How to Make a Product-to-Product Input-Output Table

Clopper Almon

In making an input-output table, data on inputs is generally available not by the *product* into which they went but by the *industry* that used them. An *industry* is a collection of establishments with a common principal product. But besides this principal product, any one of these establishments may produce a number of secondary products, products primary to other four-digit SIC industries. Establishments classified in the Cheese industry may also produce ice cream, fluid milk, or even plastic moldings. Consequently, the Cheese industry may have inputs of chocolate, strawberries, sugar, plastic resins, and other ingredients that would appal a connoisseur of cheese. The inputs, however, are designated by what the product was, not by what industry made them. Similarly, data on the final demands, such as exports and personal consumption expenditure, is by product exported or consumed, not by the industry which made it. Thus, input-output matrices usually appear in two parts. The first part, called the Use matrix, has products in its rows but industries in its columns. The entries show the use of each product (in the rows) by each industry (in the columns.) The second, called the Make matrix, has industries in the rows and products in the columns; the entries show how much of each product was made in each industry.

How can we use these two matrices to compute the outputs of the various products and industries necessary to meet given final demands? An example will help us to visualize the problem. The first matrix below shows the Use matrix for a 5-sector economy with a strong concentration in dairy products, especially cheese and ice cream.

USE	Industries				
Products	Cheese	Ice cream	Chocolate	Rennet	Other
Cheese	0	0	0	0	0
Ice cream	0	0	0	0	0
Chocolate	4	36	0	0	0
Rennet	14	6	0	0	0
Other	28	72	30	5	0

We will call this matrix U. The use of chocolate in makings cheese and rennet in making ice cream alerts us to the fact that the columns are industries, not products. The Make matrix, shown below, confirms that cheese is being made in the ice cream industry and ice cream in the cheese industry.

MAKE	Products				
Industries	Cheese	Ice cream	Chocolate	Rennet	Other
Cheese	70	20	0	0	0
Ice cream	30	180	0	0	0
Chocolate	0	0	100	0	0
Rennet	0	0	0	20	0
Other	0	0	0	0	535
Total	100	200	100	20	535

This matrix shows that of the total output of 100 of cheese, 70 was made in the Cheese industry and 30 in the Ice cream industry, while of the total ice cream output of 200, 180 was in the Ice cream industry and 20 in the Cheese industry. We will need the matrix, M, derived from the Make matrix by dividing each cell by the column total. For our example, the M matrix is:

M	Cheese	Ice cream	Chocolate	Rennet	Other
Cheese	0.7	0.1	0.0	0.0	0.0
Ice cream	0.3	0.9	0.0	0.0	0.0
Chocolate	0.0	0.0	1.0	0.0	0.0
Rennet	0.0	0.0	0.0	1.0	0.0
Other	0.0	0.0	0.0	0.0	1.0

Now let us suppose that, in fact, cheese is made by the same recipe wherever it is made and ice cream likewise. The assumption is known in input-output parlance as the "product technology assumption.” If it is true and the matrices made well, then there exists a matrix, R, in which the first column shows the inputs into cheese regardless of where it is made, the second column shows the inputs into ice cream regardless of where it is made, and so on. Now the first column of U, U_1, must be $.70*R_1 + .10*R_2$,where R_1 and R_2 are the first and second columns of R, respectively. Why? Because the Cheese plants make 70 percent of the cheese and ten percent of the ice cream. In general,

$$U = RM' \tag{2.2.1}$$

where M' is the transpose of M. Well then, it is a simple matter to compute R as

$$R = U(M')^{-1}.$$

For our example, $(M')^{-1}$ is

1.5	-0.5	0.0	0.0	0.0
-0.2	1.2	0.0	0.0	0.0
0.0	0.0	1.0	0.0	0.0
0.0	0.0	0.0	1.0	0.0
0.0	0.0	0.0	0.0	1.0

and R works out to be

R	Cheese	Ice cream	Chocolate	Rennet	Other
Cheese	0	0	0	0	0
Ice cream	0	0	0	0	0
Chocolate	0	40	0	0	0
Rennet	20	0	0	0	0
Other	30	70	30	5	0

This R is very neat. All the rennet goes into cheese and all the chocolate goes into ice cream.

One might suppose that every statistical office producing input-output tables would produce such an R matrix by exactly this method. But, in fact, none does. Why not? Because there is a fly in the ointment. Suppose that the U matrix had been just slightly different, with 1 unit less of chocolate going into cheese as shown below and one less unit of rennet used in ice cream.

