Interdyme Report #1: Import Group Fixes

Douglas Meade - May, 1995

Introduction

Fixes for elements of vectors or for predefined groups of sectors in a vector are now conveniently imposed within an Interdyme model, using the *Fixer* program, in conjunction with an input file such as VECFIX.VFX, that defines the groups and the fixes. However, the fixing of total imports, or of groups of sectors of imports can cause trouble if we try to use simple right-directional scaling. Half of the problem is that imports can only meaningfully be used to satisfy some component of domestic demand, either intermediate or final demand. The other half is that imports of one industry help determine output for that industry, and therefore also influence domestic demand, and therefore imports, for many other industries. In other words, imports and output of all industries are interdependent.

If the model is forced to take more imports than it has demand for a certain industry, then output for that industry will be calculated as negative in Seidel(). This unreasonable result, if not corrected in Seidel(), will then cause other parts of the model to react strangely. For a forecast of imports to be reasonable, then imports must be less than or equal to domestic demand. How can we scale imports to a control total, yet ensure that imports satisfy this condition?

A Scaling Rule

A rule is required which alters these ratios in a sensible manner. We have chosen a fairly simple rule, in which the largest percentage changes to import/domestic demand ratios are made to sectors which have relatively low import shares or penetration levels already. However, the rule does not raise the penetration levels of an industry with a share equal to zero, since imports will most likely not be demanded for this industry. The rule can be expressed by the following formula:

$$m_j = \frac{m_{j0}}{m_{j0} + \lambda(1 - m_{j0})}$$

with λ determined implicitly by imposing the constraint, or control that:

$$C = \sum_j m_j D_j$$

where the m_{j0} are the initial import shares (before scaling), formed as $m_{j0} = M_{j0}/D_j$, M_{j0} are the initial imports for each sector, and D_j is domestic demand for each sector. C is the control total to which the imports must sum, and the m_j are the final import shares.

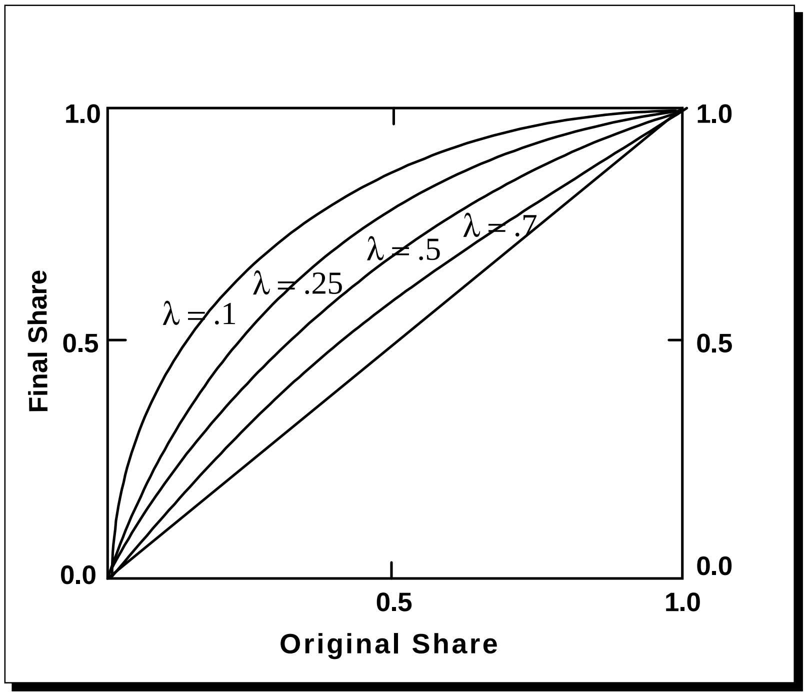
Final imports are recalculated as $M_j = m_j D_j$. If the solution is possible, then $C = \sum_j M_j$.

This nonlinear function of λ is easily solved by a modified Newton-Raphson algorithm, which will find the zero root of the function:

$$f(\lambda) = C - \sum_j \frac{m_{j0} D_j}{m_{j0} + \lambda(1 - m_{j0})}$$

The relation between the starting value m_{j0} and the revised value m_j is shown for various values of λ in Figure 1 below.

