The Craft of Economic Modeling,

Part III. Multisectoral Models

Contents

14. Input-Output in the Ideal Case 2
    1. Input-Output Flow Tables 2
    2. Input-Output Equations. The Fundamental Theorem. 6
    3. Introduction to Input-Output Computing 9
    4. Iterative Solutions of Input-output Equations 25
    5 The Seidel Method and Triangulation 30
    6. Introduction to Interdyme 31
    7. Matrix Tools in Interdyme 38
    8. TINY with Endogenous Consumption 40
    9. A Historical Note 43

15. Matrix Balancing and Updating - the RAS Method 45
    1. The RAS Algorithm 45
    2. Convergence of the Algorithm 45
    3. Preliminary Adjustments Before RAS 48

16. Trade and Transportation Margins and Indirect Taxes 50
    1. Trade and Transportation Margins 50
    2. Indirect Taxes, Especially Value Added Taxes 51

17. Making Product-to-Product Tables 52
    1. The Problem 52
    2. An Example 53
    3. The No-Negatives Product-Technology Algorithm 57
    4. When Is It Appropriate to Use This algorithm? 58
    5. A Brief History of the Negatives Problem 60
    6. Application to the U.S.A Tables for 1992 61
    7. The Computer Program 65

Chapter 14. Input-Output in the Ideal Case

1. Input-Output Flow Tables

Multisectoral models begin from an accounting of the flows of goods and services among various industries of the economy. Table 1 shows a simple interindustry accounting, or input-output flow table, for an imaginary but not unrealistic eight-sector economy. The selling industries are listed down the left side of the table. The last, abbreviated to "GovInd," is "Government Industry", a fictitious industry which in this table simply supplies the government with the services of its own employees. Below these come the classes of factor payments, here Depreciation, Labor compensation, Capital income (such as interest, profits, rents, or proprietor income), and Indirect taxes (such as property taxes, sales taxes, and excise taxes as on alcohol, tobacco, and gasoline). Note the similarity of these categories of factor payments to the categories of national income. Their sum is the row Value added. Across the top of the table the same eight industries are listed as buyers of products. Here they are followed by columns corresponding to the principal divisions of the "product side" of the national accounts, namely

Con Personal consumption expenditure
Gov Government purchases of goods and services
Inv Investment
Exp Exports
Imp Imports (as negative numbers)

In input-output terms, these are the final demand columns. The next-to- last column, labeled FD for "Final Demand," shows their sum. It is shaded to emphasized that it is derived by summing other columns. The next last column, also shaded, is the sum of all the (non-shaded) elements row.

Across each row of the table are shown the sales of that industry to each of the industries and final demand columns. Thus, the 100 in the Agriculture row and Manufacturing (Mfg) column means that Agriculture sold 100 billion dollars (bd) of products to Manufacturing in the year covered by this table. Typical sales here are grains to milling, live animals to meat packing, or fruits and vegetables to plants which can or freeze them. The 15 in the Personal consumption (Con) column of the same row means that Agriculture sold 15 bd of products directly to households during the year. These sales are primarily fresh fruits and vegetables and eggs. In the table shown here, which is said to be in producer prices, they are recorded at the price the farmer received for them. These products are not necessarily bought at the farm gate, however, for going through wholesale and retail trade channels does not change the industry of origin of a product; going through a manufacturing process does. Thus, an orange sold as an orange to she who eats it appears as a sale from Agriculture to Personal consumption, despite the fact that it went through a store. Another orange that was turned into frozen orange juice appears first as a sale from Agriculture to Manufacturing at the price received by the farmer. It then reappears as a sale from Manufacturing to Personal consumption at the manufacturer's price. But the price paid by the ultimate consumer is neither the price received by farmer in the first case nor by the manufacturer in the second. Where is the difference, the commercial margin? In this table, it is in the sales of Commerce to Personal consumption expenditure. Transportation margins are handled similarly. Tables made with this pricing convention are said to be "in producer prices". We shall look at other ways of handling the problem of margins in Chapter 2.

Table 1. An Input-Output Flow Table

Buyer / Seller	Agri-cult.	Mining	Gas Elec	Mfg	Com-merce	Trans-port	Ser-vices	Gov Ind	Con	Gov	Inv	Exp	Imp	FD	Row Sum
Agriculture	20	1	0	100	5	0	2	0	15	1	0	40	-20	36	164
Mining	4	3	20	15	2	1	2	0	2	1	0	10	-10	3	50
Gas&Electric	6	4	10	40	20	10	25	0	80	10	0	0	0	90	205
Mfg	20	10	4	60	25	18	20	0	400	80	200	120	-170	630	787
Commerce	2	1	1	10	2	3	6	0	350	10	6	10	0	376	401
Transport	2	1	5	17	3	2	5	0	130	20	8	5	0	163	198
Services	6	3	8	45	20	5	20	0	500	40	10	30	-20	560	667
GovInd	0	0	0	0	0	0	0	0	0	150	0	0	0	150	150
Intermediate	60	23	48	287	77	39	80	0							614
Deprec.	11	5	60	130	35	40	25	0							306
Labor	65	20	21	260	140	97	485	150							1238
Capital	20	2	56	60	40	12	59	0							249
Indirect tax	8	0	20	50	109	10	18	0							215
Value added	104	27	157	500	324	159	587	150							2008
ColSum	164	50	205	787	401	198	667	150	1477	312	224	215	-220	2008	

As we look down the column for an industry, we see all the products which it needs for making its own. In the Agriculture column, we see first of all 20 bd from Agriculture itself. These are sales primarily of feed grains to animal husbandry, but include also sales of seed, hay, manure, and other products. These sales within the industry are common and are referred to in input-output jargon as "diagonals" because they appear on the main diagonal of the table. Further down the Agriculture column we see 4 bd for Mining, primarily crushed limestone, but also some coal. The 20 bd spent on Manufacturing bought gasoline, fertilizers, and pesticides. The 2 bd spent on Commerce were trade margins on these manufactured products. The 2 bd spent on Transport included transportation margins on the products of the other industries as well as costs incurred by the farmer in getting products to market. The purchases from Services includes the services of veterinarians, lawyers, and accountants. All the purchases of the industries from each other are called "intermediate" purchases because they do not go directly to the final user but are "mediated" by other industries. The sum of the intermediate purchases by each industry are in the row labeled "Intermediate" and shaded, as before, to show that it is derived by adding other entries in the table. Many tables also have a total intermediate column; our table omits it for the simple reason that it would not fit on the page.

Below the "Intermediate row" are the value-added rows. We find that Depreciation of equipment came to 11 bd. Labor received 65 bd. (In our imaginary economy, we imagine that proprietor income has been divided between labor and capital income. In most actual tables, it will be shown separately or classified as capital income.) The 20 bd of capital income includes interest payments, coporate profits, and capital's portion of proprietor income. The 8 bd of Indirect taxes is mostly property taxes.

Now precisely because the Capital income row of value added -- which includes both corporate profits and proprietor income -- is the total of sales minus the total of expenses, the column sum for each industry is equal to its row sum. For example, the row sum of Agriculture is 164 and the column sum (of the unshaded entries) is 164, and so on for all eight industries. This fact has a remarkable consequence which is the cornerstone of national accounting, namely that the sum of all the value-added entries is equal to the sum of all the final demand entries. In our table, each of these groups of entries is surrounded by a double line and each adds to 2008. Why is the total the same? Since the sum of each of the eight industry rows, say R, is equal to the sum of the corresponding column, the sum of all eight rows, 2622, is equal to the sum of all eight columns, say C, which is also 2622. Thus we have with R = C. But the total of the final demands, D, is R minus the total of the intermediate flows, say X, or D = R - X. Likewise, the total value added, V, is C, the sum of all the industry columns, less the sum of that part of them which is intermediate, or V = C - X. But R = C implies that R - X = C - X or D = V. Naturally, this D or V has a name, and that name is Gross Domestic Product. We have thus proved the fundamental identity of national accounting: Gross Domestic Product (GDP) is the same whether measured by the products that go to final demand or by the income which goes to factors. In our table, this identity appears in the fact that the sum of the FD column, 2008, is the sum of the Value added row, also 2008, which is the GDP of this economy. Arrayed in format of national accounts, our economy would appear as in Table 2.

Table 2. The Income and Product Account

Gross domestic product	2008	Gross domestic product	2008
Personal Consumption	1477	- Depreciation	306
Investment	224	= Net domestic product	1702
Exports	215	- Indirect taxes	215
Imports	-220	= National income	1487
Government purchases	312	Labor income	1238
		Capital income	249

Before leaving Table 1, we must make a fundamental point about it. With one small exception, the table makes sense in physical units. We can measure the output of Agriculture in bushels, that of Mining in tons, that of Gas and Electricity in BTU's, Transport in ton-miles, Labor in worker hours, Capital income in ounces of gold, and so on. Detailed tables in physical terms have in fact been made for China. Wassily Leontief, maker of the first input-output table, used to often insist in seminars that any calculations had to make sense in physical terms.

The small exception, however, is important: the column sums of a table in physical terms are utterly meaningless since all the elements are in different units. Naturally, the row totals -- which are meaningful -- do not equal the meaningless totals of the corresponding columns. This point would seem so obvious as to be not worth making were it not for the fact that it is often forgotten, precisely by the makers of input-output tables. For if a table is made in the prices of some year other than the year to which it refers, it is essentially in physical units. Thus, we can make a table for 2000 in 1980 prices, where the physical measure in each row is "one 1980 dollar's worth" of the product. In other words, the physical unit for each product is how much of it one dollar would buy in 1980. For any product for which a price index can be made, 2000 dollar amounts can be converted into 1980-dollar physical units by the price index. For value added, since there is no very natural unit, one can simply deflate all of the value-added cells by the GDP deflator. The total real value added will then be the same as total real final demand. One can have in this way a perfectly sensible, meaningful table. *But its column sums are meaningless and certainly do not equal the corresponding row sums.*

Unfortunately, some table makers have disregarded this fact and have simply forced the value added in each industry of such a table to equal the difference between the row sum of the industry and the sum of the intermediate inputs into it. The results make as much sense as saying that five squirrels minus three elephants equals two lions. The arithmetic is right but the units are crazy.

This practice is called "double deflation" because first the outputs are deflated and then the purchased inputs deflated and subtracted from the deflated output to obtain a mongrel, mixed-up-units number, possibly positive but also possibly negative, mistakenly alleged to be a measure of "constant-price value added". It is, in fact, what would have been left over for paying primary factors, had producers, contrary to economic theory, gone right on producing with the previous period's inputs after prices have changed. That is certainly no measure of "real value added," for it is not, in all probability, what producers did. The error would perhaps be easier to see if labor input, for which we have some measures of cost, were considered as an intermediate input and indirect taxes were simply subtracted in current prices from output. The double-deflation procedure should then give a measure of "real capital income." In such a table, the deflators for capital income would be different in different industries. The residuals might well be negative, especially if there were a few years between the two periods. Trying to deflate the difference between two numbers that are very close together by deflating each of

the two numbers by different deflators and then taking the difference between the two deflated items is simply asking for trouble.

The nonsense involved in double deflation is often masked by the taking the time periods of the tables close together and "chaining" the index, so that negative values are unlikely. But nonsense in small increments is still nonsense. Unfortunately, this nonsense is compounded by the fact that these procedures are sanctioned by international statistical standards, and many statistical offices engage in them. Economists have made matters worse by taking these mixed-units numbers as measures of "real" product in studies of productivity.

As far as I am aware, there is no satisfactory way of measuring real productivity at the individual industry level, precisely because industries cooperate with one another in production, and how they do so changes. In one year, for example, the "television set industry" is a collection of plants that make the cabinets, the tubes and the electronics, and assemble the sets. In a later year, the industry has become assembly plants that buy cabinets, tubes, and electronics and assemble them. Clearly, changes in sales (even in constant prices) divided by labor input in worker hours in this one industry is not an appropriate measure of productivity increase. Rather, changes in "productivity" in this case is meaningful only as applied to how much labor and capital is required *by the whole economy* to produce a television set. We shall see how it can be meaningfully calculated. The meaningful, correct calculation has nothing whatever to do with double deflation. But the quest to allocate the changes in whole-economy productivity for particular products to individual industries is a search for a nonexistent – and superfluous – El Dorado.

2. Input-Output Equations. The Fundamental Theorem.

An input-flow table describes an economy in a particular year. Its greatest value, however, lies in the ability it gives us to answer the question What would the outputs, value added, and intermediate flows have been had the final demands been different? To answer that question in the simplest possible way, we must assume that the ratio of each input into an industry to that industry's output remains constant when the final demands are changed. These ratios are known as the "input-output coefficients," and may be defined by

$$a_{ij} = x_{ij} / q_j$$

where x_{ij} is the flow from industry i to industry j in Table 1.1 and q_j is the output of industry j, that is, it is the sum of row j or column j in the same table. For example,

$$a_{1,4} = 100/787 = 0.12706$$

Table 3 shows the complete matrix of these input-output coefficients corresponding to Table 1.

If we are willing to suppose that these coefficients remain constant as the final demand vector changes, then for any vector of final demands, f, we can calculate the vector of industry outputs, q, from the equation

(14.2.1) $$q = Aq + f$$

where A is the matrix of input-output coefficients in Table 3. If we happen to choose as f the column vector of final demands in Table 1, (the first eight elements of the FD column: (36,3,90, ..., 150)'), then q should be the column vector of industry outputs of Table 1 (the vector of row sums of the eight industry rows: (164,50,205,...,150)'). For other values of f, of course, we will find other values of q.

Table 3. Input-Output Coefficients

	Agric	Mining	Gas&Elec	Mfg	Com	Trans	Serv	GovInd
Agriculture	0.12195	0.02000	0.00000	0.12706	0.01247	0.00000	0.00300	0.00000
Mining	0.02439	0.06000	0.09756	0.01906	0.00499	0.00505	0.00300	0.00000
Electricity	0.03659	0.08000	0.04878	0.05083	0.04988	0.05051	0.03748	0.00000
Manufacturing	0.12195	0.20000	0.01951	0.07624	0.06234	0.09091	0.02999	0.00000
Commerce	0.01220	0.02000	0.00488	0.01271	0.00499	0.01515	0.00900	0.00000
Transportation	0.01220	0.02000	0.02439	0.02160	0.00748	0.01010	0.00750	0.00000
Services	0.03659	0.06000	0.03902	0.05718	0.04988	0.02525	0.02999	0.00000
GovInd	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

One way of solving (1.2.1) is to rewrite it as

$$(I - A)q = f$$

or

$$q = (I - A)^{-1}f.$$

The matrix of $(I - A)^{-1}$ on the right of this equation is known as the Leontief inverse of the A matrix. For our example, it is shown in Table 4. Its elements have a simple meaning. Element (i,j) shows how much of product i must be produced in order to produce one unit of final demand for product j. This interpretation is readily justified by taking f to be a vector of zeroes except for a 1 in row i. Then q will be the ith column of $(I - A)^{-1}$, and its jth element will show exactly how much of product j will have to be produced in order to supply exactly one unit of i to final demand. In our example, in order to supply one unit of Agricultural product to final demand, .1691 units of Manufacturing must be produced. Note that, in the example, all elements of the Leontief inverse are non-negative. In view of the economic interpretation, that result is hardly surprising. Later in this chapter, we will show mathematically that the Leontief inverse from an observed *A* matrix is always non-negative.

Table 4. The Leontief Inverse $(I - A)^{-1}$

1.1647	0.0620	0.0107	0.1634	0.0263	0.0165	0.0096	0.0000
0.0405	1.0830	0.1126	0.0352	0.0144	0.0150	0.0092	0.0000
0.0617	0.1137	1.0683	0.0748	0.0623	0.0641	0.0452	0.0000
0.1691	0.2530	0.0538	1.1201	0.0791	0.1091	0.0396	0.0000
0.0184	0.0276	0.0093	0.0185	1.0077	0.0180	0.0106	0.0000
0.0210	0.0319	0.0304	0.0297	0.0120	1.0151	0.0102	0.0000
0.0604	0.0911	0.0548	0.0791	0.0612	0.0379	1.0368	0.0000
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000

We may also ask how much of a primary resource, such as Labor or Capital, would be needed for the production of a given final demand. We may define the resource coefficients similarly to the input-output coefficients by

$$r_{ij} = y_{ij}/q_j$$

where y_{ij} is the payment to factor i by industry j. For example, from Table 1, $y_{2,4}$, the payment to resource 2, Labor, by industry 4, Manufacturing, is 360. If we denote by R the matrix of the r_{ij}, then the vector of total payments to each resource for an output vector *q* is *Rq*, and for a final demand vector, f, it is $R(I - A)^{-1}f$.

If we now think of each row of this matrix as a row vector and sum these vectors -- a process which makes sense if all the rows are measured in monetary values in the prices of the year of the table -- we get a row vector, *v*, of value-added per unit of output. Just as previously we asked how outputs, *q*, would change if *f* changed while *A* remains constant, we can now ask how prices, *p*, would change if *v* changed while *A* remains constant. The row vector *p* must satisfy the equations

(14.2.2) $$p = pA + v.$$

These equations state simply that the price of a unit of each product is equal to the cost of all products used in producing that unit (the first term on the right) plus value-added per unit produced. Just as the equations (14.2.1) provide the fundamental connection in multisectoral models between final demands and outputs, so these equations provide the fundamental connection between unit value added and prices. If we want to know how specific changes in productivity or in wages in one or several industries will affect prices in all industries, these equations are the key. If we calculate the prices for *v* vector given in the table, we should find that all prices are equal to 1.

