Eleventh Inforum World Conference
Suzdal, Russia
2003 September 7-14

Progress towards Optimization in Interdyme Models

Clopper Almon

At last year's meeting, I described optimization in macro models built with G7 and Build. This optimization could be used in two ways:

(1) to find the values of the parameters of the equations – usually regression coefficients – which would optimize the fit of the model in historical simulation, and
(2) to find policies – such as tax rates or money supply – which would optimize some measure of performance of the economy. One might, for example, optimize some measure of welfare with respect to a carbon tax.

For the second type of application, the policy variable would be represented as a regression of the policy variables on a number of functions of time. Thus, in both cases, what is done technically is to optimize some function by varying certain selected regression coefficients.

As of last year's meeting, none of this optimization worked in Interdyme models. As a result of work from March through August of this year, it now seems very close to working in IdLift, the 97-sector U.S. model built with Interdyme. Indeed, if I did not point out a mysterious little problem, you might think that it was working perfectly. And the results look very encouraging. Small changes in the parameters can produce substantial improvement in the ability of the model to reproduce the course of the economy in a historical simulation.

I chose to work with IdLift rather than with a simpler model for two reasons. The first and clearest is that an optimized version of IdLift may prove very useful. The second is that I wanted a complicated model so that most problems would be encountered.

I also decided to work only with the regression coefficients of the macro equations, not the "detached coefficient" equations. This restriction to coefficients of macro equations is temporary; I see no problem in principle in extending the optimization to the detached coefficients. On the other hand, the biggest payoff is likely to be in optimizing with respect to coefficients in the macro equations.

Step 0 of the process was then to make IdLift operate in historical simulation. I was only partly successful in this step. When I began work on it, it would run only from 1990 to 1993. After considerable sleuthing as to why it would suddenly break down and produce negative outputs or prices, I have made it run from 1990 to 1997. Then in 1998 it explodes. I do not presently know why, but to get on with the optimization, I decided – after weeks of unsuccessful search – to get on with optimization over the 1990-1997 period. Since the model had not previously been used in historical simulation, it is not surprising that it did not perform very well. It appears to me that some further adjustments are necessary to get a satisfactory historical simulation of the model.

I will first show an example of how to set up optimization in an Interdyme model which has already been adapted for optimization, next sketch the optimization algorithm, and then show the results. I hope that by that point you will want to try optimization in your own model, so I will then explain how to adapt a model for optimization.

1. Example: Optimizing the Savings Function in IdLift

Specifying the Optimization

To optimize, we must first form an objective function and then specify the variables which will be varied to achieve the optimum. The historical simulation of IdLift showed that it quickly developed negative unemployment rates, which certainly do not speak well for the model. For this example, therefore, I took as the objective function the square of the difference between the true, historical unemployment rate, *unact,* and the unemployment rate calculated by the model, *un*. This objective function is built into the model by creating the following OBJECT.SAV file:

```
# Optimization variables.
# un is the percentage rate of unemployment
fex unact = un
f err1 = @sq(un - unact)
f error = @cum(error,err1,0)
```

The effect of these lines is to make *unact* the actual, historical rate of unemployment, to make *err1* the square of the difference between it and the unemployment rate calculated by the model, and to make *error* the running sum of squares of the differences between the historical rate and the one calculated by the model. It is the value of *error* in the last period which will be minimized, that is, the sum of the squared differences in all periods.

Then in the MASTER file, the line

```
iadd object.sav
```

must be included to get the code of OBJECT.SAV into the model. Finally, somewhere in MODEL.CPP, after the unemployment rate has been calculated, we need a call to Objectf(), like this:

```
// Calculate the objective function
Objectf();
```

With the objective function thus defined and calculated, we need to tell the program that *error* is the objective function and also to tell it with respect to which of the many regression coefficients it is to be optimized. That is done by the OPTSPEC.OPT file, which may read:

```
error
50
savrat
  .01  .01  0  0  0
```

The first line says that the last value of the variable *error* is to be minimized. The second says that no more than 50 parameters will be involved in the minimization. (In fact, only two are involved, but the 50 allows room for expansion without constantly revising the upper limit.) The third line says that parameters from the equation with *savrat* (the savings rate) as the dependent variable will be involved in the minimization. The fourth line says that it is only the first and second of those parameters which will be varied and that the initial step size of variation is .01 for each of them. The 0's in this step-size line mean that the corresponding parameters in the equation will not be varied.