Figure 1. Share Revision Functions



This algorithm can also be (and was originally) applied to the problem of forecasting coefficient change in an import matrix. In this context, we may know the row control of total imports for a given commodity. We also have base period import coefficients, that express intermediate imports as shares of total intermediate for each cell, and final demand imports as a share of that category of demand. In this case, the same problem can be posed as finding the zero root to the function:

$$f(\lambda) = Y_i - \sum_j \frac{m_{ij0}x_{ij}}{m_{ij0} + \lambda(1 - m_{ij0})}$$

where: Y_i = total intermediate imports for row i

m_{ij0} = base year import coefficient for row i, column j

x_{ij} = intermediate flow in row i, column j

and the j index now runs over the categories of final demand as well as the columns of the input-output matrix, and for the categories of final demand, x_{ij} refers to the imports/demand ratio for that category.

The imsis() Function

A new function has been written for Interdyme that can handle either one of the above problems. It is called imsis(), and its calling signature is:

```
short imsis(float control, short n, short *group, float *mvec, float
*dd);
```

The `control` argument is the control total (C or Y in the above examples) to which imports must sum. The pointers `mvec` and `dd` point to zero-offset arrays of imports and domestic demand, or intermediate import flows and total intermediate flows for a row. Finally, `n` is the number of items in the vector to be scaled, and group is a list of length `n` of the sector indices (position in `im` or `dd` plus 1) of the items to be scaled.

This function in turn calls a modified Newton-Raphson algorithm called imnewton(), which is described briefly in the technical appendix to this report.

How to do Import Fixes in Seidel()

Applying the import fixes properly in the Seidel() routine can be tricky, and our suggested solution involves compromising between a number of goals. The difficulty is due to the very reason that imports are calculated in Seidel: imports and output are simultaneous; and imports should not be greater than demand.

In short, here is how import fixes are done in the Seidel which is provided in the version 2.07 MODEL.CPP file:

1. In the first pass through the Seidel iteration loop, imports (in the "im" vector) are calculated based on total domestic demand, and output is determined simultaneously. For each sector, the Vector fix function is called with the sector number `i` as a second argument, indicating that we check for single sector fixes:
   `im.fix(t,i)`
   (prototype: `short Vector::fix(short t, short i);`)
   If imports are fixed, then output is re-calculated to be consistent with imports.

2. After the Gauss-Seidel algorithm has converged for all sectors, the fix routine is called again, this time with the Vector of domestic demand `dd` as the second argument. The fact that the second argument is a Vector indicates that it is being asked to do *group fixes for imports*.

   ```
   im.fix(t,dd)
   ```
   (prototype: `short Vector::fix(short t, Vector dd);`)

   where `dd` is a measure of domestic demand[1]. This version of fix calls imsis() to perform scaling to the desired aggregate control.

3. Before calling this fix function, imports were saved into a scratch vector. After the fix function completes, the new vector is compared to the scratch vector. Any sectors that are different are assumed to have been subject to import group fixes. A vector of flags is set to 1 in positions where imports have changed, and 0 elsewhere. Also, a global flag is set to 1 to indicate that some imports have been subject to group fixes, and if this is true, Seidel returns to the top of the Gauss-Seidel iteration.

4. On the second pass through the Gauss-Seidel iteration, the imports of sectors that have been flagged as having been affected by import group fixes are not calculated, but rather treated as unchangeable. This is because if they were allowed to be recalculated themselves, the whole point of calculating the other sectors after the group fix would be futile.

5. After the second pass through Seidel, the solution is complete, and the import group fixes do not need to be applied a second time.

Rho-Adjusting Imports

In the framework of Interdyme, first each vector is calculated with the flag `setrho` set to 'n'. This means that when the Equation.rhoadj () function is called, the rho-adjustment is applied, but the error for future years is not calculated or updated. Within the Seidel() function, imports are rho-adjusted with `setrho` equal to 'n'. Single-sector fixes are applied, and then group fixes are applied. Imports not affected by group fixes are then recalculated within the Seidel loop, holding the group-controlled imports constant. Single-sector import categories are then rho-adjusted and fixed once more.