There is, furthermore, a relation of fundamental importance between the solutions of the two sets of equations. Namely, given any *A*, *f*, and *v*, the *q* and *p* which satisfy $q = Aq + f$ and $p = pA + v$ also satisfy

(14.2.3) $$vq = pf.$$

This equation says that the value of the final demands evaluated at the prices implied by equations (14.2.2) are equal to the payments to the resources necessary to produce those final demands by (1.2.1). Thus, if our outputs and prices satisfy the required equations, we can be certain that GDP measured by the final demands in current prices will be equal to the GDP measured by the payments to resources (or factors) in current prices. If we build these equations into our models, we can be certain that the models will satisfy the basic accounting identity in current prices. This relation may well be called the fundamental theorem of input-output analysis. Fortunately, it is as easy to prove as it is important, and you shouldproduce your own proof. If you need help desperately, look in the upside-down footnote. [1]

3. Introduction to Input-Output Computing

The simple calculations described in the previous section can be done with G through the use of a new sort of data bank known as a VAM (Vectors And Matrices) file. As the name suggests, this type of data bank holds time series of vectors and matrices. G has commands which can add, subtract, multiply, and invert matrices and add and subtract vectors and multiply them by matrices. Thus, the operations discussed so far, and several others, can easily be performed in G. But when we go on to build multisectoral models involving input-output matrices in combination with regression equations, then running and optimizing such models requires a further step, the use of the Interdyme modeling system, which is the extension to multisectoral models of “Build” and “Run” sequence already familiar from Parts 1 and 2 of this book. In this section, we will stay with what can be done in G with the VAM file. In section 5, we introduce the Interdyme system.

---

1. Multiplying (1.2.1) on the left by p gives
(A) pq = pAq + pf
while multiplying (1.2.2) on the right by q gives
(B) pq = pAq + vq.
Subtracting (B) from (A) gives
0 = pf - vq
so
pf = vq.

A VAM file differs in two important respects from the G data banks we have worked with so far:

(1) In the standard G bank, all elements are the same size, namely a time series of a single variable beginning at the beginning of the data bank and extending over the number of observations in the bank, as specified by the G.cfg file. In VAM files, elements are time series of vectors or matrices of various dimensions. As in the standard G bank, all time series are the same length.

(2) In standard G banks, we can create new series as we work, for example, with *f*, *fex*, or *data* commands. In VAM files, we buy the flexibility of having elements of various sizes by specifying at the outset the contents of the file, that is, the names and dimensions of each vector or matrix in the bank along with the names of the files giving the titles of the row or columns of the vector or matrix. One might suppose that it is a bit of nuisance to have to specify this structure of the VAM file at the outset. In practice, however, this need to prespecify structure proves a useful discipline in building complex models. If, as a model evolves, it becomes necessary to revise the specification of the VAM file, it is easy to copy the contents of the old file into the new, enlarged file. This specification is accomplished by a file usually named VAM.CFG.

We can illustrate the use of the VAM file and some new G commands for making some simple calculations with the input-output table presented in section 1 of this chapter, which will assume is for the year 1995 . The box below shows the VAM.CFG file for this model, which we will call TINY. It and all the files used in this chapter are in the file TINY.ZIP. I suggest that you make a directory (also called a folder), copy TINY.ZIP into it, and unzip it.

The first line in VAM.CFG gives the beginning and ending years for the VAM file. The next line, the one beginning with a #, is a comment to clarify the structure of the file. Comments beginning with a # can be placed anywhere in the file. Then come free-form lines giving

1. The name of the element
2. Its number of rows
3. Its number of columns
4. The maximum number of lags with which a vector occurs in the model or a *p* if the matrix is a "packed matrix" – a device useful in large-scale models.
5. The name of a file containing the names of the rows of a vector or matrix
6. The name of a file containing the names of the columns of a matrix
7. A # followed by a brief description of the element.

As far as the computer is concerned, these lines are free format; all that is needed is one or more spaces between each item on a line. But this is a file also read by humans, so putting in spaces to make the items line up in neat columns is also a good idea. The box below shows the vam.cfg file for the TINY model based on example of section 1 of this chapter.

To create a vam file from a vam configuration file the command in G is

vamcreate <vam configuration file> <vam file>

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*, thus:

VAM.CFG File for the TINY Model

```
1995 2010
#name   Rows Cols Lags Row names Column names Comment
FM         8    8  0  sectors.ttl sectors.ttl #Input-output flow matrix
AM         8    8  0  sectors.ttl sectors.ttl #Input-output coeff matrix
LINV       8    8  0  sectors.ttl sectors.ttl # Leontief inverse
out        8    1  0  sectors.ttl # Output
pce        8    1  0  sectors.ttl # Personal consumption expenditure
gov        8    1  0  sectors.ttl # Government spending
inv        8    1  0  sectors.ttl # Investment
ex         8    1  0  sectors.ttl # Exports
im         8    1  0  sectors.ttl # Imports
fd         8    1  0  sectors.ttl # Total final demand
dep        8    1  0  sectors.ttl # Depreciation
lab        8    1  0  sectors.ttl # Labor income
cap        8    1  0  sectors.ttl # Capital income
ind        8    1  0  sectors.ttl # Indirect taxes
depc       8    1  0  sectors.ttl # Depreciation coef
labc       8    1  0  sectors.ttl # Labor income coef
capc       8    1  0  sectors.ttl # Capital income coef
indc       8    1  0  sectors.ttl # Indirect taxes coef
x          8    1  0  sectors.ttl # Working space
y          8    1  0  sectors.ttl # Working space
```

```
vamcr vam.cfg hist
```

At this point, the newly created vam file has zeroes for all its data. We will now see how to put data into it and work with that data. 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 *v* may be used to designate banks. However, it is generally a good practice to leave *a* as the G bank which was initially 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 must already 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.

The usual ways to introduce data into a VAM file are with the *matin* command for matrices and the *vmatdat* command for vectors. We can illustrate them with the data for TINY from section 1.

The Flows.dat File for Introducing the Input-Output Flow Matrix into the VAM File

```
matin FM  1995 1 8 1 8  15
#            Agricul Mining Elect  Mfg Commerce Transp Services Govt
Agriculture       20      1     0  100        5      0        2    0
Mining             4      3    20   15        2      1        2    0
Electricity        6      4    10   40       20     10       25    0
Manufacturing     20     10     4   60       25     18       20    0
Commerce           2      1     1   10        2      3        6    0
Transportation     2      1     5   17        3      2        5    0
Services           6      3     8   45       20      5       20    0
Government         0      0     0    0        0      0        0    0
```

The *matin* command on the first line is followed by the matrix name in VAM.CFG file, then by the year to which the matrix belongs, then the number of the first row and last row in the following rectangle of data, then the number first column and last column in the rectangle. (In the present case, the rectangle is the whole table; but this ability to read in a table rectangle-by-rectangle is quite useful for reading tables scanned from printed pages.) The last number on the *matin* line is the skip count, the number of characters to be skipped at the beginning of each line. These characters usually give sector names or numbers. The # in the first position marks the second line as a comment. Then come the data; each line is in free format after the initial skip. (Do not use tabs in characters which are to be skipped; the tab character will be counted as just one character.)

The FD.dat file shown below illustrates the introduction of vectors, in this case, the final demands. The *vmatdat* command is rather flexible; it can introduce a number of vectors for one year or one vector for a number of years. The vectors can be the rows or the columns in the following rectangle of data. Because of this flexibility, we have to tell the command how to interpret the rectangle of data. The command must therefore by followed by a *c* or an *r* to indicate whether the vectors appear as columns or rows in the following rectangle of data. Here, the vectors are clearly columns. The next number is the number of vectors in the rectangle; here 5. Next is the number of years represented in the

The FD.dat File for Introducing the Final Demands into the VAM File

```
vmatdata c 5 1 1 8 15
1995          pce  gov   inv     ex      im
#           PersCon   Gov Invest Exports Imports
Agriculture      15     1     0    40     -20
Mining            2     1     0    10     -10
Electricity      80    10     0     0       0
Manufacturing   400    80   200   120    -170
Commerce        350    10     6    10       0
Transportation  130    20     8     5       0
Services        500    40    10    30     -20
Government        0   150     0     0       0
```

rectangle. Here it is 1, for the columns are different vectors for the same year. (Either the number of vectors or the number of years must be 1.) The next two numbers are the first and last element numbers of the data in the rectangle, and the last is the skip count as before. Since this command is introducing several vectors for *one* year, that year is specified at the beginning of the next line, and the names of the vectors follow it. (If we were introducing data for one vector for several years, the vector name would be in the first position on this line, followed by the year numbers.)

The value-added rows are introduced by the *vmatdat* command and data shown in the box below.

The VA.DAT File for Introducing the Value-added Vectors

```
vmatdata r 4 1  1 8 15
1995  dep lab cap ind
#                1      2      3      4      5      6      7      8
Depreciation    11      5     60    130     35     40     25      0
Labor           65     20     21    260    140     97    485    150
Capital         20      2     56     60     40     12     59      0
Indirect tax     8      0     20     50    109     10     18      0
```

Here, finally, are the G commands to create the VAM file and load the data into it:

```
# Create and load the VAM file for TINY
vamcreate vam.cfg hist
vam hist b
dvam b
# Bring in the intermediate flow matrix
add flows.dat
# Bring in the final demand vectors
add fd.dat
# Bring in the value added vectors
add va.dat
```

These and the following commands to G for making the calculations described in this section are in the file Gmodel.

Now let us look at some of the data we have introduced by displaying them in a grid on the screen. The command

```
show FM   y   1995
```

will show in a spreadsheet-like grid the FM matrix, the flow matrix for the year 1995. To adjust the default column width and the number of decimal places in the display, click the Options menu item. Not only does this display look like a spreadsheet display, it also works like one in that you can copy and paste between data from one to the other.

To look at a row, say row 2, of the FM matrix for all years of the VAM file, the command is

```
show FM r 2
```

while to show column 5for all years, the command is

```
show FM c 5
```

Thus, in showing a matrix, we have to choose among showing the whole matrix for one year and showing one row or column for all years. The choice is indicated by the letter – a *y, r* or *c* – following the matrix name.

Showing vectors is simpler because we do not have to make this choice; we just name the vector and get all values for all years. Here are two examples

```
show ind     # Display the indirect tax vector
show b.pce   # Display the personal consumption expenditure vector
```

The second of these examples shows that the *show* command allows us to specify by the bank letter followed by a dot the bank from which the item is to be shown.

Now that we have read in the data and displayed it to check that it was accurately read, we can begin to compute. To calculate the input-output coefficient matrix, we need *out*, thevector of outputs by industry. It was not read in, but it can be computed by summing the rows of the FM matrix and then adding to this row sum the final demand columns. Here are the two commands and the *show* command to see the result:

```
# Add up the intermediate rows
getsum FM r out
# Add on the final demand vectors to get total output
vc out = out+pce+gov+inv+ex+im
show b.out
```

We are now ready to copy the flow matrix, stored in FM, to AM and then convert it to input-output coefficients by dividing each element of each column by the corresponding element of the *out* vector. We do the copy with the with the *mcopy* command, for "matrix copy." The general form of the mcopy command to copy matrix or vector A from bank x to element B in bank y is

```
mcopy  y.B [=] x.A
```

The = sign is optional but is useful reminder of which way the copy is going. The y. is optional if y is the default VAM file, and the same is true for the x.. Since this copy and these calculations need be done only for one year, the first, 1995, we first set the fdates so that the *mcopy* and *coef* commands work only on the years from 1995 to 1995 (which is to say, only for 1995). Here are the commands

```
# Copy intermediate flows to AM and convert to coefficients
fdates 1995 1995
mcopy b.AM = b.FM
coef AM out
show AM y 1995
# Create value-added coefficient vectors.
vc depc = dep/out
vc labc = lab/out
vc capc = cap/out
vc indc = ind/out
# Set fdates back to the entire range of the VAM file.
fdates 1995 2010
```

With the input-output coefficients calculated, we can now go on to illustrate finding the Leontief inverse, calculating outputs from exogenous forecasts of final demands, calculating value-added components, and displaying, graphing, and making tables of the results. We will first copy the input-output coefficient matrix and the value-added coefficient vectors from 1995 to the other years out to 2010. We can conveniently do this with G's *index* command. This command is used to move all elements of a vector or matrix in the default VAM file forward or backward in proportion to a guide series. Its general form is:

index <base year> <guide series> <matrix or vector>

It operates over the range specified by the current value of the fdates. Since we just want to copy the coefficients to all the years, our guide series will be simply a series of 1's, which we shall call *one*. Here are the commands

```
# Copy the 1995 AM matrix into 1996 -  2010
dfreq 1
f one = 1.
index 1995 one AM
index 1995 one depc
index 1995 one labc
index 1995 one capc
index 1995 one indc
show AM c 1
```

The last command displays in a grid the first column of the AM matrix for all the years; all columns of this display should, of course, be identical. For purposes of our illustration, we will let *AM* remain constant in all years. The final demands, however, we will move in a slightly more interesting way. We will have all of them except investment grow at a steady 3 percent per year. Investment will also have one component growing at this same rate but added to it – to make the results more interesting to view – will be a sine curve with a period of 2π years. Here are the commands to move the final demand vectors.

```
# Create a time trend
f time = @cum(time,one,0)
# Make all final demand vectors grow by 3 percent per year
f g03 = @exp(.03*(time-1))
gr  g03
# Create waves, the guide series for the investment vector
f  waves = g03 + .3*@sin(time-1)
gr  waves
index 1995 g03 pce
index 1995 g03 gov
index 1995 waves inv
index 1995 g03 ex
index 1995 g03 im
show inv
```

To add up the components of final demand to the total, we use the *vc* (for vector calculation) command. It can add up any number of vectors to get a total. Here are the commands.

```
# Add up the final demands
vc fd = pce+gov+inv+ex+im
show fd
```

We are now going to ignore the fact that the *AM* matrix is the same in all years – we could have changed it had we wanted to – and take its Leontief inverse in all years in the fdates range. The command

linv <square matrix> [year]

converts the square matrix into its Leontief inverse. For example,

linv A

converts A into $(I - A)^{-1}$. We then multiply this inverse by the final demand vector to compute the output vector. The *linv* command works over the fdate range unless the optional *year* argument is present.

```
# Take the Leontief inverse of the A matrix
mcopy LINV = AM
linv LINV
show LINV y 1995

# Compute total outputs
vc out = LINV*fd
show b.out
```

With the outputs known, we can compute the implied value-added of each type by each industry with the following commands. In them, the *vc* command will recognize that the dimensions of the vectors on the right are such that element-by-element multiplication makes sense and perform it.

```
# Compute Value added
# The following are element-by-element multiplication
vc dep = depc*out
vc lab = labc*out
vc cap = capc*out
vc ind = indc*out
show lab
```

As we went along, we showed results in spreadsheet-like grids to check that our answers were generally reasonable. Now we need to graph the results. In doing so, we use the fact that elements of vectors in a VAM file can be referred to in G simply by the name of the vector followed by a numeral. We can graph the second element of the *out* and *pce* vectors from the VAM file assigned as bank bould do so with a number of graph commands like

gr b.out2 b.pce2

but we can get a lot more graphs more quickly by use of G's *fadd* command. The name *fadd* is a contraction of "file-directed add command." It works with text substitution in a way that is very convenient in working with multisectoral models. The general form is

fadd <command file> <argument file>

In our case, the "command file" will be the following file, named GRAPHS.FAD:

```
vr 0
ti %3 %5
subti Output and Final demand
gname out%3
gr b.out%3 b.fd%3
subti Depreciation,Labor income, Capital income, Indirect taxes
```

```
gname va%3
gr b.dep%3 b.lab%3 b.cap%3 b.ind%3
ti
subti
```

and the argument file will be the same SECTORS.TTL file which we used for supplying row and column names for the matrices and vectors in the VAM file, namely:

```
Agricul     ;1   e   "Agriculture"
Mining      ;2   e   "Mining and quarrying"
Elect       ;3   e   "Electricity and gas"
Mfg         ;4   e   "Manufacturing"
Commerce    ;5   e   "Commerce"
Transport   ;6   e   "Transportation"
Services    ;7   e   "Services"
Government  ;8   e   "Government"
```

Note that some of the lines in the command file – for example, the second – have a % followed by a number. These numbers refer to "arguments" from the "argument" file. For example, on the first line of the argument file, argument 1 is *Agricul*, argument 2 is *;*, argument 3 is *1*, argument 4 is *e* , and argument 5 is *Agriculture* . Normally an argument is ended by a space or punctuation. Enclose arguments which contain spaces – such as the names of some sectors – in quotation marks. When the second line of the command file,

```
ti  %3  %5
```

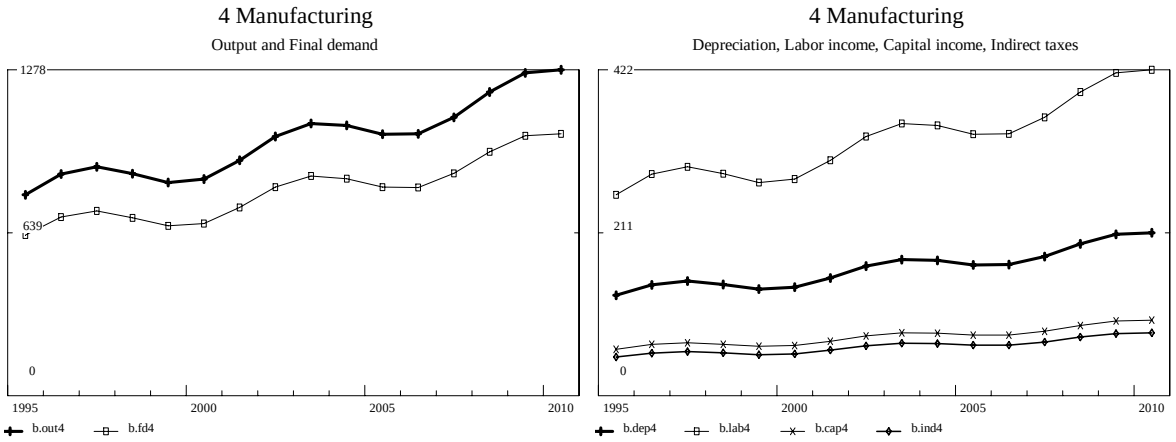
is executed with the arguments 3 and 5 from the first line of the argument file replacing the %3 and %5, the effect is that G executes the command

```
ti 1 Agriculture
```

The effect of the *fadd* command is that the entire command file is executed first with arguments from the first line of the argument file, then with the arguments from the second line of the argument file, and so on. Thus, with the single command

```
fadd  graphs.fad  sectors.ttl
```

G will draw for all sectors graphs like the two shown below for Manufacturing.