Of course, to understand the economic meaning of this specification, one needs to know the equation for the savings rate. It comes out of G as:

```
r savrat =    -2.113946*intercept +
               0.670269*(gnpgap-100) +
               1.472484*rtb[1] +
               1.151521*savdummy
```

where *gnpgap* is the ratio of model-generated GNP to an estimate of potential GNP, scaled to have a "normal" value of 100. *rtb* is the interest rate on Treasury bills and *savdummy* is a dummy variable to account for an exceptional value of the savings rate in one year. The key role here is played by the *gnpgap* variable. The idea is that in good times, people will increase their savings, while in recessions, they will eat into their savings. Our optimization is done with respect to this coefficient and the intercept.

With the optimization criterion and variables specified, there is but one last step: tell the model to optimize instead of simply running a historical simulation. This is done in the last lines of the DYME.CFG file. In the case of IdLift, this file has been renamed IDLIFT.CFG and you may also have named this file for your model. Here are the top and bottom of this file as used here; new material is in bold:

```
Title of run   ;Suzdal Example of Optimization
Start year     ;1990
Finish year    ;1997
Discrepancy yr ;1990
Use all data?  ;yes
VecFix file    ;Vecfixes
MacroFix file  ;Macfixes
Vam file       ;dyme
G bank         ;dyme
```

... <Several lines are omitted here> ...

```
Name of Optimization specification file; OptSpec.opt
Number of random draws; 0
Additive random errors; no
Random coefficients; no
```

The line crucial for the optimization is the fourth from the end. It will cause the OptSpec.opt file described above to be read and used. (To run the model without optimization, simply put "none" on this line or leave it blank.) Even though stochastic simulation is not yet fully operational, the reading of the configuration file is ready for it, so the last three lines above must be present.

The Downhill Simplex Optimization Method

The method used to minimize the objective function, the Downhill Simplex, was explained in last year's paper and is described in Part 2 of *The Craft of Economic Modelling.* I quickly review it here. A simplex in *n* dimensions is *n+1* points that do not lie in a subspace of lower dimensions. The Downhill Simplex methods begins by selecting a simplex with the first point being the estimated values of the parameters to be varied and *n* other points obtained from it by varying one parameter at a time by the step sizes given in the optimization specification file, here called OPTSPEC.OPT. The method then tries to get a point better that the worst point by *reflecting* the worst point through the mean of the other points. If that works, it then tries another step of the same size in the same direction, a step called *expanding.* If that fails, the method retreats to the reflected point. But if the reflected point fails to improve on the worst point, a step halfway from the worst point to the mean of the others is tried, a move called *contracting.* If contracting also fails to get a point better than the worst point, a *shrink* is done moving all the points other than the best point halfway towards the best point. When the difference between the value of the objective function at the best point and at the worst point is less than some specified fraction of their average, the process stops.

Results

The log of the optimization process is shown in the box below. My impression is that the process took about ten minutes. The resulting changes in the regression coefficients were fairly minor, as shown by the following table.

```
Resulting coeficients after optimization
Variable              Old              New    Variable
savrat            -2.1139          -2.0431 (intercept)
savrat             0.6803           0.9553 (gnpgap)
```

The changes in the fit of the model, however, were considerable, as shown by the two graphs below. The error has been reduced by more than 50 percent.