Alternative U	Cheese	Ice cream	Chocolate	Rennet	Other
Cheese	0	0	0	0	0
Ice cream	0	0	0	0	0
Chocolate	3	37	0	0	0
Rennet	15	5	0	0	0
Other	28	72	30	5	0

Then the R matrix would have been:

Impossible R	Cheese	Ice cream	Chocolate	Rennet	Other
Cheese	0.0	0.0	0.0	0.0	0.0
Ice cream	0.0	0.0	0.0	0.0	0.0
Chocolate	-1.7	41.7	0.0	0.0	0.0
Rennet	21.7	-1.7	0.0	0.0	0.0
Other	30.0	70.0	30.0	5.0	0.0

Anyone can see at a glance that this R is wrong, because it has negative numbers in it. In fact, experience shows that applying this method yields tables with lots of small negatives. It is not hard to see why. It is easy to forget to put into the Cheese industry the chocolate necessary for the ice cream it produces, or to put in too little. Wherever that happens, negatives will show up in the R matrix. Now one of the cardinal principles of statistical offices is not to publish numbers that are plainly wrong, so these R tables seldom see the light of day.

From this problem with the negatives most workers with input-output have come to the conclusion that the product-technology assumption is totally unworkable. *That conclusion is utterly false.* It is comparable to concluding that, because initial GDP estimates usually have to be revised, they are of no value whatsoever. We shall see that with a little gentle bending, the product-technology assumption works very well indeed. First, however, let us look at the alternative assumption currently used by most statistical offices.

This alternative assumption is known as the industry-technology assumption. It supposes that inputs are required in proportion to output and the proportions are the same for an industry's primary and secondary products. The so-called commodity-to-commodity matrix, C, derived from this assumption is

$$C = UM' .$$

For example, the Cheese column of C is $C_1 = .7U_1 + .1U_2$ because 70 percent of the product of the first industry is cheese and 10 percent of the product of the second industry is cheese. The result of applying this assumption to our example is:

C Indust. Tech.	Cheese	Ice cream	Chocolate	Rennet	Other
Cheese	0	0	0	0	0
Ice cream	0	0	0	0	0
Chocolate	6.4	33.6	0	0	0
Rennet	10.4	9.6	0	0	0
Other	26.8	73.2	30	5	0

This result is sheer nonsense. The original U matrix had 4 units of chocolate going into the

Cheese industry, which admittedly made some ice cream. Now this industry-technology product-to-product matrix asserts that *6.4 units of chocolate went into producing pure cheese*! Not into the Cheese *industry* but into the *product* cheese! And 9.6 units of rennet went into producing curdled ice cream! That this method has been officially recommended by international organizations and used by numerous statistical offices is little short of scandalous. Fortunately, the new SNA indicates a preference for the product-technology assumption, but provides no way to apply it without producing negative flows.

Fortunately, it is easy to rely mainly on the product-technology assumption, yet avoid the negatives, as we will now show.

We wrote the basic equation relating U, M, and R as

$$U = RM'.$$

It will prove convenient to rewrite this equation as

$$U' = MR'.$$

Using U'_i to denote the i^{th} column of U' and R'_i to denote the i^{tb} column of R', we can write

$$U'_i = MR'_i\ .$$

Notice that this is an equation for the distribution of product i in row i of the Use matrix as a function of M and the distribution of the same product in row i of the R matrix. We can simplify the notation by writing

$$\mathrm{u} = U'_i \quad \text{and} \quad r = R'_i\ ;$$

then the previous equation becomes

$$u = Mr$$

or

$$0 = -Mr + u$$

and adding r to both sides gives

$$r = (I - M)r + u\ .$$

Save in the unlikely case in which less than half of the industry's output is primary to it, the column sums of the absolute values of the elements of (I - M) are less than 1, and the convergence of the Seidel iterative process for solving this equation is guaranteed. We start this process with

$$r^{(0)} = u$$

and then define successive approximations by

$$r^{(k+1)} = (I - M)\, r^{(k)} + u\ .$$

To see the economic interpretation of this equation, let us write out the equation for the use of a product, say chocolate, in producing product j, say cheese:

$$r_j^{(k+1)} = u_j - \sum_{\substack{h=1 \\ h \neq j}}^{n} m_{jh} r_h^{(k)} + (1 - m_{jj}) r_j^{(k)} \qquad (2.2.2)$$