After the model has converged for a given year, `setrho` is set to 'y', and a function called impfunc() is called. Impfunc calculates imports outside the Seidel function, using (out+imp) as a measure of domestic demand[2]. The Equation.rhoadj() function is called, which this time calculates the error (if the year is less than or equal to the rhostart year) or

[1] Domestic demand is conveniently available in the Seidel iteration, for we have:

```
sum=fd[i];
for(j = first; j <= last; j++){
  sum += A[i][j]*q[j];
  }
dd[i] = sum;
```

[2] In many models, domestic demand for imports is calculated as (out+imp-exp).

updates the error for next year (in all subsequent years). Note that impfunc also applies all single-sector fixes and then all group fixes again.

The Output Calculation in Seidel(), With a Simple Import Equation

In the case of a simple import equation, which is a linear function of domestic demand, we have:

$$M_i = b_{1i} + b_{2i} D_i$$

In the calculations of Seidel, D_i is defined as

$$D_i = F_i + \sum_j a_{ij} Q_j$$

A variable called `sum` is being accumulated for use in the solution for output, which is expressible as:

$$sum = F_i + \sum_j a_{ij} Q_j - a_{ii} Q_i$$

Therefore:

$$D_i = sum + a_{ii} Q_i$$

Note that we can write the output identity for Q_i as:

$$Q_i = F_i + \sum_{j \neq i} a_{ij} Q_j + a_{ii} Q_i - M_i$$

Substituting the expression for D_i into the equation for M_i, we get:

$$M_i = b_{1i} + b_{2i}(sum + a_{ii} Q)$$

and then substituting this into the output identity, we get:

$$Q_i = F_i + \sum_{j \neq i} a_{ij} Q_j + a_{ii} Q_i - (b_{1i} + b_{2i}(sum + a_{ii} Q_i))$$

Combining terms yields:

$$(1 - a_{ii} + b_{2i} a_{ii}) Q_i = F_i + \sum_{j \neq i} a_{ij} Q_j - (b_{1i} + b_{2i} sum)$$

or

$$Q_i = \frac{F_i + \sum_{j \neq i} a_{ij} Q_j - (b_{1i} + b_{2i} sum)}{(1 - a_{ii}(1 - b_{2i}))}$$

This explains the line in Seidel where the output calculation is made for each sector:

```
out = (sum - imports[j][1] - imports[j][2]*sum - imports.errors[j])
  /(1.- A[i][i]*(1. -imports[j][2]));
```

For years in which imports are not calculated by equation, the code is much simpler:

```
out = (sum - imp[i])/(1. - A[i][i]);
```