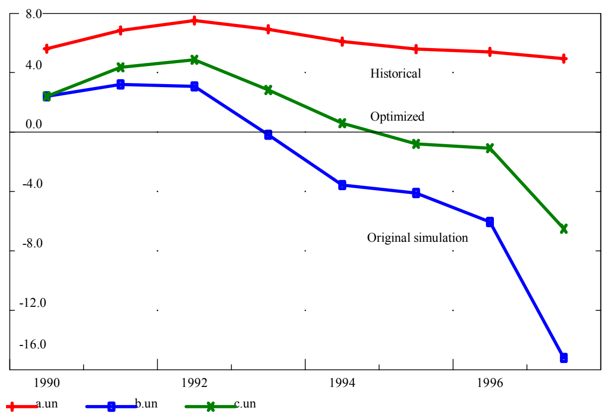
While this improvement is impressive for such a simple change, it might have been even greater except for the mysterious small problem noted at the beginning of this paper. This problem appears about halfway down the log where the one and only Shrink step (in bold type) occurred. Notice that in the Shrink step the best point got worse. That should not happen. The value for the best point is simply calculated with the same values of the parameters as it was originally calculated with. The answers should be identical. The fact that they are not means that something is changing from one iteration to the next besides the values of the parameters – some initial value, or fix, or rho adjustment, or something else. I

Log of Optimization Process

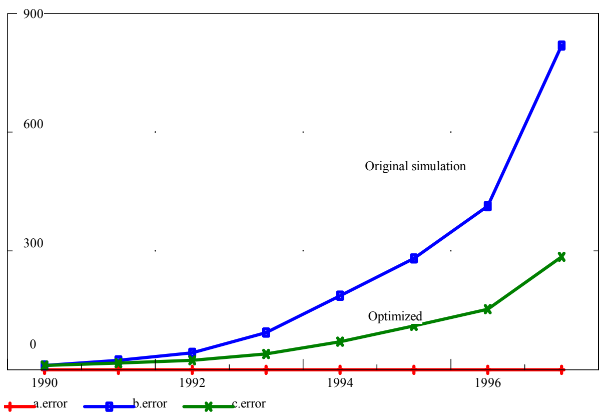
```
                     Objective Function Values
                    Best Point      Worst Point     Diff/Average    Criterion
Initial:            699.1032104     719.4415894     0.0286750       0.0010000
Expand:             692.8596191     718.2062988     0.0359256       0.0010000
Expand:             655.3725586     699.1032104     0.0645721       0.0010000
Expand:             629.1927490     692.8596191     0.0963152       0.0010000
Expand:             583.6524658     655.3725586     0.1157686       0.0010000
Expand:             515.0829468     629.1927490     0.1994446       0.0010000
Expand:             418.9052124     583.6524658     0.3286539       0.0010000
Reflect:            418.9052124     515.0829468     0.2059506       0.0010000
Contract:           418.9052124     442.9800110     0.0558654       0.0010000
Expand:             361.0180664     440.4505920     0.1982174       0.0010000
Contract:           361.0180664     429.1417236     0.1724301       0.0010000
Reflect:            361.0180664     418.9052124     0.1484432       0.0010000
Contract:           361.0180664     415.6075134     0.1405811       0.0010000
Contract:           287.8601685     375.1665649     0.2633571       0.0010000
Contract:           287.8601685     361.0180664     0.2254904       0.0010000
Shrink:             347.1479187     363.3522339     0.0456138       0.0010000
Contract:           347.1479187     360.6577148     0.0381737       0.0010000
Expand:             286.1284790     347.1479187     0.1927103       0.0010000
Contract:           286.1284790     347.1479187     0.1927103       0.0010000
Reflect:            286.1284790     288.7161865     0.0090032       0.0010000
Expand:             284.4019165     286.9833679     0.0090358       0.0010000
Contract:           284.4019165     286.1284790     0.0060525       0.0010000
Expand:             283.5445862     286.1201782     0.0090425       0.0010000
Contract:           283.5445862     285.0385132     0.0052549       0.0010000
Contract:           283.5445862     284.5006714     0.0033662       0.0010000
Contract:           283.5445862     284.4019165     0.0030191       0.0010000
Expand:             283.3757935     284.2439575     0.0030590       0.0010000
Contract:           283.3757935     283.8456421     0.0016567       0.0010000
Contract:           283.3757935     283.6492310     0.0009645       0.0010000
```

have spent a lot of time identifying things which might change and insuring that they did not. But I must have missed something. I will have to keep trying. From the fact that the value of *error* went up when recalculated, I had guessed that rerunning the model with the optimized values of the regression coefficients might give a better value of *error*, but such was not the case; the value of *error* got a little worse.

