow, 47
@qchain, 47
@qchwt, 47
@qtoa, 47
@qtom, 47
@rand, 47
@round, 47
@sin, 47
@sq, 47
@sqrt, 47
@stdev, 47
@sum, 47
@yoy, 47
@zero, 47
%{}, 62
%ceiling, 62
%eliteral, 62
%floor, 62
%literal, 62
%lower, 62
%max, 62
%min, 62
%mod, 62
%nseries, 62
%s, 62
%strcmp, 62
%strcmpi, 62
%strlen, 62
%strncmp, 62
%strncmpi, 62
%strnset, 62
%strstr, 62
%titlecase, 62
%trim, 62
%trimleft, 62
%trimright, 62
%upper, 62
%version, 62
%w, 61
%xldate, 62
%xlday, 62
%xlgdate, 62
%xlmonth, 62
%xlquarter, 62
%xls, 62
%xlyear, 62

G7 Functions, 47

G7 Keyword
- %NARGS, 62
- %%, 62
- %betaX, 62
- %date, 39, 62, 154
- %dw, 62
- %fdateX, 62
- %gdateX, 62
- %getval, 62
- %limX, 62
- %line, 61
- %linelen, 61
- %logl, 62
- %mape, 62
- %mexvalX, 62
- %ncoef, 62
- %nobs, 62
- %numwords, 61
- %obsinregression, 62
- %rbarsq, 62
- %rho, 62
- %rsq, 62
- %see, 62
- %subtitle, 62
- %tdateX, 62
- %time, 39, 62, 154
- %title, 62
- %vaxtitle, 62
- %xlcol, 62

G7 Keywords
- %NARGS, 57

G7 New
- For 2007, 20
- For 2008, 19
- For 2009, 19
- For 2010, 18
- For 2011, 17
- For 2012, 15
- For 2013, 12
- For 2014, 11

- For 2015, 10
- For 2016, 8
- For 2017, 7
- For 2018, 7
- For 2019, 6
- For 2020, 6
- For 2021, 5
- For 2022, 5
- For 2023, 4
- For 2024, 3

G7 Reference Manual, 158
- A, 159
- B, 161
- C, 162
- D, 166
- E, 170
- F, 171
- G, 175
- H, 179
- I, 181
- L, 185
- M, 191
- N, 198
- P, 200
- Q, 204
- R, 205
- S, 211
- Symbols, 159
- T, 218
- U, 221
- V, 221
- W, 228
- X, 228
- Y, 243
- Z, 243

G7 Tutorial, 245
Generalized Least Squares, 105, 107
Graph | Shift, 33
Graph Color, 36
Graph Mark, 36
Graph Styles, 36
Graph Width, 36
Graphing in G7, 36, 91, 118
Graphing Model Results, 118
Gross Value RAS, 148
group Fixes, 305
Groups, 51
Groups in Compare, 284

H

Hard Constraints, 103
HEART.DAT, 116
Hildreth-Lu, 105
Hodrick-Prescott Filter, 47
horizontal range, 91
How To Read and Write in Excel Format, 72

I

IdBuild
- Macro-Equation Processor for InterDyme, 311

IdBuild Menu Item, 123
Inforum, 3
Installation, 22
Interdyme, 126, 301, 305
Interindustry Forcasting, 3
Interpolation, 143
Introduction, 3
Introduction to Compare, 259

J

Jarque-Bera, 99, 105

L

Lags, 105
Legend, 91
Leverage, 99
Line Graph, 36
List Vectors, 126
Logarithmic Graphs, 91
Looking at a Series in a Data Bank, 26
Lotus, 88

M

MacFixer, 301
- cta, 301
- dgro, 301
- dind, 301
- dstp, 301
- eqn, 301
- fol, 301
- gro, 301
- ind, 301
- mul, 301
- ovr, 301
- rho, 301
- shr, 301
- skip, 301
- stp, 301

MacFixer - Macro Variable Fixing Program, 301
MacFixer Menu Item, 123
Making Tables, 89
MAPE (*Mean Absolute Percentage Error*), 99
Matrices, 286
Matrix Algebra, 138
Matrix Listing, 288
Mean (*Statistics*), 99
Methods of Balancing Matrices, 148
Methods of Scaling, 145