Sample Seidel() Function with Import Group Fixes

```
short Seidel(Matrix& A, Vector& q, Vector& imp, Vector& sdc,
   Equation& imports, Vector& f, short *triang, float toler)
{
   short i,j,k,first,last,n,iter,im1,imax,retval;
   float depend,discrep,dismax,fdismax;
   double sum,out;

   iter = 0; // Iteration count
   n = A.rows();
   first = A.firstcolumn();
   last =  A.lastcolumn();

   if(q.numelm() < A.rows() ){
    printf("In Seidel, the solution vector is not large enough.\n");
    return(ERR);
    }
   short m = imp.rows();
   Vector impsave(m), dd(m), qact(n);
   dd.set(0.);
   qact = q;
   short *impchanged = new short[m+1];
   short GroupFixFlag=0;  // Indicates if some imp grp fixes have been applied.

   restart:
   // If group fixes on imports, go through again, with no recalculation
   //   of fixed sectors.
   while(1){  // start Seidel loop.
    dismax = 0;
    for(k = 0; k < n; k++){
      i = triang[k];
      sum = f[i];
      for(j = first; j <= last; j++){
        sum += A[i][j]*q[j];
        }
      dd[i] = sum;
      sum -= A[i][i]*q[i]; // Take off the diagonal element of sum
      // Imports Calculation here;
      // If we are before the last year of data, or if this sector
      //   was affected by an imports group fix, take imp as given.
      if(t <= imp.LastDat || (GroupFixFlag && impchanged[i]) ){
        out = (sum - imp[i])/(1. - A[i][i]);
        }
      else {
        /* The following formula for output is derived as follows,
        with m1 and m2 as the parameters of the import equation.
        out = sum + aii*out - imp
        out = sum + aii*out -(m1 + m2*(sum + aii*out) + error)
        (1 - aii + m2*aii)out = (sum - m1 - m2*sum - error)
        */
        j = secimp[i]; // j is the import equation number for sector i
        if(j == 0){
          imp[i] = 0;
          out = sum/(1. - A[i][i]);
          }
      else{  // Here is an example imports equation:
        out = (sum -imports[j][1] - imports[j][2]*sum - imports.errors[j])
        /(1.- A[i][i]*(1. -imports[j][2]));
        imp[i]=imports[j][1]+imports[j][2]*(sum+A[i][i]*out)+imports.errors[j];
        depend = imp[i];
        imp.fix(t,i);
        if(fabs(imp[i]-depend)>1.0e-7) { // Single-sector fix, recalc out
          out = (sum - imp[i])/(1. - A[i][i]);
          }
        }
      }

    discrep = fabs(out - q[i]);
    if(discrep > dismax){
      dismax = discrep;
      imax = i;
      fdismax = fabs(out);
      fdismax = (fabs(fdismax-0.0)<.000001)?0.0:dismax/fdismax;
      }
    q[i] = out;
    arith("Seidel",i);
    }
  iter++;
  if(dismax < toler || fdismax < .000001) break;
  if(iter > 25 ){
    printf("No convergence in %d iterations. Discrep = %7.2f in sector %d.\n",
      iter, dismax, imax);
    retval = ERR;
    goto cleanup;
    }
  }
gotoxy(26,wherey());
cprintf("%d ",iter);
// If import group fixes haven't already been done, do them.
if(!GroupFixFlag) {
  impsave = imp;
  imp.fix(t,dd);
  for(i=1;i<=m;i++)     {
    if(impsave[i] != imp[i]) {
      impchanged[i] = 1;
      GroupFixFlag = 1; // indicate there are group fixes.
      }
    else impchanged[i] = 0;
    }
  if(GroupFixFlag)
    goto restart;
  }

retval=OK;
cleanup:
if(impchanged)
  delete[] impchanged;
return retval;
}
```

Code for Import Equation Function impfunc()

```
/* impfunc() -- the imports functions for Slimdyme */
short impfunc(Vector& im, Vector& out,  Equation& imports)
{
 short n,i,j;
 float depend;
 n = imports.neq;
 Vector dd(NSEC);
 for(i = 1; i <= n; i++){
   j = imports.sec(i);
```

```
  depend = (imports[i][1] + imports[i][2]*out[j])/(1.-imports[i][2]);
  im[j] = imports.rhoadj(depend,im[j],i);
  }
// Redo import single-sector fixes and group fixes:
dd = out + im;
short m = im.numelm();
for(i=1;i<=m;i++)
  im.fix(t,i);
im.fix(t,dd); // group fixes.
return(n);
}
```

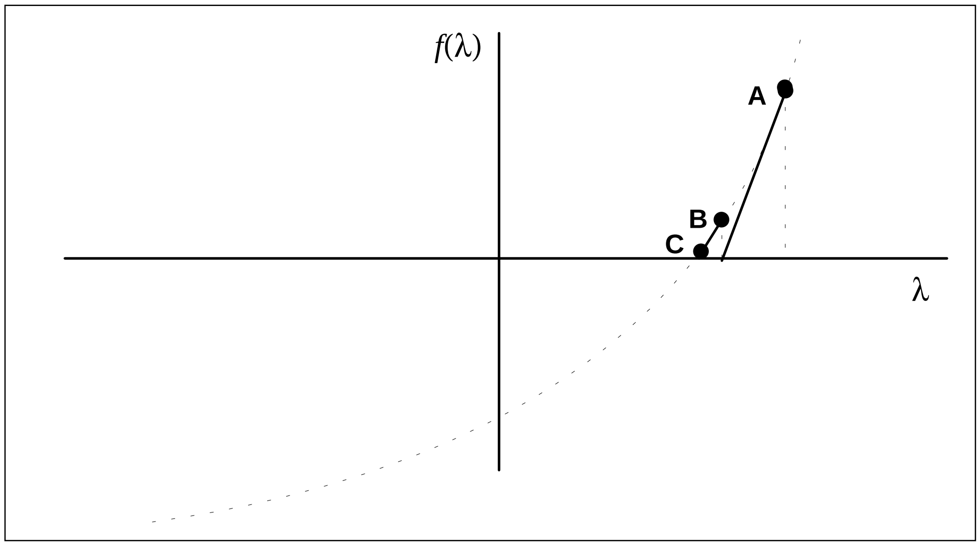
Technical Appendix: Modified Newton Method