We have used some but not all of the G commands for matrix arithmetic in a VAM file. For reference, here are some others.

minv A	converts A into its inverse
madd A = B + C	adds B and C and stores in A
madd A = B - C	subtracts C from B and stores result in A
mmul A = B*C	multiply B and C and store result in A
mmul A = B'C	multiplies B transpose by C and stores result in A
mmul A = B&C	does element-by-element multiplication of B and C and stores in A
mmul A = B/C	element-by-element division of B by C stored in A
mtrans A B	the transpose of B is stored in A

In all of them, the command may be followed by an optional year in which to do the calculation; absent the year, the calculation is done for all years in the fdates range.

For tabulating the contents of a VAM file, we use exactly the same program, Compare, as we have used for macro models. It has, however, some capacities we have not used previously but now need. First of all, when we click Model | Tables on the G main menu, we need to choose "vam" as the type of the first bank, then give "hist" as its name; in the "Stub file" control, fill in "tiny", and in the "Output file name" box type "tiny.out".

The TINY.STB File

```
\dates 1995 2000 2005 2010 1995-2000 2000-2005 2005-2010
\pages off
\noformat
\title TINY G-ONLY MODEL, ILLUSTRATIVE FORECAST

; out  Output of Industries
&
out1 ;1 Agriculture
out2 ;2 Mining and quarrying
out3 ;3 Electricity and gas
out4 ;4 Manufacturing
out5 ;5 Commerce
out6 ;6 Transportation
out7 ;7 Services
out8 ;8 Government
;
\add tiny.tab pce "Personal Consumption Expenditure"
;
\add tiny.tab gov "Government Expenditures"
;
\add tiny.tab inv "Investment by Supplying Industry"
;

# The next line forces a new page
*
\matcfg Matlist.cfg
\center Matrix Listing
\row
\cutoff .001
\matlist 1-8
```

The first lines with the "\dates" command is familiar from macro models. Since I want to bring the results into a word processor for printing, I have turned off the page numbering and all commands to the printer in the next two lines. The "\title" command gives a title to be printed across the top of each page of output. As with macro stub files, a line beginning with a ";" just puts the rest of the line in the output file, and a "&" command puts a line of dates across the page. The next eight lines then show the output and its growth rates for the eight industries of the Tiny model for the dates specified.

We have not previously used Compare's *\add* command, which works just like G's *add* command, including a feature of the *add* command which we have not much used, namely, that it accepts arguments. The TINY.TAB file is shown in the box below. Instead of the lines in TINY.STB for printing the output of industries, we could have used the single line

```
\add  tiny.tab  out  "Output of Industries"
```

The effect would have been exactly the same.

The TINY.TAB File

```
; %1 %2
&
%11 ;1 Agriculture
%12 ;2 Mining and quarrying
%13 ;3 Electricity and gas
%14 ;4 Manufacturing
%15 ;5 Commerce
%16 ;6 Transportation
%17 ;7 Services
%18 ;8 Government
```

The TINY.TAB is a bit confusing to the eye because of the strings “%11" , “%12", and similar strings below them. To the eye, this may look like a reference to argument 11 or argument 12. But the computer knows that there can be only nine arguments and thus the third character in these strings is not part of the argument specification. It will read these as “argument 1 followed by the character 1" or “argument 1 followed by the character 2.”

The results the tabulations described thus far are shown in the first box below.

The last five lines of TINY.STB are concerned with making a *matrix listing* from the VAM file. What is meant is best explained by looking at the results, which are shown for the first three industries in the second box below. For each row of the input-output table, the matrix listing shows each element of the identity:

output = intermediate demand + final demand.

Indeed, each element is shown in each year specified by the *\dates* command and growth rates of the element are shown for the periods specified by the same command. This matrix listing technique is important not only for the information it displays but also the consistency of the forecasts which it emphasizes.

TINY G-ONLY MODEL, ILLUSTRATIVE FORECAST

out Output of Industries

	1995	2000	2005	2010	95-00	00-05	05-10
1 Agriculture	164.0	181.0	216.0	263.7	2.0	3.5	4.0
2 Mining and quarrying	50.0	56.0	66.3	79.8	2.3	3.4	3.7
3 Electricity and gas	205.0	233.5	274.1	324.7	2.6	3.2	3.4
4 Manufacturing	787.0	849.4	1025.5	1278.3	1.5	3.8	4.4
5 Commerce	401.0	463.0	539.7	630.8	2.9	3.1	3.1
6 Transportation	198.0	225.9	264.9	313.3	2.6	3.2	3.4
7 Services	667.0	767.2	896.0	1051.3	2.8	3.1	3.2
8 Government	150.0	174.3	202.5	235.2	3.0	3.0	3.0

pce Personal Consumption Expenditure

	1995	2000	2005	2010	95-00	00-05	05-10
1 Agriculture	15.0	17.4	20.2	23.5	3.0	3.0	3.0
2 Mining and quarrying	2.0	2.3	2.7	3.1	3.0	3.0	3.0
3 Electricity and gas	80.0	92.9	108.0	125.5	3.0	3.0	3.0
4 Manufacturing	400.0	464.7	539.9	627.3	3.0	3.0	3.0
5 Commerce	350.0	406.6	472.5	548.9	3.0	3.0	3.0
6 Transportation	130.0	151.0	175.5	203.9	3.0	3.0	3.0
7 Services	500.0	580.9	674.9	784.2	3.0	3.0	3.0
8 Government	0.0	0.0	0.0	0.0	0.0	0.0	0.0

gov Government Expenditures

	1995	2000	2005	2010	95-00	00-05	05-10
1 Agriculture	1.0	1.2	1.3	1.6	3.0	3.0	3.0
2 Mining and quarrying	1.0	1.2	1.3	1.6	3.0	3.0	3.0
3 Electricity and gas	10.0	11.6	13.5	15.7	3.0	3.0	3.0
4 Manufacturing	80.0	92.9	108.0	125.5	3.0	3.0	3.0
5 Commerce	10.0	11.6	13.5	15.7	3.0	3.0	3.0
6 Transportation	20.0	23.2	27.0	31.4	3.0	3.0	3.0
7 Services	40.0	46.5	54.0	62.7	3.0	3.0	3.0
8 Government	150.0	174.3	202.5	235.2	3.0	3.0	3.0

inv Investment by supplying industry

	1995	2000	2005	2010	95-00	00-05	05-10
1 Agriculture	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2 Mining and quarrying	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3 Electricity and gas	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4 Manufacturing	200.0	174.8	237.3	352.7	-2.7	6.1	7.9
5 Commerce	6.0	5.2	7.1	10.6	-2.7	6.1	7.9
6 Transportation	8.0	7.0	9.5	14.1	-2.7	6.1	7.9
7 Services	10.0	8.7	11.9	17.6	-2.7	6.1	7.9
8 Government	0.0	0.0	0.0	0.0	0.0	0.0	0.0

TINY G-ONLY MODEL, ILLUSTRATIVE FORECAST

Matrix Listing

Seller: 1 Agriculture

	1995	2000	2005	2010	95-00	00-05	05-10
Sales to Intermediate							
1 Agriculture	20.0	22.1	26.3	32.2	2.0	3.5	4.0
2 Mining and quarrying	1.0	1.1	1.3	1.6	2.3	3.4	3.7
4 Manufacturing	100.0	107.9	130.3	162.4	1.5	3.8	4.4
5 Commerce	5.0	5.8	6.7	7.9	2.9	3.1	3.1
7 Services	2.0	2.3	2.7	3.2	2.8	3.1	3.2
SUM: Intermediate	128.0	139.2	167.4	207.2	1.7	3.7	4.3
Sales to Final Demand							
Personal consumption expenditure	15.0	17.4	20.2	23.5	3.0	3.0	3.0
Government consumption	1.0	1.2	1.3	1.6	3.0	3.0	3.0
Exports	40.0	46.5	54.0	62.7	3.0	3.0	3.0
Imports	-20.0	-23.2	-27.0	-31.4	3.0	3.0	3.0
Output	164.0	181.0	216.0	263.7	2.0	3.5	4.0

Seller: 2 Mining and quarrying

	1995	2000	2005	2010	95-00	00-05	05-10
Sales to Intermediate							
1 Agriculture	4.0	4.4	5.3	6.4	2.0	3.5	4.0
2 Mining and quarrying	3.0	3.4	4.0	4.8	2.3	3.4	3.7
3 Electricity and gas	20.0	22.8	26.7	31.7	2.6	3.2	3.4
4 Manufacturing	15.0	16.2	19.5	24.4	1.5	3.8	4.4
5 Commerce	2.0	2.3	2.7	3.1	2.9	3.1	3.1
6 Transportation	1.0	1.1	1.3	1.6	2.6	3.2	3.4
7 Services	2.0	2.3	2.7	3.2	2.8	3.1	3.2
SUM: Intermediate	47.0	52.5	62.2	75.1	2.2	3.4	3.8
Sales to Final Demand							
Personal consumption expenditure	2.0	2.3	2.7	3.1	3.0	3.0	3.0
Government consumption	1.0	1.2	1.3	1.6	3.0	3.0	3.0
Exports	10.0	11.6	13.5	15.7	3.0	3.0	3.0
Imports	-10.0	-11.6	-13.5	-15.7	3.0	3.0	3.0
Output	50.0	56.0	66.3	79.8	2.3	3.4	3.7

Seller: 3 Electricity and gas

	1995	2000	2005	2010	95-00	00-05	05-10
Sales to Intermediate							
1 Agriculture	6.0	6.6	7.9	9.6	2.0	3.5	4.0
2 Mining and quarrying	4.0	4.5	5.3	6.4	2.3	3.4	3.7
3 Electricity and gas	10.0	11.4	13.4	15.8	2.6	3.2	3.4
4 Manufacturing	40.0	43.2	52.1	65.0	1.5	3.8	4.4
5 Commerce	20.0	23.1	26.9	31.5	2.9	3.1	3.1
6 Transportation	10.0	11.4	13.4	15.8	2.6	3.2	3.4
7 Services	25.0	28.8	33.6	39.4	2.8	3.1	3.2
SUM: Intermediate	115.0	128.9	152.6	183.5	2.3	3.4	3.7
Sales to Final Demand							
Personal consumption expenditure	80.0	92.9	108.0	125.5	3.0	3.0	3.0
Government consumption	10.0	11.6	13.5	15.7	3.0	3.0	3.0
Output	205.0	233.5	274.1	324.7	2.6	3.2	3.4

Perhaps you are wondering how the Compare program knows what elements go into the identity and what are the names of the sectors and final demands. The answer was given to the program in the "matrix listing configuration file" whose name, MATLIST.CFG, was given to Compare in the command

```
\matcfg  matlist.cfg
```

The matrix listing configuration file to produce the matrix listing shown above is in the box below.

The MATLIST.CFG File for TINY

```
Matrix listing identity;out=AM*out+pce+gov+inv+ex+im
# Title file name for the rows of out, the lefthand side vector
out; "sectors.ttl"
# Title file names for matrix columns
AM; "sectors.ttl"
# headers for each term
header for out;    "Output"
header for AM*out; "Intermediate"
header for pce;    "Personal consumption expenditure"
header for gov;    "Government consumption"
header for inv;    "Investment"
header for ex;     "Exports"
header for im;     "Imports"
```

In the MATLIST.CFG file, any line beginning with a # is a comment and anything before the " ; " is likewise a comment. The first line gives the crucial identity on which the matrix listing is built. Recall that Compare has the VAM file and thus knows all the matrix and vector names and dimensions. It knows how to interpret correctly the expression "AM*out." The next line gives the file name of the sector titles for the vector on the left. Then follow the file names for column titles of any matricies for any matrices appearing in the identity. Here we have only one such matrix. Then come headers for each section of the table, a section being a vector or a matrix-vector product.

We return now to the TINY.CFG file to explain the last four lines, namely

```
\center Matrix Listing
\row
\cutoff .001
\matlist 1-8
```

The *center* command centers the following text on the page. The command *row* tells Compare to interpret the identity of the matlist configuration file as an identity in the rows. (The other possibility, *column*, would be used for showing the identity – that holds only in current prices – between the value of the output of an industry and the sum of its intermediate inputs and value-added components.) The *cutoff* command eliminates the printing of entries which account for less than the specified fraction of the total of the row or column being listed.

Finally, the *matlist* command instructs Compare to make the matrix listing for the *group* of sectors following the command. This is our first encounter with the *group* concept which is quite useful in working with multisectoral models. A group is just a collection of integers; it can be specified in a rather flexible way. Our specification, 1 - 8, means every sector from 1 to 8. An equivalent specification would be 1 2 3 4 5 6 7 8. If we want just 1 to 3 and 6 to 8, we could write any of following:

1 - 3 6 - 8
1 2 3 6 7 8
1-8 (4 5)

The numbers in the parenthesis are stricken from the list created by the ranges to the left; the parenthesis can also include ranges.

You may now want to ask, "Shouldn't we connect personal consumption expenditure to labor and capital income?" Of course we should, but to do so goes beyond what we can do in G alone. It requires the Interdyme modeling system, which is similar to Build but for multisectoral models. Everything we have covered in this section is directly relevant to working with Interdyme, but surely you need to pause here and be sure that you have mastered the large amount of information we have already covered. What better exercise could there be than to build your own Tiny model, so do exercise 1 for sure and the others to explore some other ideas.

Exercises

1. Make up the input-output table for your own imaginary economy in 2005. It should have five to ten sectors, but not eight. Use different sector titles. Make forecasts to 2025. Graph the forecasts and make tables of output and other vectors. Make a matrix listing.

2. For the economy of our example, what levels of output and use of primary imputs would be required for the final demand (40, 6, 100, 600, 400, 170, 700, 148)?

3. How much of each of the four factors does one dollar of each of the final demands contain?

4. Was this economy a next exporter or importer of depreciation?

5. What would happen to the prices of each of the eight products if all indirect taxes were eliminated?

6. Greenhouse gases are emitted by the production of the various sectors of our model economy. Measured in tons per billion dollars of output, the emission coefficients for the various sectors of our economy are

   2.1 1.3 6.1 1.8 1.0 4.3 0.8 0.0

   What is the emission of greenhouse gases per billion dollars of final demand for each of the eight products? How much is attributable to a billion dollars of each of the types of final demand -- consumption, government, etc.? Was this country a next exporter or importer of greenhouse gas emissions?

7. The input-output flow table illustrated in the text was for year A. A comparable table for the same country but for a later year, year B, may be found in the files YBF.DAT, YBX.DAT and YBV.DAT in the TINY.ZIP file. (You have to fix up the correct commands to get the data into G.) Price indexes for the eight sectors from year A to B are given by the vector

   (1.01 1.10 1.06 1.07 1.15 1.24 1.18 1.20),

   while the cost of labor increased twenty percent between the two years. (The price indexes are in the file PINDEX.DAT.) What has happened between the two years to total labor requirements for producing one unit of final demand for each product?

8. Return to exercise 7 but now consider that the depreciation and capital income are produced with material inputs in the proportions given by the investment vector of the year in question. Ignore the indirect taxes and imports. The reciprocals of the labor requirements are productivity indexes for the economy in producing the various products supplied to final demand.

As we noted in section 1, it is impossible to know what has happened to productivity in a single industry, because the industry may have reduced its primary inputs while increasing its intermediate inputs; and the

double-deflation method, supposed to handle this problem, is totally fallacious. The same problem does not arise in looking at total labor required, indirectly as well as directly, for the production of each unit delivered to final demand, for if the direct supplier to final demand has shifted required labor to other industries by buying more intermediate goods, that indirect labor will be automatically picked up. Thus, input-output calculations may offer a way of studying trends in productivity by product which elude methods which do not take into account indirect effects.