The first term on the right tells us to begin with the chocolate purchases by the establishments in the cheese industry. The second term directs us to remove the amounts of chocolate needed for making the secondary products of those establishments by using our present estimate of the technology used for making those products, $r^{(k)}$. Finally, the last term causes us to add back the chocolate used in making cheese in other industries. The amount of chocolate added by the third term is exactly equal to the amount stolen, via second terms, from other industries on account of their production of product j:

$$(1 - m_{jj}) r_j^{(k)} = \sum_{\substack{h=1 \\ h \neq j}}^{n} m_{hj} r_j^{(k)}$$

because

$$\sum_{h=1}^{n} m_{hj} = 1.$$

It is now clear how to keep the negative elements out of *r*. When the "removal" term, the second on the right of (2.2.2), is larger than the entry in the Use matrix from which it is being removed, we just scale down all components of the removal term to leave a zero balance. Then instead of adding back the "total-stolen-from-other-industries" term, $(1 - m_{jj})r_j$, all at once, we add it back bit-by-bit as it is captured. If a plundered industry, say Cheese, runs out of chocolate with only half of the total chocolate claims on it satisfied, we simply add only half of each plundering product's claim into that product's chocolate cell in the R matrix.

Applied to either of the U matrices given above, this method gives the "neat" R matrix with no rennet in ice cream and no chocolate in cheese. Inforum has used this method with little problem on many matrices for over 30 years. It never produces negative entries nor positive entries where U has a zero. In view of its very satisfactory performance and results, it should replace the implausible industry-technology matrix in all applications.

Here follows the C++ code for this method, using functions from BUMP, the Beginner's Understandable Matrix Package, for handling matrices and vectors.

```
#include <stdio.h>       // for printf();
```

```
#include <math.h>        // for abs()
#include "bump.h"
int purify(Matrix& R, Matrix& U, Matrix& M, float toler);

void main(){
     Matrix Use(5,5), Make(5,5), R(5,5);
     Use.ReadA("Use.dat");
     Make.ReadA("Make.dat");
     purify(R,Use,Make,.000001);
     R.Display("This is R");
     tap();
     printf("\nEnd of calculations.\n");
     }

/* Purification produces a produt-to-product (or Recipe) matrix R 
from
     a Use matrix U and a Make matrix M.  M(i,j) shows the 
fraction of
     product j made in industry i.  U(i,j) shows the amount of 
product i
     used in industry j.  The product-technology assumption leads 
us to
     expect that there exits a matrix R such that U = RM'.  If, 
however, we
     compute R = U*Inv(M') we often find many small negative 
elements in
     R.  This routine avoids those small negatives in an 
iterative process.
     */
```

```
int purify(Matrix& R, Matrix& U, Matrix& M, float toler){
     int row, i, j, m, n, iter, imax;
     const maxiter = 20;
     float sum,rob,scale,dismax,dis;
     n = U.rows();  // n = number of rows in U
     m = U.columns(); // m = number of columns in U
     Vector C(m), P(m), Flow(m), Discrep(m);
     // Flow is row of U matrix and remains unchanged.
     // P becomes the row of the purified matrix.
     // C is the change vector at each iteration.
     // At the end of each iteration we set P = Flow + C, to
start the next iteration.


     // Purify one row at a time
     for(row = 1; row <= n; row++){
          C.set(0.); // C, which will receive the changes, is
initialized to zero.
          // P = Flow + C will be the new P.
          pulloutrow(Flow,U,row);
          P = Flow;
          iter = 0;
          start: iter++;
          for(j = 1; j<=m; j++){
               // Calculate total claims from other industries on
               // the inputs into industry j.
               sum = 0;
               for(i = 1; i <= m; i++){
                    if(i == j) continue;
                    rob = P[i]*M(j,i);
                    sum += rob;
                    C[i] += rob;
                    }
               // Did we steal more from j than j had?
               if (sum > Flow[j]){
                    // scale down robbery
                    scale = 1. - Flow[j]/sum;
                    for(i = 1; i <= m; i++){
                         if(i == j) continue;
                         C[i] -= scale*P[i]*M(j,i);
                         }
                    sum = Flow[j];
                    }
               C[j] -= sum;
               }
          // Check for convergence
          imax = 0;
          dismax = 0;
          for(i = 1; i <= m; i++){
               dis = fabs(P[i] - Flow[i] - C[i]);
               Discrep[i] = dis;
               if(dis >= dismax){
                    imax = i;
                    dismax = dis;
                    }
               }
          P = Flow + C;
          C.set(0);
          if(dismax > toler){
               if(iter < maxiter) goto start;
               printf("Purify did not converge for row %d. Dismax
= %7.2f. Imax = %d.\n",
                    j,dismax,imax);
               }
          putinrow(P,R,row);
          }
     return(OK);
     }
```