The function that needs to be solved :

$$f(\lambda) = C - \sum_j \frac{m_{j0} D_j}{m_{j0} + \lambda(1 - m_{j0})}$$

is a function of only one variable, λ. Therefore, we use a one-dimensional root-finding routine. Where a first derivative of the function is easily expressed, the *Newton* method, also known as the *Newton-Raphson* method, is the most efficient. Geometrically, this method consists of extending a tangent line at a current point λ_i , until it crosses the zero axis, and then setting the next guess λ_{i+1} to the function value of that zero-crossing (see Figure A-1).

Figure A-1 - Illustration of Newton's Method



In this example, the first guess for λ, say λ_0 , finds the function at point **A**. From here the tangent is drawn to the axis, to the next guess, λ_1 , with $f(\lambda_1)$ at point **B**. From here another tangent is drawn to the axis, to λ_2 , with $f(\lambda_2)$ at **C**. The process continues until the difference between each successive λ is less than some small tolerance, or until $f(\lambda) = 0$.

While the convergence of the Newton algorithm is fast, its global convergence properties are poor. For this reason, we use a more fail-safe routine which uses a bisection technique to home in on the root whever the Newton algorithm would diverge, or whenever Newton is not reducing the size of the brackets rapidly enough.

The code that implements the Newton method is in the file IDFIXES.CPP, and the function is imnewton(). This function is, in turn, called by imsis(), which is also found in IDFIXES.CPP.

Sometimes, when using the import fixer, you may get a message that starts with the phrase: "Root is not bracketed in newton!" Some rather long text follows, which explains that the newton routine didn't converge, either because the maximum number of iterations was exceeded, or that getting the control to be satisfied would be impossible without some import shares being outside the range (0,1). In this case, you should check the fix with the domestic demands for the affected industries. For example, in the U.S., the imports to domestic demand ratio is already rather large for Shoes and for Radio and TV. If you tried to put a group fix on these two industries, and tried to increase imports substantially, it would probably force imports to be greater than demand. In this case, the newton algorithm cannot find a solution, and it will issue the above complaint.

This appendix closes with a simple example, using made up imports and domestic demand data. In Figure A-2 below, there are 10 sectors to be scaled. The initial (pre-scaling) imports are shown in the column labeled "imports", and domestic demand is

in the column labeled "DD". In the rest of the columns, the header label shows the control total that imports were scaled to, and the necessary value of λrequired to force imports to sum to the control total. The starting total for imports is 300. Note that to scale imports upwards, λ must be less than 1.0. To reach a control total of 400, λmust be .41. Note that imports of sector 1 almost doubled, from 20 to 36.6, whereas imports from sector 9 only increased from 70 to 80.4. To reach a control total lower than the unscaled total, λmust be greater than 1. For example, to reach 250, λmust be 1.52; to reach 50, λ must be 15.6!

Figure A-2. Some Scaling Exercises

	Imports	*DD*	$C = 400, \lambda = .41$	$C = 250, \lambda = 1.52$	$C = 100, \lambda = 6.69$	$C = 50, \lambda = 15.62$
1	20	90	36.6	14.2	3.7	1.6
2	30	80	47.2	22.7	6.6	3
3	40	60	49.7	34.1	13.8	6.8
4	50	70	60	43.5	19	9.7
5	10	30	16.4	7.4	2.1	0.9
6	10	40	17.8	7.2	1.9	0.8
7	15	30	21.2	11.9	3.9	1.8
8	35	50	42.4	30.3	12.9	6.5
9	70	90	80.4	62.8	30.9	16.5
10	20	40	28.2	15.9	5.2	2.4
Total	300	580	400	250	100	50

Figure A-1 below shows the share revision scaling function plotted out for various values of λ. Note how the function bows out more extremely when values approach 0 or extremely large numbers. When λ is equal to 1.0, the shares undergo no scaling revision. Also note that the figure below was created entirely in *G*.

Figure A-1. Share Revision Function Plotted in *G* for Various Values of