4. Iterative Solutions of Input-output Equations

Before moving on to the Interdyme software, we must explain one of the mathematical techniques it uses extensively, namely the Seidel iterative solution of the input-output equations. In actual input-output computations, the Leontief inverse is seldom used, for the equations $q = Aq + f$ or $p = pA + v$ can be solved directly from the A matrix in about the same time required to multiply $(I - A)^{-1}$ by f or v. Thus, the effort of calculating $(I - A)^{-1}$ would be pointless. Moreover, for large matrices, many cells of A are zero. This fact can be exploited to reduce the computer storage required for the matrix. But the Leontief inverse will have non-zeroes nearly everywhere, so there is no way to reduce the space required for it. Further, changes to A are easily recorded and applied, but a change of one element in A can easily change all the elements in the inverse. Thus, from the point of view of solving the equations, nothing is gained and a good deal lost by computing the inverse.

How to solve the equations without the use of the inverse is the subject of this section. We will explain two methods of successive approximation, for it is worth knowing that both work even though we mainly use the second. The first, the simple iterative method, takes as a first approximation of q, $q^0 = f$. Then, given the n^{th} approximation, q^n, the next approximation is

$$q^{n+1} = Aq^n + f. \qquad (14.4.1)$$

If the process converges so that one q is indistinguishable from the previous one, then the vector to which it has converged is clearly the solution of the equation. In economic terms, we first set the output equal to the final demands. Then we increase it to allow for the intermediate goods needed by the first approximation and then incease it again for the intermediate goods needed for the second approximation, and so on.

It is clear from equation (1.4.1) that if the matrix A is non-negative and f is non-negative, then no element of q ever becomes negative in the course of the iterations. Thus, the conditions on A that insure the convergence also insure that a non-negative f leads to a non-negative q. Thus, our inquiry, initially motivated by considerations of practical computation, also provides an answer to the theoretical question of whether an economy could exist with a given f and A, for the economic interpretation of Aq is dependent on all elements of q being non-negative.

The second method, the Seidel process, takes the same first approximation, and then, to get the second approximation, solves first the first equation for q_1, given all the other elements of q. Then, using this new value of q_1 and the old values of q_3, q_4, *etc.*, solve the second equation for q_2, and so on. If the A matrix is triangular, that is, if all the entries above the main diagonal are zero, this method gives the right answer with one iteration. If it is not triangular, the whole process is repeated until little or no change occurs with each new iteration. While no actual input-output matrix is ever exactly triangular, the sectors can often be taken in an order which makes the matrix almost triangular, and this almost-triangularity speeds the convergence process.

Instead of starting this process with the final demands, it is also possible to start with any guess of *q*. In dynamic models, a good guess, namely the previous year's *q* is available. With a good starting point, four or five iterations of the Seidel process is usually sufficient to produce adequately accurate solutions. If twenty percent of the elements of *A* are non-zero -- a fairly typical situation -- we can make five iterations of the Seidel process in the same time which would be required to multiply *f* by the inverse if we had it.

If *A* is not an input-output matrix but just any old matrix you happen to meet on the street, there is not much chance that either of these methods will converge and give a solution. What then makes us so sure that they will converge for an input-output matrix? To discuss *convergence*, we need to be able to say how far apart two vectors are. The concept of the *norm of a vector* gives us that ability. We even need to be able to say how far a given vector is from the solution when we do not know what the solution is. The concept of the *norm of a matrix* enables us to turn that trick. We will now explain these two concepts.

We can say how far apart two vectors are if we can say how "long" a vector *x* is, that is, how long the line is which connects *x* with the origin or zero point. For if $\|x\|$ represents the length of any vector, then the length of the difference of two vectors *a* and *b*, $\|a-b\|$, serves as a measure of how far apart they are. How shall we weasure the length of a vector? In two dimensions, the usual lenght of the vector (x_1, x_2) is $\sqrt{x_1^2+x_2^2}$. This concept of length readily generalizes to vectors of any dimension by the definition $\|x\| = \sqrt{x'x}$. This formula, called the Euclidean length (or norm), gives one possible way of measuring length.

Why, however, do we bother to take the square root in the Euclidean norm? Because we certainly want *any* way of calculating the length of *x* to be such that multiplying each element of *x* by a scalar, λ, multiplies the length of *x* by the absolute value of λ:

(a) $\|\lambda x\| = |\lambda| \cdot \|x\|$.

Other properties which any definition of length should have are

(b) $\|0\| = 0$ and $\|x\| > 0$ if $x \neq 0$

and

(c) $\|x+y\| \leq \|x\| + \|y\|$.

Property (c) expresses the requirement that the shortest distance between any two points must be a straight line. Let us denote the points by *x* and *-y*. Then we must have

$$\|x-(-y)\| \leq \|x\| + \|-y\|,$$

since $\|x\|$ is the distance from x to 0 (the origin of the vector space) and $\|-y\|$ is the distance for 0 to *-y*, while $\|x-(-y)\|$ is the distance directly from *x* to *-y*. By applying property (a) to the second term on the right, this requirement may be written more simply as (c) above.

Any way of assigning a number, $\|x\|$, to each vector, *x*, of the vector space in such a way that (a), (b), and (c) are satisfied is called a *norm* of the space, and $\|x\|$ is read "the norm of x.". It is quite remarkable that we can often prove the convergence of a process in terms of a norm without knowing exactly which norm we are using. Besides the Euclidean norm, there are two more important examples of norms:

the l-norm: $\|x\| = \sum_{i=1}^{n} |x_i|$

the m-norm: $\|x\| = \max_i |x_i|$

You may easily verify that each of these norms has the required three properties, though the values they give as the norm of a given vector may be quite different. For example, the vector (1, -3, 2) has a Euclidean norm of 3.74, while its l-norm is 6 and its m norm is 3. (The l in l-norm refers to Henri Lebesgue, a French mathematician of the early years of the twentieth century.)

Exercise 9: Draw the unit circle for each of these three norms. (The unit circle is the locus of points with norm 1.)

With each of these three norms, if x^k, for k = 0, 1, 2, etc., is a sequence of vectors and x^* is a vector such that

$$\lim_{k\to\infty} \|x^k - x^*\| = 0,$$

then

$$\lim_{k\to\infty} x^k = x^*.$$

That is, convergence of a sequence of vectors in norm implies element-by-element convergence. This property is easily seen for the examples of the three norms and is a characteristic of finite dimensional vector spaces.

What we now want to show is that if q^* is a solution of the input-output equations, so that

$$q^* = Aq^* + f, \tag{14.4.2}$$

then the sequence $q^0, q^1, q^2, \ldots$ defined by

$$q^{k+1} = Aq^k + f \tag{14.4.3}$$

converges in norm to q^*. Subtracting the first equation, (14.4.2), from the second, (14.4.3), gives

$$q^{k+1} - q^* = A(q^k - q^*),\ k = 1, 2, 3, \ldots . \tag{14.4.4}$$

If we have computed to iteration m, then setting $k = m$ in this equation gives

$$q^{m+1} - q^* = A(q^m - q^*).$$

But setting k = m+1 in (1.4.4) gives

$$q^{m+2} - q^* = A(q^{m+1} - q^*).$$

Together the last two equations imply

$$q^{m+2} - q^* = A(q^{m+1} - q^*) = A^2(q^m - q^*).$$

For any positive integer, p, similar reasoning applied p times gives

$$q^{m+p} - q^* = A^p(q^m - q^*). \tag{14.4.5}$$

We would like to be able to show that the norm of the vector on the left of (14.4.5) goes to zero as *p* goes to infinity. To do so, we need to extend the concept of *norm* to matrices. We introduce that extension by a question:

Is there a number, call it $\|A\|$, such that

$$\|Ax\| \le \|A\| \cdot \|x\| \tag{14.4.6}$$

for all x?

There are indeed such numbers, and we call the least of them (for any norm of the vectors) the *norm* of A. Intuitively speaking, the norm of the matrix A is the greatest "stretch" which multiplication by A performs on any vector. For the l-norm and m-norms of the vectors, the corresponding norms of a matrix are easily computed, as we shall see in a moment. Note that the norms of matrices also have the three basic properties of the norms of vectors:

a) $\|A\| = 0$ if and only if $A = 0$.
b) $\|\lambda A\| = |\lambda| \|A\|$
c) $\|A + B\| \le \|A\| + \|B\|$

plus a fourth, which can be easily verified from the definition

d) $\|AB\| \le \|A\| \|B\|$.

First, however, note that we can apply this inequality repeatedly to equation (14.4.5). After applying it p times, we have

$$\|q^{m+p} - q^*\| = \|A\|^p \cdot \|q^m - q^*\|$$

If we can show that $\|A\| \le 1$ for *some* norm, then $\|A\|^p \to 0$ as $p \to \infty$,
and therefore $q^k \to q^*$ as $k \to \infty$, and the iterative calculations converge to the solution.

The norm of the n-by-n matrix A induced by the m-norm of vectors, and therefore called the m-norm of the matrix, is

$$\|A\|_m = \max_i \sum_{j=1}^{\cdot} |a_{ij}|.$$

while the norm of A induced by the l-norm of vectors, and therefore called the l-norm of the matrix, is

$$\|A\|_l = \max_j \sum_{i=1}^{n} |a_{ij}|.$$

We shall prove the formula for the l-norm, and leave that for the m-norm as an exercise. (The Euclidean norm of A is more complicated and not of immediate concern to us. It is the largest characteristic root of A′A.) For the l-norm, let

$$\alpha = \max_j \sum_{i=1}^{n} |a_{ij}|.$$

Then $\|A\|_l \le \alpha$, because

$$\|Ax\|_l = \sum_{i=1}^{n} |\sum_{j=1}^{n} a_{ij}x_j| \le \sum_i \sum_j |a_{ij}|\cdot|x_j|$$
$$= \sum_j |x_j| \sum_i |a_{ij}| \le \sum_j |x_j|\,\alpha = \alpha\|x\|_l.$$

On the other hand, let k be the number of the column with the largest sum of absolute values, so that

$$\alpha = \sum_{i=1}^{n} |a_{ik}|$$

and then choose a vertor, x, with $x_k = 1$ and $x_j = 0$ for $j \neq k$. Then $\|x\| = 1$, and

$$\|Ax\|_l = \sum_i |\sum_i a_{ij}x_j| = \sum_i |a_{ik}| = \alpha = \alpha\|x\|.$$

Therefore, $\|A\|_l \ge \alpha$. But we have already shown the opposite inequality, so the only possibility is that $\|A\| = \alpha$.

If an input-output A matrix comes from an observed economy with a positive value-added in every industry, then the column sums of every column are less than 1.0 and therefore the l-norm of the matrix is less than 1. Thus, returning to the iterative solution of the input-output equations, we see that it will indeed converge if such is the source of A. Furthermore, in that case, $(I - A)^{-1}$ will be non-negative, because if we start from an *f* vector which is all zero except for a 1 in some position, the resulting solution will never have any opportunity to acquire any negative elements in the course of the iterative process. But the columns of $(I - A)^{-1}$ are precisely the solutions of such equations, so the whole matrix is non-negative.

The norm of the *A* matrix not only allows us to be sure that the iterative process converges, it also allows us to set an upper bound on how far we are from the solution at any stage. If, as before, q^k indicates approximation k, then

$$q^{k+p} - q^k = q^{k+1} - q^k + q^{k+2} - q^{k+1} + \ldots + q^{k+p} - q^{k+p-1} \qquad (14.4.7)$$

But since

$$q^{m+1} = Aq^m + f \quad \text{and} \quad q^m = Aq^{m-1} + f$$

for any positive integer m, subtraction gives

$$q^{m+1} - q^m = A(q^m - q^{m-1}).$$

Repeatedly applying this equation gives

$$q^{k+1} - q^k = A(q^k - q^{k-1})$$
$$q^{k+2} - q^k = A^2(q^k - q^{k-1})$$
$$\cdots$$
$$q^{k+p} - q^{k+p-1} = A^p(q^k - q^{k-1})$$

and substitution in the above equation (1.4.7) gives

$$q^{k+p} - q^k = (A + A^2 + A^3 + \ldots + A^p)(q^k - q^{k-1})$$

Taking the norms of both sides and applying properties c and d of the norms of matrices gives

$$\|q^{k+p} - q^k\| \le \|(A + A^2 + A^3 + \ldots + A^p)\|\|q^k - q^{k-1}\|$$
$$\le(\|A\| + \|A\|^2 + \|A\|^3 + \ldots + \|A\|^p)\|q^k - q^{k-1}\|.$$

Now as $p \to \infty$, $q^{k+p} \to q^*$ and the sum of the geometric progression on the right goes to $\|A\|/(1 - \|A\|)$ because $\|A\| < 1$. Thus, when we have reached iteration k, we know that the distance to the true solution is less than $\|q^k - q^{k-1}\|\|A\|/(1 - \|A\|)$. In other words, when the differences of the successive approximations get small, we can be sure that we are close to the true solution.

Now suppose for a moment that A is a matrix in *physical* units -- with coefficients in units like kilowatt hours per pound -- so that column sums are meaningless and the l-norm perhaps much greater than 1. Further let *w* be an all-positive vector of the hours of labor -- the only primary input -- required per physical unit of output in each industry. Can an economy exist with this technology? In other words, if the vector f of final demands is all positive, will the vector of outputs, *q*, such that $q = Aq + f$ also be all positive? (Mathematically, it is quite possible for some element of q to be negative, but it is economic nonsense to run an industry at a negative level. Coal can be converted into electricity, but all the electricity in the world can't make a ton of coal.)

The answer to these questions lies in the solution of $p = pA + w$ (where *p* is a row vector). If *p* is all positive, then it can be thought of as a vector of prices (with an hour of work as the numeraire) at which each process has a positive value added. If we now change the units of measurement of output of each product to one "hour's worth," the coefficient matrix, say A^*, in these new units corresponding to A in the old units will have columns whose sums are each less than 1. Thus, in these units, the iterative procedure will converge. But the iterative procedure in the original units (with A) would give successive approximations which differ from those with A^* only in their units. Hence the process would converge in the original units as well and $(I - A)^{-1}$ will be non-negative. Since the Leontief inverse is non-negative, any vector of non-negative final demands can be met by non-negative levels of output of all the industries.

5 The Seidel Method and Triangulation.

As mentioned at the outset of the previous section, there is a variation of the iterative method, known as the Seidel method, which converges even faster. In it, one starts with f as the initial guess of the solution just as in the simple iterative method, but then solves the first equation for the first variable and puts this value into the guess, then solves the second equation for the second variable and puts that value into the guess, and so on. Formally,

$$q_i^{(k+1)} = \left(\sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} + \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} + f_i\right)/(1 - a_{ii}).$$

In input-output work, the *f* vector is generally non-negative as are the elements of the *A* matrix. Hence, in the simple iterative method, the approximate solutions form a monotonicly increasing sequence of vectors. The Seidel approximate solutions are also monotonicly increasing but are always larger than the

corresponding simple iterative solution. Hence, it also converges to the solution and does so faster than does the simple iterative method.

If all the non-zero elements of A are on or below the main diagonal, A is said to be *triangular*. If A is triangular, one pass of the Seidel process is sufficient to reach the exact solution. If A is merely almost triangular, a few iterations will suffice for a good solution. It general, input-output matrices arrive from the statistical offices more or less triangulated in exactly the wrong way. They start with Agriculture first, later Textiles, then Apparel. The right order for a fast Seidel solution is the reverse, Apparel, Textiles, Agriculture. It is not, however, necessary to physically re-arrange the rows and columns. All that is necessary is to take the rows in the Seidel operation in the order that would make the matrix nearly triangular.

For large matrices, however, it may be convenient to have a mechanical way to generate an approximately triangular order. A simple but effective is to pick as the first industry the one which has the smallest ratio of intermediate to final demand in its row. Then move into final demand all the inputs into this industry and again pick from the remaining sectors the one with the lowest ratio of intermediate to final in its row. Continue until all industries have been selected.

Solving input-output equations by the Seidel method is not only generally much faster than inverting the $I - A$ matrix by Gauss-Jordan reduction, it may even be faster than multiplying $(I - A)^{-1}$ by f when $(I - A)^{-1}$ is already known. How can that be? It is common for the A matrix to be quite sparse. A 300-by-300 matrix may have some 9,000 non-zero elements, not 90,000. It can be stored in a "packed" form in which only non-zero elements are stored, and the Seidel algorithm can be written to use this packed form, so that only as many multiplications and additions are required per iteration as there are non-zero elements. Thus, if the Seidel process requires less than ten iterations in our example, it will require *less than* 90,000 multiplications and additions. The Leontief inverse, however, will generally have 90,000 non-zeroes and thus multiplying it by f involves exactly 90,000 multiplications and additions. To economize on both space and solution time, large, sparse matrices are thus best stored in a packed form; and equations involving them should be solved by the Seidel process without ever inverting the matrix.

Exercises

10. Using C, Fortran, Basic or any programming language you know, write a program to compute the triangular order of a matrix. Apply it to the flow matrix used as an example in this chapter. Write the results as a vector of integers, the first being the number of the equation to be taken first; the second, that of the equation to be taken second, etc.

11. Write a program to use the Seidel method to solve input-output equations, taking the equations in the order specified by the vector produced in exercise 7. Apply the program to solve exercise 1 earlier in this chapter. (Bump has a Seidel method. Try to create yours without looking at it.)