Unemployment Rates



Error



I hope that these results are dramatic enough to make you want to optimize in your model. If

I have been successful, you will want to know how to adapt your model for maximization, and that is our next topic.

2. Adapting a model for optimization

The steps above assume that the model has been adapted for optimization. I will now explain as much as I have been able to find out about what has to be done for this adaptation. The "mysterious small problem" reported above would seem to indicate that I have not found everything necessary, but surely I must have found most of the necessary things. To tell the truth, I do not suggest that you start the process until I can report that it works totally without any problems.

Step 1. Run the New IdBuild 3.0.

In previous versions of the Interdyme software, the regression coefficients of the macro equations were constants in the C++ code. For example, the savings rate equation, which comes out of G looking like this:

```
r savrat =    -2.113946*intercept +
               0.670269*(gnpgap-100) +
               1.472484*rtb[1] +
               1.151521*savdummy
```

After processing by IdBuild versions below 3.0, the C++ code of the model looks like this:

```
/* savrat */ depend = -2.113946+
     0.670269*(gnpgap[t]-100)+
     1.472484*rtb[t-1]+
     1.151521*savdummy[t];

savrat.modify(depend);
```

In this form, optimization (or stochastic simulation) with respect to the coefficients is impossible because the program cannot vary a constant in the code.

The first step towards optimization was therefore to rewrite IdBuild to produce code like the following:

```
/* savrat */ depend =   +coef[87][0] +
          coef[87][1]*(gnpgap[t] + coef[87][2]) +
          coef[87][3]*rtb[t-1] +
          coef[87][4]*savdummy[t];

savrat.modify(depend);
```

Here, the constants have been replaced by variables, elements of the `coef` array. (The savings equations happens to be equation number 87 in this array.) In this format, the program can vary the elements of the `coef` array to optimize or perform stochastic simulation. IdBuild 3.0 and above creates this sort of code.

IdBuild 3.0 also writes two files, HEART.DIM and HEART.DAT, that give, respectively, the dimensions of the `coef` array and its values. The user should have no occasion to look at these files, but, since they are "humanly" readable, I will put their explanation here in in a box which can be skipped without loss of continuity.

The ouput of IdBuild 3.0 is **not** compatible with earlier versions of Interdyme. Since it seems unlikely that anyone will want to use it before we have ready a "pedagogical" model for the rest of Interdyme consistent with this IdBuild, I decided against making a general distribution of it at this conference. It is, however, available on request.

It took me a while to make these changes in IdBuild, but for you, the user of the Interdyme software, this step is very easy: just run the new IdBuild 3.0 with your existing Master and .sav files. It should take only a few seconds.

Step 2. Make the Interdyme model run with input from IdBuild 3.0.

This step should also be fairly easy. First, add the file MAXI.CPP to the Interdyme model project. (In Borland Builder, click the "Project" item on the main menu, then click "Add to project." This Maxi module contains most of the new code specifically necessary for optimization and, for that matter, for stochastic simulation. Most of it was borrowed from the same module for macro models with very little change.

The routines in MAXI.CPP require some new global variables, so the following lines need to be added to DYME.INC to make compilation possible.

```
/* For optimizing and stochastic simulations*/
// Connected with Optimization or Stochastic Simulation
GLOBAL char MaxFlag,OptSpecFileName[80],objective[60];
GLOBAL short rnderr, rndcoef,num_draw,ranseed,iobjective;
GLOBAL char  randm;
GLOBAL short npy;
GLOBAL float increment;
GLOBAL short EquationCount,nequ;
GLOBAL short *ncoef;
GLOBAL float **coef, *rho1, *see1, **svp, **mvp;
GLOBAL FILE *fpbug;
GLOBAL FILE *fpdat;
GLOBAL char    **depvarnames, **eqname;
```