Mexval (*Marginal Explanatory Value*), 99
Miscellaneous Commands, 96, 283
Model, 111, 113, 116–119
Models, 301, 305
More on Add Files, 38
More on the f Command, 251
Moving Averages, 106
Moving Data from Vam to G7 Banks, 157
Moving Vectors and Matrices, 144
Multi-Scale Graph, 91
Multiple Banks, 118

N

Naming Graphs, 38
Naming graphs, 35
Nested Loops, 51
Nonlinear Regression, 108
Normality Test, 105
NorRes (*Normalized Residual*), 99

O

Objective Function, 116
Obser (*Number of Observations*), 99
Optimizing a Model, 111, 116
Ordinary RAS Method, 148
Ordinary Scaling, 145
Other Documentation, 158, 257

P

Packed Matrices, 132
Panel Data, 107
Parsing Routines, 61
PDG, 22
PMX Files, 132
Pooled Regression, 107
Powell's Method, 108
Preconditions (*Balancing of Matrices*), 151
Preconditions (*Part of gvras Command*), 150
Preconditions (*Part of pras Command*), 149
Preconditions (*Part of RAS Command*), 148
Preparing the Compare Stub File, 259
Printer, 260
Proportional Scaling, 145, 148
Proportional scaling, 147

Q

Quadratic Programming, 103

R

Ranking, 282
RAS Method, 148
RBSQ (*Adjusted R-Squared*), 99
rcoef (*Regression*), 99
Reading Data Into G7, 68
Recursive Regression, 99
Regression, 33, 99, 108
Regression Tests, 105
resid (*Residual Series*), 99
Results, 24, 38
RHO, 99
Rho Adjustment, 111
Rho adjustment, 113
Ridge Regression, 103
Right-Direction RAS, 148
Right-Direction Scaling, 145
RSQ (*R-Squared*), 99
Run dyme Menu Item, 123
Run Model, 111, 113, 116
Running a Model -- Deterministic Simulation, 113
Running G7 Scripts (*Add files*), 251
Running Interdyme, 123
Running the Compare Program, 286

S

Saving Graphs, 38
Saving graphs, 35
Saving Results, 35
Saving results, 35
Scaling, 145, 147
Scatter Graphs, 91
Scripts, 251
Scroll Results, 38
SEE (*Standard Error Estimate*), 99
SEE+1 (*SEE Using Rho Adjustment*), 99
Seemingly Unrelated Regressions, 107
Setting Dates in G7, 43
Setting Graph Styles, 36
Setting the Properties of the Results Area, 38
Settings, 36
Shortcuts, 24, 27
Sigma Bank, 117
Simplex Method, 116
Simplex Method of Non-Linear Solution, 108
Soft Constraints, 103
Software, 3
Sorting, 282
Special Uses with Compare, 288
Splicing, 41, 281
Spreadsheet, 275
Starting G7, 24
Step Size, 111
Stochastic Simulation, 111, 117
Strings, 61
Stub File, 259
Stub file, 26
Stub Files, 253
Sums of Matrix Columns or Rows, 148, 152
Systemic Least Squares, 110

T

t-value (*Statistic*), 99
Tables, 89, 111, 119, 253
Tables Express Menu Item, 123
Tables Menu Item, 123
Tabulating Model Results or Data, 119
Tabulating Results, 119
Text Files, 27
The 11 Basic G7 Commands, 30
The G7 Command Box and Button Bar, 249
The G7 Editor, 27
The G7 Main Window and Some Basic G7 Commands, 246
Title for Vam Files, 154
try-catch Routine, 60

U

User-specified Error Handling, 60
Using Compare to Make Tables, 253
Using the G7 Editor, 252

V

Vam, 286
Vam (*Bank Type*), 39, 128
VAM Configuration, 126
Vam File, 128, 138, 157
Vam File Data Display, 135
Vam File Input, 129, 130, 132
Vam File Title, 154
Vam Files, 126
VAM.CFG File, 126
Vam2Vam Program, 157
VamToG Program, 157
Variables, 62, 286
Variables, Expressions, and Functions, 286
Variance-Covariance Matrix, 99
Variance-Covariance Matrix of Regression Coefficients, 117
VecFixer Menu Item, 123
Vector Calculations, 136
Vectors, 286
Vertical Range, 91

W

Web Site, 3
What is G7?, 21
Where to Go to Learn More, 257
Windows Metafiles, 35
WK1, 88
Workspace Bank, 39
Writing Data, 70
wsbank (*Workspace Bank*), 41

X

XL Command Examples, 87
XL Commands, 87