6. Introduction to Interdyme

In section 4, we became acquainted with the VAM file and saw that G could do a number of calculations with the matrices and vectors in these files. By the end, however, we came up against the limit of G by itself; to integrate regression equation and input-output we have to go on to Interdyme. Interdyme is a collection of C++ programs which make it easy to construct interindustry dynamic models involving regression equations, input-output computations with matrix algebra, and lag relationships that provide the dynamics. One of the modules, contained in the MODEL.CPP, is written by the user and the

whole is then compiled and linked just as are macro models when you click Model | Build to build a macro model. The result is a very fast-running model. This speed of calculation becomes especially important when we undertake optimization or stochastic simulation with the model.

I have more than once encountered the reaction, "C++ is hard; I don't have time to learn it, so I'll just stay with the models I can build in Excel." Probably the best answer to this objection is to look at heart of the TINY model written in C++ for Interdyme. It is in the box below. Now it is true that you need to know a little about C++ to read it. Here is what you need to know:

1. Anything following a // on the same line is a comment.
2. Every C or C++ statement ends with a semicolon.
3. For any variable *x*, *x*++ means "add 1 to x."
4. C and C++ group statements with braces {like this}.
5. The *for* statement

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

means start with *t = godate* (a year, like 1995) and execute all of the statements enclosed in the braces, then increase *t* by 1 and repeat, and continue this process as long as *t* is less than or equal *stopdate* (another year, like 2010).

This program also uses the Interdyme functions *load(t)* to load up from the VAM file all vectors and matrices with their data for year *t*, while *store(t)* stores their newly computed values back to the VAM file. The Interdyme function *Seidel(AM, out, fd, triang, toler)* solves by the Seidel method the equation

*out = AM*out + fd*

Heart of the TINY Model

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

       // Load all vectors and matrices.
       load(t);

       // Add up the final demands
       fd = pce + gov + inv + ex + im;

       // Solve the input-output equations for output by the Seidel method
       Seidel(AM, out, fd, triang, toler);

       // Compute the value-added vectors by element-by-element
       // multiplication of the coefficient vectors with the output vector.
       dep = ebemul(depc,out);
       lab = ebemul(labc,out);
       cap = ebemul(capc,out);
       ind = ebemul(indc,out);

       // Store the values of vectors and matrices for this period.
            store(t);
       }
```

using the order of equations specified by the vector *triang* and with the convergence tolerance specified by *toler*. Finally, it is Interdyme, not raw C++, which knows how to add vectors like *pce, gov, inv* and so on. In fact, though we have used here only vector addition, Interdyme has a rather complete library of matrix and vector routines.

With this much information and the comments in the program, it should be clear that this C++ program in the Interdyme system accomplishes exactly what the G-only Tiny model accomplished, namely, to add up the vectors of final demand, compute the outputs, and calculate value-added vectors. The results are written into the VAM file, from whence they can be graphed or displayed with Compare just as in the case of the G-only model. The difference is that, whereas in the G-only model we had at this point more or less reached the limits of what could be done, in the Interdyme framework we stand at the beginning of wide possibilities. Interdyme makes some common calculations, such as matrix algebra and use of equations estimated by regression, very easy; but if you need to do something for which there is no existing Interdyme routine, you have the full power of C++ to make the computer do just about anything of which it is capable.

I do not mean to discourage you from learning C and its extension, C++; indeed, if you are serious about modeling, you should do so. My point, however, is that you can use the Interdyme system to program your model with a very minimal knowledge of these languages. You will need to learn a little more as we go along.

Before we go further into Interdyme's capacities, however, we have to deal with a basic question: How is the Interdyme program connected with the VAM file? How does it know, for example, that there are vectors in TINY's VAM file named *fd, pce, gov, ind* and so on? The answer is that user has to tell it the names of these vectors and matrices; it, however, will figure out from the VAM file their dimension. This identification has to be done in two steps. First, there is a file called USER.H; the one for TINY is shown in the box below.

USER.H for the TINY Model

```
// USER.H -- Put here any includes that refer to the user model, per se.

// From DYME.CFG and opening screen:
   GLOBAL char RunTitle[80],CfgFileName[80],VamFileName[80],
            GbankName[80],VecFixFileName[80],MacFixFileName[80];

// Vector declarations:
   GLOBAL Vector out,pce,gov,inv,ex,im,fd,dep,labinc,capinc,indtax,x,y;

// Matrix declaration
    GLOBAL Matrix AM;

// triang is a vector of integers giving the sector numbers in
// approximately triangular order
   GLOBAL IVector triang;
```

All the variables declared in USER.H are “global,” C - jargon for variables which can be accessed from anywhere in the program. This fact is indicated by the word GLOBAL in front of these variables. To explain exactly what is going on here requires a digression which you can skip if you are familiar with C or don’t want to bother with the why and wherefore, so I will put it in slightly smaller type.

Unlike Basic and Fortran, C and C++ are strictly “typed” languages. That is, the nature of every variable must be declared before it can be used. Typical declarations are:

```
char sex;
int zip;
float height, weight;
char name[40];
```

The variable *sex* is a single character, presumably M or F; the variable *zip* is an integer, something like 20742; *height* and *weight* are floating point numbers, that is, numbers that potentially have a fractional component like 72.5 or 217.8. The variable *name* is a string of up to 40 characters, something like “Thomas”. If a variable is declared inside a function, it is *local* to that function. Other functions know nothing about it. But a variable declared outside of all functions, typically near the top of a file containing C code, is *global* and can be accessed by all functions in that file. A large program such as Interdyme usually consists of a number of files with names ending in .cpp; each of these files is called a *module*, and is compiled separately. If some variable, say *name*, is to be accessed in several different modules, then it must be declared globally in all the modules where it is accessed. But one and only one of the modules should actually make space for it. Which one? To answer this question, the program should mark all the declarations which are **not** to make space as *extern*, like this:

```
extern char name[40];
```

In one and only one module should appear the simple declaration

```
char name[40];
```

That is the module where the space is made. When the compiled modules are “linked” to form one whole executable program, the references to the variable in all the modules where it was external are made to point to the space allocated by the one module where it was not external.

It could potentially be quite a nuisance to remember to mark all but one declaration as extern. That is where the word “GLOBAL” comes in handy. It is not a standard C keyword but is used in Interdyme, G and many other programs. The declarations of all global variables are put into “header” files like USER.H, and they are all marked GLOBAL. These header files are then “included” into all the modules where they are relevant by a *compiler directive* like

```
#include “user.h”
```

In all but one module, this directive is preceded by another:

```
#define GLOBAL extern
```

In these modules, the compiler will replace “GLOBAL” by “extern” before compiling. In that one and only one other module, namely dyme.cpp in Interdyme, the “include” is preceded by

```
#define GLOBAL
```

which defines GLOBAL to be nothing, so that in this module the “extern” is omitted and space is made for the global variables.

When a vector or matrix is declared locally, it can be fully "constructed", that is, space allocated for its elements. But there is a problem with declaring an object like a vector globally; since the declaration is outside of an function, no computing can be done. But to find out how much space to allocate for the array of numbers in the vector, the VAM file has to be read, but reading the VAM file is computing, so what is to be done? The answer is that once computing has begun, we must *resize* all the global matrices and vectors. That is, we must read the VAM file, find out how big the matrix or vector is, grab enough memory to hold it, and stick the pointer to that memory into the space saved for the pointer by the global declaration.

That may seem complicated to understand, but it is easy to do. The box below has the code for the TINY model. The lines concerned with resizing are the following:

```
// Resize Vectors

out.r("out");pce.r("pce");gov.r("gov");
inv.r("inv");ex.r("ex");im.r("im");fd.r("fd");dep.r("dep");
depc.r("depc"),lab.r("lab");labc.r("labc");cap.r("cap");
capc.r("capc"),ind.r("ind");indc.r("indc");x.r("x");
y.r("y");

// Resize matrices
AM.r("AM");
```

The "resize" function or method is abbreviated to just .r . The argument to the resize function is the name of the vector or matrix in the VAM file. These lines are readily prepared from the VAM.CFG file with a text editor with good macro capabilities. The rest of the *loop()* function should be regarded as standard which the user of Interdyme should have no need to change. The *spin()* function which we have seen previously is repeated here with the addition of the Interdyme standard commands at the beginning to print the year and the standard conclusion of the *spin()* which allows slightly different behavior when optimizing .

If you look at model.cpp in the accompanying software, you will see that it has extensive comments and some conditional compilation which I have removed to show just what is actually used and done in the TINY model. In C or C++, a comment extending over more than one line is begun with a /* and ended with a */.

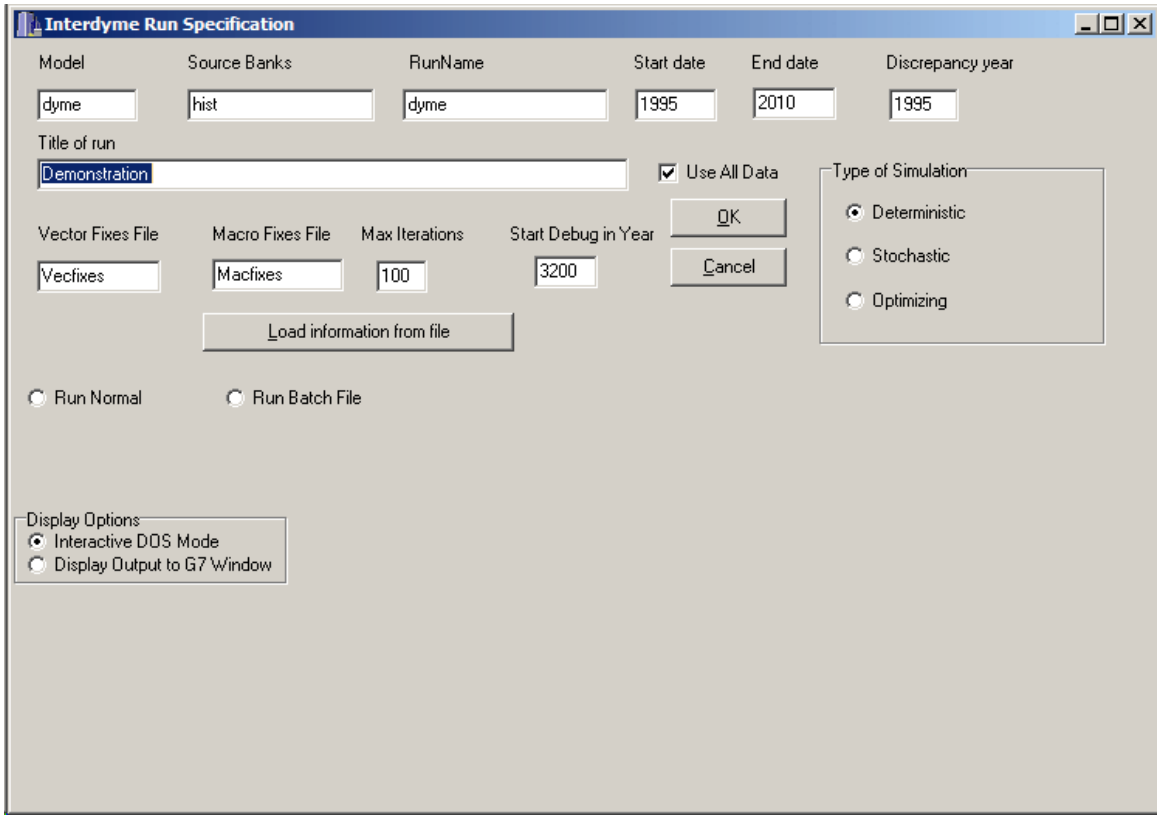
At this point, you should be able to build and run the TINY model. Start G in directory where you have put TIDY. In the editor, look at TINY.PRE. You will see that it is just the command file we developed in section 3 down to but not including taking the Leontief inverse. At that point, it ends with the command

```
close b
```

which closes the VAM file HIST.VAM and thus allows it to be used by other programs. Execute this command file by clicking Run on the editor menu. Then on the G main menu, click Model | IdBuild; if you have the free Borland C++ compiler 5.5 installed as was required for Part 1 of this book, you should now see the entire Interdyme system compiled and linked. Then click Model | Run Dyme, and the form show below appears.

The MODEL.CPP File for TINY

```
#include "user.h"     // All global variables for user model.

// Prototype for Seidel used in TINY
short Seidel(Matrix& A, Vector& q, Vector& f, IVector& triang,float toler);
void spin();
const int NSEC=8; // Number of sectors in I-O table, most vectors.

void loop(void){
    // Resize the vectors of TINY
    out.r("out");pce.r("pce");gov.r("gov");inv.r("inv");ex.r("ex");
    im.r("im");fd.r("fd");dep.r("dep");labinc.r("labinc");capinc.r("capinc");
    indtax.r("indtax");x.r("x");y.r("y");
    // Resize matrices
    AM.r("AM");
    // Read the triangularization vector
    triang.resize(NSEC);
    triang.ReadA("triang.iv");

    // The rest is standard in Interdyme
    // MaxFlag is 'y' if optimizing, otherwise 'n'.
    if(MaxFlag == 'n')
        spin();
    else{
        maxsolve();
        MaxFlag = 'n';
        spin();
        }
   printf("\nThe run has finished.  Use G or Compare to view results.\n");
   tap();
   }

// The spin() function sends the model through one cycle of years.

void spin(){

    for (t = godate; t<= stopdate; t++) {
       // General start of the spin() function:
       if (MaxFlag == 'n') printf("%d ",t);
       // Load all vectors and matrices.
       load(t);

        // Particular to TINY:
        fd = pce + gov + inv + ex + im;
        Seidel(AM, out, fd, triang, toler);
        dep = ebemul(depc,out);
        lab = ebemul(labc,out);
        cap = ebemul(capc,out);
        ind = ebemul(indc,out);

        // General end of the spin() function:
        if(MaxFlag == 'y')
            shiftback(t);
        else{
            // Store the values of vectors and matrices for this period.
            store(t);
            }
        }
    }
```



The top left edit control gives the name of the model; the default name is "dyme," which is what we will use. The name of the VAM file which we prepared was, you will recall, HIST.VAM, so "hist" appears as the source banks name. We can call the output VAM file anything we like, but the default is DYME.VAM, which is what we will use. If the names in these two fields are different, G will copy source bank (in our case, HIST.VAM) to the "Run name" file (in our case, DYME.VAM) before the calculations begin. If the two names are the same, no copy is made. The "start date" and "end date" fields are clear enough; the "discrepancy year" is not used in TINY. The "Title of run" field becomes the title of the VAM file which is displayed when making tables with the Compare program. To run TINY requires no change in any of this form's content, so just click the OK button, and the model should run. More precisely, G will make the copy of the VAM file if need, write the file DYME.CFG, and then execute the program DYME.EXE, which was created by the IdBuild command. A command line screen opens and the DYME program will begin by reading the DYME.CFG file. DYME will print to the screen the year it is calculating and the number of iterations in the Seidel solution.

As an exercise, you should write a command file for G to display graphs from the DYME.VAM file such as we produced for the G-only version of TINY.

Exercise

12. Convert the model of your imaginary economy to Interdyme.

7. Matrix Tools in Interdyme

Here is a quick overview of the actions, operators, and functions available in Interdyme. Some of them we have seen, but others were not needed in the examples above. You need not learn the exact syntax of each of them; just make a mental note of the possibilities.

If A is a Matrix or Vector and k is a scalar (a float in C terms), then

k*A	multiplies each element of A by k.
A/k	divides each element of A by k.

If A and B are both Matrices or both Vectors of the same dimension, then

A + B	gives the matrix or vector sum
A - B	gives the matrix or vector difference
ebemul(A,B)	gives the element-by-element product
ebediv(A,B)	gives the element-by-element quotient, the elements of A being divided by the corresponding elements of B. If, an element of B is zero, the corresponding element of A is returned in that position.

If A has the same number of columns as B has rows, then

A*B	gives the matrix product.

If A and B have the same number of columns, then

A/B	gives the same thing as ~A*B, that is, the transpose of A multiplied by B, but without actually forming the transpose of A.

Interdyme understands parentheses; if all the dimensions are appropriate, the following is an acceptable statement:

```
A =k* (B+C+D)*(E + F + G*(H+I));
```

If x and y are both Vectors with the same number of elements:

dot(x,y)	gives the inner product as a float.

If A is a Matrix and x a Vector with the same number of elements as A has columns,

A%x	gives the "coefficient" Matrix obtained by dividing each column of A by the corresponding element of x.

For a Matrix A, Vector v, float z, and int k,

v.set(z)	sets all elements of Vector v to z.
A.set(z)	sets all elements of Matrix A to z.
pulloutcol(v, A, k)	pulls column k of A into v.
putincol(v, A, k)	puts v into column k of A.
pulloutrow(v, A, k)	pulls row k of A into v.
putinrow(v, A, k)	puts v in row k of A.
v = colsum(A)	puts the column sums of A into the vector v.
v = rowsum(A)	puts the row sums of A into the vector v.
z = v.sum()	puts the sum of the elements of v into z.
v.First()	gives the number of the first row of v if v is column and vice versa.

If A is a square, non-singular matrix,

!A	gives the inverse of A.
A.invert(i,j)	transforms A into its inverse by Gauss-Jordan pivoting. The pivot operations start in row i and stop when the pivot has been in row j. If these arguments are omitted, the pivoting starts in the first row and continues through the last, to produce the true inverse.

The difference here is that !A does not change A but creates a new matrix for the inverse while A.invert() transforms A into its inverse. Thus, if memory space is scarce, the invert action may be preferable. The algorithm in both cases is Gauss-Jordan pivoting with no niceties. Don't trust it if your matrix poses any problems for inversion.

If A a is either a Vector or Matrix object, then

~A	gives the transpose of A.
A.rows()	gives the number of rows as an integer.
A.columns()	gives the number of columns as an integer.
A.firstrow()	gives the number of the first row as an integer.
A.lastrow()	gives the number of the last row as an integer.
A.firstcolumn()	gives the number of the first column as an integer.
A.lastcolumn()	gives the number of the last column as an integer.

If A is a square Matrix and q and f are Vectors of the appropriate dimension, the equation

q = Aq +f

can be solved by the Seidel iterative method (if it converges) by the function

Seidel(A, q, f, triang, toler)

where triang is an array of integers giving the order in which the rows of A should be selected in the Seidel process, and toler is a float giving the tolerance which is accepted in the iterative solution. Similarly, the equation

p = pA + v

can be solved by PSeidel(A, p, v, triang, toler);

If you need a “scratch” Matrix A or Vector B is not in the VAM file, you can declare it locally in the function where it is needed by:

```
Matrix A(n,m);
Vector B(n);
```

were *n* is the number of rows and *m* is the number of columns.

In the process of debugging a program, it is sometimes useful to display a Matrix or Vector A on the screen. To do so, use

```
A.Display("message", fieldwidth, decimals);
```

To write a Matrix A to a file use

```
writemat(A, filename,fieldwidth, decimals);
```

To write a Vector A to a file use

```
writevec(A, filename, fieldwidth, decimals);
```

For completeness, we mention a function which will be explained in following sections. A Matrix A can be balanced to have the row sums given by Vector a and column sums given by Vector b by the function

```
int ras(A, a, b)
```

If the sum of the elements of a and b are not equal, the user is required to pick which governs.

Finally, I should mention that all of these matrix routines are available in a matrix package call BUMP (Beginner's Understandable Matrix Package) which can be used in C++ independently of the rest of Interdyme. The code of BUMP is carefully explained so that beginners with C++ can learn from it how to write such functions. Understanding how it works, however, is not necessary for using the functions in Interdyme any more than it is necessary to know how Excel is programmed in order to use it.

8. TINY with Endogenous Consumption

At the end of section 3 on the G-only TINY model, I mentioned that you might well feel that Consumption expenditure should depend on labor and capital income from the value-added vectors. I promised that by making the transition to Interdyme, we could introduce that dependence. Now it is time to make good on that promise. In the base year table, the total consumption was 99.3275 percent of the total of labor and capital income. The new version of the model preserves that relationship in the future. We will achieve the desired relation iteratively by summing up the *pce* vector before calculating outputs and income, to get a floating point number, *pcetot*. Then we compute output and the value-added vectors and sum up labor and capital income. If *frac,* the ratio of 99.325 percent of their total, a figure we call *impliedcon* (implied consumption) to *pcetot* differs from 1.0 by more than .0001, then we scale the *pce* vector to have a total equal to *impliedcon,* and repeat the process. An integer variable, *iter,* is used to count the iterations.

In the G editor, open MODELT2.CPP and do a File | Save as to save it as MODEL.CPP. (Do not worry about destroying the previous MODEL.CPP file; it is already saved as MODELT1.CPP.) This new MODEL.CPP file differs from the former one only in the *spin()* function, which is shown in the box below with the new material in boldface type. The first new material is the two lines

```
float pcetot,frac,impliedcon;
int iter;
```

declares the variables we have just mentioned.

The next new code is

```
iter = 0;
top:
iter++;
pcetot = pce.sum();
```

The first line initializes the iteration count each year. The second line is a label; it marks the spot to which the computer returns if the convergence test on *frac* is not met. The first thing the new iteration does is to increment the iteration counter by one (iter++;) and the next is to record in *pcetot* the sum of the *pce* vector. Then follow the lines from the old code to calculated output and value added.

Then comes the rest of the new code:

```
impliedcon = .993275*(lab.sum()+cap.sum());
frac = impliedcon/pcetot;
if(fabs(1.-frac) > .0001 && iter < 100){
    pce = frac*pce;
    goto top;
    }
printf(" Iter %d",iter);
```

Note here the ability of Interdyme to sum up Vectors by .sum() added to the Vector name, and to multiply every element of the *pce* vector by the scalar *frac*. On the other hand, the floating absolute value function

fabs(), the use of && for logical *and* in the condition of the *if* statement, the *if* statement itself, and the *goto* statement are standard C.

Perhaps I should add that C statements are completely free-form; the indentation which you see in the above code is purely for the human eye. But it contributes greatly to the readability of the code, and I strongly urge you to pick one convention and stick with it. Personally, I have a strong preference for the convention you see illustrated here over the more common one which would have written the above *if* statement as

```
 if(fabs(1.-frac) > .0001 && iter < 100)
{
      pce = frac*pce;
      goto top;
 }
```

I find that hard on the eye, but choose your convention and stick with it.

Once you have saved the new MODEL.CPP, just click Model | IdBuild to create the DYME.EXE file and then click Model | Run Dyme to run it. You will get a lot more "chatter" on the screen than previously because Seidel will be printing out its number of iterations on each iteration inside the *spin()* function. The final item for each year will be the number of iterations in *spin()* for that year. You should then graph the outputs and value-added components as before. You will see that the "waves" introduced into the investment function now have a much more pervasive effect than before.

Exercise

Endogenize consumption in the Interdyme model of your imaginary economy.

```
void spin(){
    // Set tolerance for Seidel process
    toler = 1.;
    float pcetot,frac,impliedcon;
    int iter;
    for (t = godate; t<= stopdate; t++) {
        // General start of the spin() function:
        if (MaxFlag == 'n') printf("%d ",t);
        // Load all vectors and matrices.
        load(t);

        iter = 0;
        top:
        iter++;
        pcetot = pce.sum();

        // Particular to TINY:
        fd = pce + gov + inv + ex + im;
        Seidel(AM, out, fd, triang, toler);
        dep = ebemul(depc,out);
        lab = ebemul(labc,out);
        cap = ebemul(capc,out);
        ind = ebemul(indc,out);
        impliedcon = .993275*(lab.sum()+cap.sum());
        frac = impliedcon/pcetot;
        if(fabs(1.-frac) > .0001 && iter < 100){
            pce = frac*pce;
            goto top;
            }
        printf(" Iter %d",iter);