The HEART.DIM and HEART.DAT Files

Here are the first few lines of HEART.DIM for IdLift

```
  89
d 7
d 7
d 6
d 6
```

The first line says that there are 89 macro equations, so the `coef` array needs 89 rows. The next line says that the first of these equations was a "deterministic" equation, so no space needs to be allocated for the variance-covariance matrix of the regression coefficients – and they do not need to be read or even skipped over. The same line informs the program that this first equation has 7 coefficients, so the first row of the `coef` array needs 7 columns. The second row will also need 7 columns, but the third and fourth rows need only 6 columns. Thus, `coef` is not a rectangular array.

Here are a few lines of the HEART.DAT file

```
cst1h
coeff =
0.420538  0.002037  0.144628
0.095173  -0.107844  7.092513
0.023499
[... many lines omitted here]
savrat
coeff =
-2.113946  0.670269  -100  1.472484
1.151521
```

The last four lines above give the equation for the savings rate. When the values of the parameters given here are substituted for coef[87][1] ... coef[87][5] in the second, new, form of the equation, one gets the same equation as given by the old form.

It must be emphasized that the user should never need to look at or modify these files. This explanation is given just to avoid the impression that they are a "black box."

These lines will, of course, already be in DYME.INC as it comes with the new Slimdyme model adapted for optimization, but if your DYME.INC has been changed from the standard, you may want to include the new lines in your old file. Some of these new variables apply only to stochastic simulation which is not yet fully operational, but it seemed pointless to remove them from the code only to return in a few months to put them back in.

To get values for these variables, a few lines need to be added to the DYME.CFG file, as illustrated above. Code needs to be added to the *configure* function in CONFIG.CPP to read these additions. It should be something like

```
    err += getopt(fpcfg,szOptSpecFN,60);// a file name
    err += getopt(fpcfg,szRandomDraws,7); // number
    err += getopt(fpcfg,szAdditiveRandomErrors,5); // yes or no
    err += getopt(fpcfg,szRandomRegressionCoefficients,5); // yes or no
```

After interpreting other items in the dyme.cfg file, the new items must also be interpreted with code like this:

```
    depad(szOptSpecFN);
    if(strcmp(szOptSpecFN,"none")== 0 || strlen(szOptSpecFN) == 0)
        MaxFlag = 'n';
    else{
        MaxFlag = 'y';
        strcpy(OptSpecFileName,szOptSpecFN);
        }
    num_draw = atof(szRandomDraws);
    depad(szAdditiveRandomErrors);
    depad(szRandomRegressionCoefficients);
    if(szAdditiveRandomErrors[0] == 'y') rnderr = 1;
    else rnderr = 0;
    if(szRandomRegressionCoefficients[0]== 'y') rndcoef = 1;
    else rndcoef = 0;
    // Check consistency of desires
    if (num_draw > 0 && rnderr + rndcoef == 0) num_draw = 0;
    if(MaxFlag == 'y' && num_draw >0){
      printf("Cannot have both optimization and stochastic simulation.\n");
        printf("Fix .cfg file.\n");
        exit(1);
        }
```

The MAXI.CPP module contains the function *coefdim*, which reads the HEART.DIM file, and the function *coefread* which reads the HEART.DAT file created by IdBuild. To use these routines, you need to modify the main program of your model, which is probably in DYME.CPP. You should find there the lines:

```
      if((err=Initialize(argc,argvec))==ERR) {
            exit(1);
            }
```

This call to Initialize will read the DYME.CFG file and detect the information you have put at the end of it which informs the program that you want to optimize. Right after these lines, put the following code:

```
      // read number of variables and stochastic indicator for each equation
      nequ = coefdim();

      /* If we are doing a deterministic or optimizing simulation,
      num_draw, the number of draws of random coefficients will be zero.
      In that case, call coefread to read the regression coefficients of
      the macro equations from Heart.dat and, if optimizing, the .opt file
      which specifies the objective function and which coeffients in which
      equations are to be varied, and the initial stepsizes.
      */

      if(num_draw == 0) coefread();

      /* Otherwise -- that is, if we are doing stochastic simulations --
      read heart.dat and generate the first set of random coefficients. */
```

```
    else coefgen();
```

Again, this code will be standard in the next version of Slimdyme, the pedagogical model, but it is given here in case you are modifying an existing model.