        // General end of the spin() function:
        if(MaxFlag == 'y')
            shiftback(t);
        else{
            // Store the values of vectors and matrices for this period.
            store(t);
            printf("\n");
            }
        }
    }
```

Answers

3.2 The new outputs are
166.14 55.21 222.30 763.57 426.48 206.41 812.58 148.00
and the primary resources required to produce them are
317.24 1351.84 268.34 226.58.

3.3 The net export of depreciation is -5.73.

3.4 The new price vector is (.92 .96 .89 .90 .71 .93 .96 1.00).

3.5 Net export of greenhouse gas production is -31.87.

9. A Historical Note

All of us tend to presume that the world was made the way we found it; if there were input-output tables in it when we arrived, then they must have always been there. Of course, that is not the case. In fact, they are so much connected with the work of one man, Wassily W. Leontief, that without his remarkable contribution they would probably not have been developed until decades later. Born in St. Petersburg in 1906, he was already a university student when the Bolsheviks began taking over the educational program. He joined a group protesting this process, was caught pasting up a poster, spent a while in jail and was periodically jailed and interrogated thereafter. Though deeply interested in the economy of his country and in the efforts at economic planning, he clearly had little to hope for from the Bolshevik government. Even as an undergraduate, however, his paper on "The Balance of the Economy of the USSR" describing efforts in Russia to investigate interindustry relations came to the attention of professors in Germany. When he graduated from the University of Leningrad in 1926, he was offered the possibility of graduate study in Germany, but it was already difficult to get out of the Soviet Union. By an extraordinary turn of fate, he developed a bone tumor on his jaw. It was removed, but the surgeon warned him that he would surely soon die. Armed with the surgeon's written statement, he argued to the officials that he should be allowed to leave the country since he would certainly be useless and possibly expensive to the government. The argument worked, and in 1925 he arrived in Germany with the tumor in a bottle. It was there re-examined and found ... benign! His work in Germany led, via Nanjing, to an appointment at the National Bureau of Economic Research in New York. His theoretical writings came to the attention of the Harvard faculty which offered him an instructorship. He accepted the Harvard offer on the condition that he be given a research assistant to help him build what we would now call an input-output table. The reply informed him that the entire faculty had discussed his request and had unanimously agreed that what he proposed to do was impossible and, furthermore, that even if it were done, it would be useless. Nonetheless, they were so eager to have him come that they would grant the request and hope that he would use the resources for better purposes. He didn't. In 1936, his first results were published; in 1939 a book *The Structure of the American Economy* appeared. It had input-output tables for the United States for 1919 and 1929. The theoretical parts of the book had the major ideas of input-output analysis: coefficients, simultaneous solution, and price equations. During World War II, Leontief constructed, with support of the U.S. Bureau of Labor Statistics (BLS), a 96-sector table for 1939 and, by 1944 was able to study changes in employment patterns which could be expected after the end of the war. In 1947, a second edition of the book appeared with the addition of a 1939 matrix and a comparison of input-output and single-equation projections.[2] In 1973, he was awarded the Nobel prize in economics for this worth. Leontief remained active until shortly before his death in 1999 at the age of 93.

In 1949, a group at the BLS began work on a 400-sector table for 1947. A 190-sector table was published in 1952, but financing -- which had come through the Defense budget -- for the more than fifty people working on the project was discontinued early in the Eisenhower administration, so that neither the full table nor the extensive documentation of the details of its production were ever published.

In other countries, making of tables spread rapidly. They were incorporated in the United Nation's standard System of National Accounts prepared by Richard Stone. In 1950, the first international

[2] The spelling of Leontief's name in Latin letters was for German speakers; English speakers almost invariably mispronounce it, though he never corrected anyone. In *Wassily*, the *W* is pronounced V, the *a* is long as in "father," and the accent is on the *si* which is pronounced "see". In *Leontief* the accent is on *on* and the *ie* is pronounced like the ye in "yet". The final *f* is a soft v.

conference on input-output methods was sponsored by the United Nations; the eleventh (without U.N. support) was held in 1995.

In the late 1950's, Soviet authors, eager to make input-output acceptable in their country, put together a table for the Soviet Union in 1924 and argued that all the essential ideas had originated in the Soviet Union. The difference, however, between what they could find in the literature of that period and Leontief's comprehensive treatment only heightens an appreciation of his contribution.

Gradually, it has come to be recognized that an input-output table is not only useful for economic analysis and forecasting but is also an essential step in making reliable national accounts. The statistical offices of most major industrial countries, therefore, prepare input-output tables, often on a regular basis. Annual tables for France, the Netherlands, Norway, and Japan are prepared as a part of annual national accounting. In the USA, a comprehensive table is made every five years in the years of economic censuses (years ending in 2 and 7) and is used in revising and "benchmarking" the national accounts.

In 1988, the International Input-Output Association was organized as a group of individuals interested in using input-output techniques. In 1989, it began publishing its own journal, *Economic Systems Research*.

The Interdyme modeling system, like the G program, was developed by the Inforum group in the Department of Economics at the University of Maryland. It has been used in developing and linking dynamic input-output models of about twenty countries. Most of these models have been developed and used mainly in the country concerned.

Chapter 15. Matrix Balancing and Updating - the RAS Method

1. The RAS Algorithm

Making an input-output matrix from scratch for a country is a major undetaking often involving a group of ten or more persons for a number of years. By the time the project is finished, the matrix refers to a year that is apt to seem part of ancient history. Hence the question arises, Given an input-output table for a base year, is there a way to update it to a more recent year with less work than making the table from scratch? In this updating, one usually has some data for the more recent year. One wants the matrix for this year, which we may call the target year, to conform to all those data.

Usually those data include at least industry outputs, major GDP components, and value-added by industry. The value-added by each industry can then be subtracted from its output to give the total intermediate inputs by each industry. Thus, we would know the row total for each industry and the column total for each final demand column and for the intermediate use of each industry. An obvious check on the accuracy of this information is that the sum of the row totals equals the sum of the column totals. We will assume that this condition has been met, although meeting it is not always easy except by a rough scaling. Thus, we have the margins or frame for the table for the target year.

An initial guess of the inside of the table for the target year can then be made by assuming constant coefficients for the input-output coefficients and for the shares in each of the final demand vectors. More sophisticated initial estimates could also be made. One could use, for example, consumption functions to "forecast" the purchases of households. However the initial inside elements of the table are estimated, it is almost certain that they will not have the right row and column sums. Adjusting them to make them conform to these control totals is generally done by what has come to be called the RAS procedure, a name derived from notation in Richard Stone's description of the method in *A Computable Model of Economic Growth* (Chapman and Hall, London, 1962). The idea had been mentioned by Leontief in the 1941 edition of *The Structure of the American Economy*, but the idea seemed to pass unnoticed until applied by Stone.

The method is extremely simple in practice. First scale all of the rows so that each has the correct total. Then scale all the columns so that each has the correct total. The row sums are then probably no longer correct, so scale them again, and then scale the columns again, and so on until the scaling factors have converged to 1.0. The matrix at that point has the desired row and column sums. If A^t denotes the flow matrix at stage t of the operation, R^t denotes the row scaling factors at step t arrayed as the diagonal elements of an otherwise zero matrix, and S^t denotes the column scaling factors similarly arrayed, then the flow matrix at the beginning of stage t+1 is

$$\boldsymbol{A}^{t+1} = \boldsymbol{R}^t \boldsymbol{A}^t \boldsymbol{S}^t. \qquad (1)$$

The expression on the right gave rise to the name RAS, which should be pronounced as the three letters, though foreign speakers of English often turn it into one syllable, "ras".

2. Convergence of the Algorithm

The practice is simple, but will the process converge? To answer that question, we will need some notation. Let the original matrix be $\boldsymbol{A}$, whose elements we will denote by a_{ij}, let b be the positive vector of required row sums and c be the positive vector of required column sums. The first condition

is that ***A*** be non-negative. The second is simply that there must exist at least one matrix with zeroes everywhere that ***A*** has zeroes and positive numbers everywhere that ***A*** has positive numbers which has row sums equal to *b* and column sums equal to c.

Notice that this second condition did *not* assume a solution of the form we are seeking, that is, derived from ***A*** by scaling the rows and columns. It does, however, have some important implications. The first is that the sum of the elements of *b* must be the same as the sum of the elements of *c*. A further implication is that, if it is possible rearrange the rows and columns of ***A*** so that an all-zero block appears, then the corresponding subtotals of *b* and *c* must be consistent with those blocks remaining zero while the other cells are positive. For example, if

$$A = \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}$$

then we must also have $b_1 < c_1$. In practice, one insures that the first implied condition (the equality of the sum of row sums and column sums) is met before beginning the RAS calculations. If they fail to converge, then one looks for inconsistences along the lines of the second implication.

The proof of the convergence of the RAS procedure under these general conditions is requires a complicated notation. The essence of the proof, however, can be seen in the special case in which ***A*** is all positive, and we will limit ourselves to that case. (For the general case, see M. Bacharach, *Biproportional Matrices.* Cambridge University Press.)

We will start the process by scaling the rows, then the columns, and so on. In the first row scaling, we choose the first-round row-scaling factors by

$$r_i^{(1)} = b_i / \sum_j a_{ij}. \tag{2}$$

where the superscript on the *r* refers to the iteration number. Then we compute the first-round column scaling factors by

$$s_j^1 = c_i / \sum_i r_i^{(1)} a_{ij}. \tag{3}$$

Then we come back to compute the second-round row scaling factors,

(4)

$$r_i^{(2)} = b_i / \sum_j r_i^{(1)} a_{ij} s_j^{(1)}$$
$$= b_i / \sum_j \frac{b_i}{\sum_k a_{ik}} a_{ij} s_j^{(1)}$$
$$= 1 / \sum_j \frac{a_{ij} s_j^{(1)}}{\sum_k a_{ik}}$$

Thus, we can see that the second-round row factors are reciprocals of convex combinations of the first-round column factors, that is, they are reciprocals of a weighted average of those first-round column factors with positive weights which sum to 1. Thus,

$$\underset{i}{Max}\ r_i^{(2)} \le 1/(\underset{j}{Min}\ s_j^{(1)}) \text{ and } \underset{i}{Min}\ r_i^{(2)} \ge 1/(\underset{j}{Max}\ s_j^{(1)}) \ . \tag{5}$$

By similar reasoning,

$$\underset{j}{Max}\ s_j^{(1)} \le 1/(\underset{i}{Min}\ r_i^{(1)}) \text{ and } \underset{j}{Min}\ s_j^{(1)} \ge 1/(\underset{i}{Max}\ r_i^{(1)}) \ . \tag{6}$$

The inequalities in (6) imply

$$1/\underset{j}{Max}\ s_j^{(1)} \ge (\underset{i}{Min}\ r_i^{(1)}) \text{ and } 1/\underset{j}{Min}\ s_j^{(1)} \le (\underset{i}{Max}\ r_i^{(1)}) \ . \tag{7}$$

Then combining the first inequality of (5) with the second of (7) and the second of (5) with the first of (7) gives

$$\underset{i}{Max}\ r_i^{(2)} \le 1/(\underset{j}{Min}\ s_j^{(1)}) \le \underset{i}{Max}\ r_i^{(1)} \text{ and } \underset{i}{Min}\ r_i^{(2)} \ge 1/(\underset{j}{Max}\ s_j^{(1)}) \ge \underset{i}{Min}\ r_i^{(1)}. \tag{8}$$

In other words, the biggest element of *r* diminishes from iteration to iteration while the smallest rises. Since ***A*** is all positive, all of the inequalities in (5) through (8) will be strict inequalities unless all the elements of ***r*** are equal or all the elements of ***s*** are equal. But if they are all equal, they must be all be equal to 1, for otherwise the scaling would increase or decrease the total of all elements in the matrix, contrary to the fact that, after the first row scaling, the sum of all elements remains equal to the common sum of the vectors ***r*** and ***s***. Since the sequences of $\boldsymbol{r}^{(k)}$ and $\boldsymbol{s}^{(k)}$ vectors both lie in closed, bounded sets, they have limit points. Can these limit points be other than the vectors that are all 1's? No, because at any such point, one more iteration of the process would bring a finite reduction of the maximum element (and a finite increase in the minimum element) of each vector. (This is where we use the all positive assumption to have strict inequalities in (8).) Thus, for points sufficiently close to these limit points, the next iteration must also bring lower maximal and higher minimal elements than

those of the limit point, contrary to the limit point being a limit point. Therefore the unique limit of each sequence of vectors is a vector of ones.

Thus the convergence is proven for the case of all positive ***A***. The proof is similar for ***A*** with some 0 elements, but in this case, it may require several iterations to get a finite reduction in the maximal elements of ***r*** and ***s***.

In practice, the condition that the sum of ***b*** equals the sum of ***c*** is checked and assured before the iterative process begins. The initial ***r*** and ***s*** vectors should be reported by the program because they often indicate discrepancies between ***b*** and ***c*** vectors and the initial ***A*** matrix. Once the iterations start, the largest and smallest elements of the ***r*** and ***s*** vectors should be reported every five or ten iterations. It is common to observe "wars" between a row control and a column control when one element looms large in both its row and column but the control totals for the two are quite different. Such "wars" are symptomatic of a failure of the second assumption and an indication that the ***b*** and ***c*** vectors should be revised.

It should be noted that the RAS procedure works for rectangular matrices just as well as for square ones. It is also useful in making input-output tables and the bridge matrices used to convert investment by investor to investment by product bought or consumption by consumer categories to consumption by product categories used for productive categories.

3. Preliminary Adjustments Before RAS

It often happens in updating or making tables that one has better information about some cells than about others. For example, in updating the a table with a Glass row, we may have quite good information on the sales of glass products to Beer, because we have information on the production of glass beer bottles. In this case, we can simply remove the "relatively well-known flow" from both its row and column control, perform the RAS balancing on the remaining flows, and then put back in the known flow.

The problem with this procedure is that the "relatively well-known flows" tend to be big flows. If they are not quite consistent with the row or column controls, then removing them requires that all of this inconsistency should be attributed to changes in the remaining small flows. Thus, the small flows can be pushed about rather considerably. This problem can be reduced by a preliminary scaling of the relatively well-know flows before removing them from the process. To describe this adjustment, let R_i be the sum (in the base year) of the relatively well-known flows in row i ; S_i , the sum of the other flows; and B_i , the row control. Then let

$$\alpha_i = \frac{R_i}{R_i + S_i} \tag{9}$$

and define z_i as the solution of

$$S_i z_i + R_i z_i^{\alpha_i} = B_i. \tag{10}$$

The value of *z* which satisfies (10) is readily found by Newton's method. We then scale all the relatively well-known flows by $z_i^{\alpha_i}$ and all the other flow by z_i. By (10), the row will then have the correct sum. By (9), $0 \le \alpha_i \le 1$. If $0 < \alpha_i < 1$, then $z_i^{\alpha_i}$ is closer to1.0 than is z_i; that is to say, the relatively well-known flows are scaled *less* than the other flows. They are however, scaled somewhat. If they account for a small fraction of the total of all flows in the row, they will be scaled but little; if they account for much of the row, they will be scaled almost as much as the other flows.

After this preliminary scaling, the known flows can be removed for the rest of the RAS process. While this scaling may seem a bit arbitrary, in practice it has given plausible results in many applications. In fact, it worked so well that the first person working with it, Thomas Reimbold, felt that the *z* must stand for *Zauber*, "magic" in his native language, and the procedure is therefore often referred to as the *Zauber* process.

Chapter 16. Trade and Transportation Margins and Indirect Taxes

1. Trade and Transportation Margins

A perennial problem in applied input-output analysis is the treatment of trade and transportation margins and of indirect taxes. The problem is nicely illustrated with transportation costs. If output is valued at the producer's price — the price at the factory gate, so to speake — then the cost of transporting the goods to the user must be considered to be paid separately by the purchasing industry. Thus, the cost of the rail services used in hauling the coal used by electric power plants shows up as an input of rail transportation into electric generation. The cost of hauling generation equipment to and from the utilities' repair facilities would appear in the same cell. Similarly, the cost of hauling coal to a steel mill and of hauling iron ore to the same mill will appear in the same cell.

The problems with this treatment are (1) it puts quite diverse activities into the same cell and (2) the table does not reflect the way the rail industry thinks about its business. It thinks in terms of products hauled — and prepares statistics on products hauled, not on industries to which it delivers. (Despite these problems, this treatment is the one most commonly followed.)

All of the problems apply with equal force to all the other transportation margins and to wholesale and retail trade margins.

One alternative is to change the measure of output of the industry to include the cost of delivering the product to the user. One disadvantage of this treatment is that it removes the numbers in the input-output table one step further from the numbers in terms of which people in the industry think, namely in producer prices. Another problem is that transportation margins may be very different for a dollar's worth of product delivered to different users. The transportation cost of oil delivered to an electric utility by pipeline from a marine terminal may be very different from delivering by truck or rail to a small industrial user.

A better alternative is to add another dimension to the input-output tables. Thus, corresponding to each cell of the tables we have considered so far there would be a vector. The first entry in the vector would be the transaction in producer prices; the second entry would show the rail margin; the third, the truck margin; the fourth, the air freight; and so on through the wholesale and retail trade margins. In effect, we would have a table with layers, the first layer for the producer price transaction, the second for the rail margins, and so on. In fact, the benchmark tables for the United States are prepared with all this information. It has not been commonly used because the size of the matrices involved has been, until fairly recently, large relative to the power of the computers available. That constraint has now been effectively removed, and we may ask, How would we in fact compute with such a layered table?

If $\boldsymbol{A}$ represents the coefficient matrix in producer prices and $\boldsymbol{T}_i$ represents the i^{th} layer of transportation and trade margin coefficients, then the fundamental input-output equations become

$$q = Aq + \sum_i S_i T_i q = (A + \sum_i S_i T_i) q = f$$

where $\boldsymbol{S}_i$ is a matrix with 1's in the row which produces the service distributed by layer i and elsewhere all zero. The matrix $(A + \sum_i S_i T_i)$ is, in fact, the matrix in producer prices with which it has been traditional to compute. What is gained by distinguishing the layers is not a correction of the traditional computations but rather a better description of what the flows are and a better basis for studying changes in coefficients in the $\boldsymbol{T}_i$ matrices.

2. Indirect Taxes, Especially Value Added Taxes

Indirect taxes such as property taxes or franchise taxes are always and without problems treated as a component of value added, along with depreciation, profits, interest, and labor compensation. Excise taxes such as those on gasoline, alcohol, and tobacco are usually similarly treated, but with less justification, because some uses of these products are exempt. For example, gasoline used to power agricultural machinery or exported whiskey or cigarettes are exempt. Thus, these taxes should also be treated as a layer of the table, since they are not uniform for all cells. Retail sales taxes are usually treated as a component of value added by Retail trade. This treatment assumes that the tax is proportional to the retail margin in all products in all cells. In fact, there are different tax rates on different products, and some products are sold by retail establishments for intermediate use without retail sales tax.

The greatest problems, however, have probably been created by the value added tax (VAT) in the tables of countries which use this tax, a group that now includes all members of the European Union and numerous other countries. Producers pay VAT on the value of their sales but may deduct the VAT paid on their purchases. VAT is not charged on certain products, such as health services. Nor is it charged on exports. Many European input-output tables have been published in producer prices plus non-deductible VAT. That practice meant that the cell for paper products sold to the hotel industry did not contain VAT, because the VAT on those sales was deductible from the VAT owed by the hotels. The cell for paper products sold to hospitals, however, contained VAT, because the hospitals owed no VAT from which the VAT on the paper products could be deducted. Similarly, since households owe no VAT, they cannot deduct the VAT on the paper products they buy, so the VAT is included in the cell showing the sales of paper products to households. Thus, the cells in the paper products row of such a matrix have very diverse levels of VAT content. That means that the valuation of the product across the row is not homogeneous. It takes more wood pulp to make a dollar's worth paper towels used by a hotel than to make a dollar's worth of paper towels used by a hospital or household, because a significant portion of their dollar goes to VAT. This heterogeneity in the pricing in the row is obviously detrimental to the accuracy of the input-output calculations. The solution to the VAT problem is simply to create a VAT layer of the table.

Chapter 17. Making Product-to-Product Tables

1. The Problem

Makers of input-output tables often find data on inputs 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 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 necessary to meet a final demand given in product terms?

One way is to consider that each product will be produced in the various industries in the same proportion as in the base year of the table. This assumption is used, for example, in computable general equilibrium models based on social accounting matrices that explicitly show the Make and Use matrices. This assumption, however, can produce anomalous — not to say silly — results. In the above example, an increase in the demand for cheese would automatically and immediately increase demand for chocolate, strawberries, and sugar. That is nonsense. There must be a better way to handle the problem.

This highly unsatisfactory situation has led to efforts to make a product-to-product matrix. Indeed, the problem is so well recognized that the "Transmission programme of data" of the European system of accounts requires that all national statistical offices of the member states of the European Union transmit "symmetric" input output tables to Eurostat every five years. No real advice, however, is offered by Eurostat to the statistical offices on how to make these product-to-product tables. This paper offers a valuable tool for the process. ("Symmetric" is here intended to mean that the same concepts are used in both rows and columns. Its use as applied to these *matrices* is both highly confusing and not descriptive. Since it is the *nature* of the rows and columns that is the same, not their measure, *symphysic* would be both a better characterization and less confusing.)

To make such a matrix, we need to employ an additional assumption. There are basically two alternatives:

9. The product-technology assumption, which supposes that a given product is made with the same inputs no matter which industry it is made in.
10. The industry-technology assumption, which supposes that all products made within an industry are made with the same mix of inputs.

The *System of National Accounts 1993* (SNA) reviews the two assumptions and finds (Section 15.146, p. 367) "On theoretical grounds, ... the industry technology assumption performs rather poorly" and is "highly implausible." (Section 15.146, p 367) “From the same theoretical point of view, the product (commodity) technology model seems to meet the most desirable properties .... It also appeals to common sense as it is found *a priori* more plausible than the industry technology assumption. While the product technology assumption thus is favoured from a theoretical and common sense viewpoint, it may need some kind of adjustment in practice. The automatic application of this method has often shown results that are unacceptable, insofar as the input-output coefficients appear as extremely improbable or even impossible. There are numerous examples of the method leading to negative coefficients which are clearly nonsensical from an economic point of view." (Section 15.147)

Since 1967, the Inforum group has used a "semi-automatic" method of making "some kind of adjustment" in calculations based on the product-technology assumption, as called for by the SNA. We have used it with satisfactory results -- and without a single negative coefficient -- on every American table since 1958. The method was published in Almon 1970 and in Almon *et al.* 1974. Despite this long and satisfactory use of the method, it seems not to have come to the attention of the general input-output community. In particular, the authors of the section quoted from the SNA seem to have been unaware of it. The purpose of this note is to record the method where it is more likely to come to the attention of anyone working in input-output. At the same time, it expands the previous exposition with an example, provides a computer program in the C++ language for executing the method, and presents some of the experience of applying the method to the 1992 table for the USA.