At this point, you should be able to run the model in standard, non-optimizing, non-stochastic simulation but with the regression coefficients of the macro equations in the `coef` array rather than as constants in the program. This step was also fairly easy, since most of the new programming was down in MAXI.CPP where you did not have to touch it.

Step 3. Globalize the Equations, Matrices and Vectors

The optimization code in MAXI.CPP assumes that a call to a new function, `spin()`, which you must write, will run the model through the cycle of years. The `spin()` function must, of course, have access to all the Equation, Matrix, and Vector objects which have been created. The standard Slimdyme of the past has put all the Equation, Matrix, and Vector declarations in the `loop` function in MODEL.CPP. For some time, we have recognized that that practice was unfortunate because it resulted in a lot of passing of arguments to functions. The need to write `spin()` forced a resolution of this issue. The solution is to make all of these objects global. To do so, we put their declaration into USER.H, where only a default constructor can be put which allocates no memory and loads no data. Then in the loop function in MODEL.CPP we call a new method for each vector, matrix, or equation, named simply *r* for *read* which does the work formerly done by the declaration. For example, IdLift formerly had

```
    Vector out("out",'a'), im("im",'a'), fd("fd"), pdm("pdm",'a');
```

in the loop() function in MODEL.CPP. It now has

```
    GLOBAL Vector out, im, fd, pdm;
```

in USER.H and

```
    out.r("out",'a'); im.r("im",'a'); fd.r("fd"); pdm.r("pdm",'a');
```

in MODEL.CPP where the above declaration used to be. Otherwise, the loop() function is pretty much unchanged down to the big loop over the years which begins

```
    for (t = godate; t<= stopdate; t++) {
```

That loop is now moved to a separate function, `spin()`, in MODEL.CPP.

Step 4. Write the spin() function.

The spin() function will look something like this:

```
void spin(){
      int i,j,k, err;
      ... [other declarations] ...
      tserin();// load macrovariables
      for (t = godate; t<= stopdate; t++) {
      ... [all of the existing code for the model] ...
      shiftback();
      }
```

Note that there is a call to tserin() at the beginning to reinitialize all the time series variables before each "spin" of the model. Also note that at the end of the each year's calculation there is a call to `shiftback()` rather than to `store()`. The call to `store()` would store the calculated values of the vectors and matrices in the vam file and shift back the matrices of lagged values of vectors. The call to `shiftback()` only shifts back the matrices of lagged values of vectors. This change avoids any change in the objective function coming from changed starting values of vectors or matrices. The code for `shiftback()`, which is new, is found in IDRUN.CPP. The `spin()` function must **not** contain any call to `storets()` which stores time series values.

Back in the `loop()` function, the whole loop beginning

```
      for (t = godate; t<= stopdate; t++) {
```

is replaced by

```
if(MaxFlag == 'n')
    spin();
else
    maxsolve();

storets(); // Store the macro variable time series
```

The first two lines allow for running the model in an ordinary simulation without maximization; the value of `MaxFlag` will be `'n'` if there was no Optimization Specification file given in the DYME.CFG file.

The code for `maxsolve()` is in MAXI.CPP and should not require modification from one model to another.

3. What comes next

Looking back at these steps, it is a little hard to see how it took me six months to get optimizing working. A lot of time went into finding my way around IdLift, and of course it is easier to say "the code is in MAXI.CPP" than it is write that code. I hope that I can soon identify the "mysterious little problem" and produce a guide to adaptation for optimization which I can be reasonably sure is complete. I will then produce a version of Slimdyme coded in the new way. In the meantime, if you want to get started with optimization in its current "beta" state, I will be glad to share with you what I have.