2. An Example

An example will help us to visualize the problem. The Table 1 below shows the Use matrix for a 5-sector economy with a strong concentration in dairy products, especially cheese and ice cream.

Table 1. The Use Matrix

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. (Rennet is a substance used to make milk curdle. It is commonly used in making cheese but never in ice cream.) The Make matrix, shown in Table 2 below, confirms that cheese is being made in the ice cream industry and ice cream in the cheese industry.

Table 2. The Make Matrix

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. It also shows that, of the total output of 90 by the cheese industry, 78 percent (70/90 = .77778) was cheese and 12 percent ice cream. 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 shown in Table 3.

Table 3. The M Matrix

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. That is, we will make the "product-technology assumption." If it is true and the matrices made well, then there exists a "recipe" 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,

$$U = RM' \tag{1}$$

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

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

For our example, $(M')^{-1}$ is given in Table 4.

Table 4. M′ Inverse

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

Table 5. The R or "Recipe" Matrix

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. Unfortunately, as indicated by the quotation from the SNA, it is rare for the results to turn out so nicely.

Indeed, just a slight change in the U matrix will show us what generally happens. 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.

Table 6. An Alternative Use Matrix

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

Table 7 shows what the R matrix would have been:

Table 7. An Impossible R Matrix

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

Here we find the infamous small negative flows. It is not hard to see how they arise. While it is conceivable that the Cheese industry does not produce chocolate ice cream, it is also very easy for the table makers 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.

The negatives have driven at least some statistical offices to the industry-technology assumption. The so-called commodity-to-commodity matrix, C, derived from this assumption is

$$C = UN', \tag{3}$$

where N is the matrix derived from the Make matrix by dividing each row by the row total. For example, the Cheese column of C is $C_1 = .77778U_1 + 0.14285U_2$ because 77.778 percent of the product of the first industry is cheese and 14.285 percent of the product of the second industry is cheese. The result of applying this assumption to our example is Table 8.

Table 8. The Mess Made by the Industry Technology Assumption

C Indust. Tech.	Cheese	Ice cream	Chocolate	Rennet	Other
Cheese	0	0	0	0	0
Ice cream	0	0	0	0	0
Chocolate	8.254	31.746	0	0	0
Rennet	11.746	8.254	0	0	0
Other	32.063	67.936	30	5	0

This "solution" has made matters worse. 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 *8.25 units of chocolate went into producing pure cheese*! Not into the Cheese *industry* but into the *product* cheese! And 8.25 units of rennet went into producing curdled ice cream! To call the result a product-to-product table would be little short of scandalous.

Fortunately, we do not have to choose between this sort of massive nonsense and negative flows. It is perfectly easy to rely mainly on the product-technology assumption, yet avoid the negatives, as we will now show.

3. The No-Negatives Product-Technology Algorithm

We wrote the basic equation relating U, M, and R as equation (1) above. It will prove convenient to rewrite equation (1) as

$$U' = MR'. \tag{4}$$

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

$$U'_i = MR'_i . \tag{5}$$

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 distribution of the same product in row i of the R matrix. We can simplify the notation by writing

$$u = U'_i \quad \text{and} \quad r = R'_i ; \tag{6}$$

then the previous equation becomes

$$u = Mr \tag{7}$$

or

$$0 = -Mr + u \tag{8}$$

and adding r to both sides gives

$$r = (I - M)r + u . \tag{9}$$

Save in the unusual case in which less than half of the production of a product is in its primary industry, 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 a by well-known theorem. (If the share of the total production of a particular product coming from the industry to which it is primary is x, then the absolute value of the diagonal of (I - M) for that product is | 1 - x | and the sum of all the absolute values of off-diagonal elements in the column is |1 - x |, so the total for the column is 2*| 1 - x |, which is less than 1 if x > .5.) We start this process with

$$r^{(0)} = u \tag{10}$$

and then define successive approximations by

$$r^{(k+1)} = (I - M)\, r^{(k)} + u . \tag{11}$$

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:

$$r_j^{(k+1)} = u_j - \sum_{\substack{h=1 \\ h \neq j}}^{n} m_{jh} r_h^{(k)} + (1 - m_{jj})\, r_j^{(k)} \tag{12}$$

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:

$$(1 - m_{jj})\, r_j^{(k)} = \sum_{\substack{h=1 \\ h \neq j}}^{n} m_{hj} r_j^{(k)} \tag{13}$$

because

$$\sum_{h=1}^{n} m_{hj} = 1. \tag{14}$$

It is now clear how to keep the negative elements out of *r*. When the "removal" term, the second on the right of (12), 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. We will call the situation where the plundered industry runs out of the product being removed before all claims are satisfied a "stop".

The process can also be applied to the rows of the value added part of the matrix. It is not certain, however, that the column sums of the resulting value-added table will match the value added as calculated from product output minus intermediate input. This value-added matrix will generally require RAS balancing to make it consistent with the product-to-product intermediate table.

4. When Is It Appropriate to Use This algorithm?

This algorithm is appropriate where the product-technology assumption itself is at least approximately true. Essentially, it allows there to have been slightly different technologies in industries where assuming *strictly* the average product technology would produce negatives. It is appropriate where the negatives arise because of inexactness in making the tables or because of slight differences in technologies in different industries. Applied to the Use matrix of either Table 1 or Table 6, this method gives the "neat" Recipe matrix of Table 5 with no rennet in ice cream and no chocolate in cheese. It never produces negative entries nor positive entries where Use has a zero. The row totals are unaffected by the process. It is, moreover, equivalent to deriving Recipe from equation (1) if no negatives would arise, so that if the product-technology assumption is strictly consistent with the Use and Make tables, the method produces the true matrix. It may even produce a correct Recipe matrix from a faulty Use matrix — as it has perhaps done in our example — so that equation (1) could be used to revise the estimate of the Use matrix.

Certain accounting practices, however, may produce situations which appear to be incompatible with the product-technology assumption, even though the underlying reality is quite compatible. For example, local electric utilities generally buy electricity and distribute it. In the U.S. tables, they are

shown as buying electricity (not coal), adding a few intermediate inputs and labor, and producing only a secondary product, electricity, which is transferred, via the Make matrix, back to electricity. Looked at mechanically, this method of making electricity is radically different from that used in the Electricity industry, which uses coal, oil, and gas to make electricity, not electricity itself. If our algorithm is applied thoughtlessly to this situation, it cannot be expected to give very sensible results.

Fortunately, it is easy to generate signs of this sort of problem. One can compute the new Use matrix implied by equation (1) with the Recipe matrix found by the algorithm and the given Make matrix. This "NewUse" matrix can then be compared with the original Use matrix and the causes of the differences investigated. We will follow this procedure in next section on the experience of using the method on the 1992 tables for the USA.

To fix the problem in the above example about electricity, we have only to consider the output of the State and local utilities as production of their own primary product, which is then sold, via the Use matrix — not transferred via the Make matrix — to the Electricity industry. In essence, we use the industry technology assumption for the local electric utilities — and for all other industries where all of the output is secondary. The industry technology assumption may also be preferable for transfers to some catch-all sectors such as "Miscellaneous food preparations" (SIC2099), which includes such disparate products as vinegar, yeast, Chinese noodles, and peanut butter. It is probably just as reasonable to suppose that a product transferred into this industry is made with the average technology of the industry where it is made as with the average technology of this catchall sector. Indeed, this sort of industry can produce the reverse of the negatives problem. For example, because of the importance of peanut butter in this industry, it has significant inputs of oil seeds. Now the no-negatives algorithm will not pull oil seeds out of the "Macaroni, spaghetti, vermicelli, and noodles" industry, (SIC2098), (which used no oil seeds) just because it transferred some Chinese noodles to 2099. But neither will it take out an adequate amount of flour for those noodles, because flour is quite unimportant in the 2099 input mix. This problem shows up only indirectly by substantial oil-seed inputs to many food industries in the NewUse which transferred products to 2099 but, in fact, used no oil seeds. That is a signal to switch to the industry technology for these transfers by converting them to sales in the Use matrix.

Thus, in the use of this method, a number of iterations may be necessary. Changes in concepts, in treatments of some transactions, and occasionally in underlying data may be necessary. Although the calculation of the non-negative Recipe matrix is totally automatic, it may be necessary to make several runs to get acceptable results.

In this process, it must be recognized that a nice, clean accounting system may not be operational, that is, it may not provide by itself a simple, automatic way to go from final demand vectors specified by products to total outputs of those products. We may have to change slightly some of the concepts in the accounting system to make it operational. In making the change required for the Electricity example, we have messed up the neat accounting concept of the Electricity column of the Use matrix as a picture of what came into a particular group of establishments. We have, however, taken a step toward creating what might be called an operational Use matrix. I do not say, therefore, that statistical offices should not produce pure accounting Use matrices. But I do feel that they should also prepare the operational use matrix and the final product-to-product matrix, for in the process, they will learn about and deal with the problems which the users of the matrix will certainly encounter. They may even discover and correct errors in their work before they are discovered by their users.

This process is totally inappropriate for handling by-products such as hides produced in the meat packing industry or metal scrap produced in machinery industries. Their treatment is a different subject.

5. A Brief History of the Negatives Problem

The idea to compute R from equation (1) seems to have been first put in print by Van Rijckeghem (1967). He realized that there could be negatives but did not think they would be a serious problem. The idea of using equation (1) in this way, however, must have been in the air, for by early 1967, I had used it, without thinking that it was original, found negatives, and started work on the algorithm presented here.

The problem was encountered by ten Raa, Chakraborty and Small [1984] in the course of work which was primarily concerned with identifying by statistical means true by-products. They note the existence of the method presented here but write,

> [Almon] iterates truncated Neumann series in which matrix multiplications are carried out only to a limited extent to avoid negatives. This arithmetic manipulation goes without justification, is arbitrary and depends on the choice of [make matrix]-decomposition as well as the iteration scheme.

I do not believe that any of this comment is correct. The Neumann series is the expansion $(I - A)^{-1} = I + A + A^2 + A^3 + \ldots$. The algorithm used here makes no use of this series; rather it uses the Seidel procedure. There are no matrix multiplications, nor is there is any equivalence between a "limited" number of terms in the Neuman series and the Seidel solution. The procedure is carried to convergence. We have seen that the procedure has a perfectly reasonable economic interpretation; indeed, it arose from the economic interpretation of the Seidel procedure. The only thing perhaps "arbitrary" is that 0 is considered a reasonable input flow while negatives are considered unreasonable. I do not know what the "[make matrix]-decomposition" refers to, but I can assure the reader that the solution does not depend on the "iteration scheme." While I could not see how it could, given that it is carried to convergence, I changed the program and ran the "robberies" in the opposite order. The answers were identical.

The ingenious attempt of ten Raa [1988] to modify elements of the matrices in such a way as to find a most probable U matrix consistent with a non-negative R should be mentioned even though it ended, in the author's view, in frustration.

Rainer and Richter [1992] have documented a number of steps which they took towards making what I have called here the operational Use and Make matrices. Such steps should certainly be considered and applied if need. These authors still ended up with hundreds of negative flows in the R matrix because they were using just equation (1). At that point, the process described here could have been applied.

Steenge and Konijin [1992] point out that if the R matrix computed from equation (1) has any negatives in it, then it is possible to change the levels of output of the various industries in such a way that more of all products is produced *without* using more of all inputs. They feel that it is implausible that such a rearrangement is possible and observe that perhaps the negatives "should not be regarded as rejecting the commodity technology assumption, but as indicators of flaws in the make and use tables." (p. 130). I feel that there is much merit in that comment. It seems to me that the right time

and place to use the algorithm presented here is in the process of making the tables. If there are not good statistical grounds for preferring the original Use matrix, the recomputed NewUse might well be argued -- following the reasoning of Steenge and Konijn -- to be a better estimate.

The caveat here is that there may well be cases where it really would be possible to increase the outputs of all products while using less of some product. For example, if there are shoes made in the Plastics products industry without any use of leather, while the Footwear industry uses leather, then by moving shoe production from Footwear to Plastic products it may be possible to produce more of all products while using less leather. Where such cases arise, a different solution is necessary, for example, moving the shoes made in the Plastics products industry together with their inputs into the Footware industry or insisting that the two kinds of shoes are separate if substitutable products.

6. Application to the U.S.A Tables for 1992

The method described here has been applied to all of the USA tables since 1958 with experiences broadly similar to those described here for the 1992 table. This table has 534 sectors, counting some construction sectors which have no intermediate sales. Of these 534, 425 have secondary production. Of the 283,156 possible cells in a 534 X 534 matrix, the Use matrix has 44,900 non-zero cells, and the Make matrix has 5,885. The matrix was produced in two versions. In one, certain activities, such as restaurant services of hotels, were removed from the industry where they were produced (Hotels) and put into the sector where these activities were primary (Restaurants). In the other, these activities were left in the industry where they were conducted. The first version was designed to make the product-technology assumption more valid, and it has been used here. The matrix also puts true by-products (such as hides from meat packing) in a separate row, not one of the 534 considered here.

To try to convey a feeling of what it is like to work with the algorithm, we will look at the process midway along, rather than at the very beginning or the somewhat polished end. That is, some adjustments in the Use and Make matrix from which the algorithm starts will have already been made. As a result of this application, further adjustments will be suggested before the next application.

Before this application of the algorithm, the output of industries which had only secondary production had been changed, for reasons explained above, to be primary and the flows moved from the Make to the Use matrix.

In the following rather detailed descriptions, necessary to give a picture of what the process is really like, I will, to avoid confusion, capitalize the first letter of the first word in industry names but not in product names.

The industry Water and sewer systems failed to satisfy the requirement that at least half of the output of a product should be in the industry where it is primary. Indeed, some 85 percent of this product's output comes from Other state and local enterprises, and the iterative procedure failed to converge for a few rows until this secondary transfer was converted into a primary sale. Production of secondary advertising services, which occurred in many sectors, was also converted to a primary product of the producing industry and "sold" via the Use matrix to the Advertising industry. Secondary production of recreational services in agricultural industries was similarly converted. Much of the output of the several knitting industries had been treated originally as secondary production, and these had been changed to primary sales before the calculations shown here. Finally, the diagonals of many columns of the Use matrix are large, in part because intra-firm services, such as those of the central offices,

often appear there. Thus the same sort of service that is on the diagonal of industry i is also on the diagonal of industry j. In this case, the product-technology assumption does not apply, not because it is untrue, but because of the way the table was made. Until we are able to obtain tables without this problem, we have just removed half of the diagonals from the Use table before calculating Recipe, and have then put back this amount in both of these matrices and in the NewUse matrix.

The data in both Use and Make tables were given to the nearest 1 million dollars, and all dollar figures cited here are in millions. The convergence test in the iterative process was set at one tenth of that amount, .1 million dollars. The iterative process converged for most rows of the R matrix in less than five iterations. The most iterations required for any row was 15.

The resulting Recipe matrix looks very similar in most cells to the original Use table. The Recipe matrix contains, of course, only non-negative entries and can have strictly positive entries only where U has positive entries. It may, however, as a result of the "robbing" process, have a zero where U has a positive entry. In all, there were only 95 cells in which Recipe had a zero where Use had a positive entry.

Although it is the Recipe matrix that we need from this process, it is also interesting, as noted above, to compare the original Use matrix with what we may call NewUse, computed by the equation 1 by NewUse = Recipe*Make'. The difference between Use and NewUse shows the changes in the Use matrix necessary to make it strictly compatible with product-technology assumption, the given Make matrix, and the calculated Recipe matrix. If there was no "stop" in a row, the two matrices will be identical in that row. There were 118 such identical rows, 109 of them having no secondary output.

In the other rows, these differences turn out to be mostly small but very numerous. The first and most striking difference is that NewUse has almost twice as many non-zero cells as does Use. Nearly all of these extra non-zeros are very small, exactly the sort of thing to be reasonably ignored in the process of making a table. But it is precisely this "reasonable ignoring" that leads to the problem of many small negatives in the product-to-product tables calculated without the no-negatives algorithm.

To get a closer look at how Use and NewUse compare, we may first divide each column by corresponding industry's output and then look at the column sums of the absolute values of the differences of individual coefficients in the column. This comparison is shown in Table 9. Clearly the vast majority of industries show only small differences compatible with "reasonable ignoring" of small flows in the Use matrix. They, therefore, cast no serious doubt on the product-technology assumption or the usability of the Recipe matrix obtained by the no-negatives algorithm. If what we are interested in is the R matrix, we can ignore the small differences between Use and NewUse.

Table 9. Comparison of Use and NewUse

Sum of Absolute Differences	Count
.050 - .250	17
.030 - .050	24
.020 - .030	54
.010 - .020	117
.000 - .010	312

Table 10. Largest Differences between Use and NewUse

Sum |dif|	Column numb.	Column Name	Largest single difference Row	| dif |	Row name
0.250	272	Asbestos products	31	0.023	Misc. nonmetallic minerals
0.232	88	Sausages	3	0.151	Meat animals
0.167	125	Vegetable oil mills, nec	15	0.074	Oil bearing crops incl s
0.118	493	Auto rental & leasing	232	0.025	Petroleum refining
0.088	128	Edible fats and oils, nec	15	0.043	Oil bearing crops incl s
0.088	126	Animal & marine fats	126	0.038	Animal & marine fats &
0.086	87	Meat packing plants	3	0.057	Meat animals
0.079	285	Primary metals, nec	22	0.006	Iron & ferroalloy ore m
0.079	225	Manmade organic fibers	212	0.036	Indl chem: inorg & org
0.074	450	Transportation services	232	0.019	Petroleum refining
0.068	123	Cottonseed oil mills	5	0.048	Cotton
0.065	357	Carburetors, pistons,	391	0.011	Electronic components
0.060	99	Pickles, sauces	1	0.011	Dairy farm products
0.060	95	Canned & cured sea food	19	0.039	Commercial fishing
0.059	139	Yarn mills & textile fini	212	0.035	Indl chem: inorg & org
0.055	459	Sanitary services, steam	413	0.018	Mechanical measuring devices
0.051	248	Leather gloves	244	0.012	Leather tanning

There are, however, a few cases that should be looked at more closely. Table 10 shows a list of all of industries which had a sum of absolute differences greater than .050. We will look at the top five.

For Asbestos products, the cause of the difference is quickly found. The fundamental raw material for these products comes from industry 31 Misc. non-metallic minerals. Over forty percent of the output of asbestos products, however, is produced in industry 400 Motor vehicle parts and accessories, but this industry buys neither miscellaneous non-metallic minerals nor asbestos products. In other words, it seems to be making almost half of the asbestos products without any visible source of asbestos. This anomaly seems to me to be an oversight in making the Use matrix which should be simply corrected. If our only interest is the Recipe matrix, the algorithm seems to have computed pretty nearly the right result from the wrong data. On the other hand, if we want to correct the Use table, NewUse, gets us started with the right entry for Misc. non-metallic minerals into both Motor vehicle parts and Asbestos products. To keep the right totals in these two columns of Use will require manual adjustments.

The second largest difference between Use and NewUse shown in Table 10 is in the input of meat animals into Sausage. The Sausage industry is shown in the Use matrix to buy both animals ($655) and slaughtered meat ($9688). It had a primary output of $13458 and a secondary output of $2612 of products primary to Meat packing. Meat packing had a secondary output of $4349 of sausage. Now in Meat packing, the cost of the animals is over eighty percent of the value of the finished product, so the purchases of animals in the Sausage industry is insufficient to cover even the secondary meat output of this industry, not to mention making any sausage. In making Recipe, the input of animals directly into sausage is driven to zero and cut off there rather than being allowed to become negative. Then when NewUse is made, the direct animal input for all the secondary production of meat packing products is put in, thus making a flow some six times as large as the purchase of meat animals by the Sausage industry in the original Use matrix.

What I believe to be really happening here is that Sausage plants are mostly buying halves of slaughtered animals from meat packers, selling off the best cuts as a secondary product, and using the rest to make sausage. Over in the Meat packing plants, the same thing is happening. Fundamentally, there is only one process of sausage making. The question is how to represent it in the input-output framework. The simplest representation of it in the Use matrix would be to have packing houses sell to sausage plants only the meat that would be directly used in sausage. The rest, the choice cuts sold off as meat by Sausage mills, would simply be considered sold by the packers without ever passing through the Sausage mills. The industry output of Sausage mills is reduced but cost of materials (namely, meat) is reduced by exactly the same amount, so there is no need to adjust other flows. Product output of meat is reduced, but not the industry output. Thus, a slight adjustment in the accounting makes it broadly compatible with the product-technology assumption. The seventh item in Table 10, by the way, is just the other side of this problem.

The third largest of the discrepancies lies in row 16, oil-bearing crops, of industry 125 Vegetable oil mills n.e.c (not elsewhere classified). The differences in the underlying flows is not large, $298 in Use and $251 in NewUse, but it turns up in Table 10 because the cost of these oil crops is such a large fraction of the output of the Vegetable oil mills. A comparison of the oil-bearing crops row of Use and NewUse shows that NewUse has a number of small positive entries for industries where, as for Cheese, Use has a zero and where, moreover, it is highly implausible that there was any use of oil seeds. On the other hand, most of the large users of oil seeds, like Vegetable oil mills have had their usage trimmed back. The key to what is going on is found in industry 132 Food preparations n.e.c.. In Use, this industry bought $558 from oil bearing crops, nearly twice the consumption of the vegetable oil mills themselves. Peanut butter, as noted above, is in this catchall industry. That fact, by itself, is not a problem. The problem is that about a quarter of the production of products primary to this industry are made in other industries. In fact, most of the food manufacturing industries have some secondary production of the miscellaneous food preparations. Probably "preparations" made in the Cheese industry are quite different from those made in the Pickles industry. And it certainly makes no sense to spread oil seed inputs all over the food industries. Here we have a clear case of the inapplicability of the product-technology assumption if all these secondary products are considered to be truly the same product. On the other hand, as argued above, the very heterogeneity of the products makes it appropriate to consider each as a primary product of the industry which produces it and then "sell" it, via the Use matrix, to Food preparations for distribution. In the next pass at making Recipe, this change is to be made.

The vegetable oil industries also present another interesting case of apparent but perhaps not real violation of the product-technology assumption, which shows up in the fifth item in Table 10. Industry 125 Vegetable oil mills n.e.c. has inputs of oil-bearing crops, cotton, and tree nuts totaling $437. It uses these oil sources to produce a primary output of $572. Industry 128 “Edible fats and oils” produces $92 of products primary to 125 without a penny of any of these inputs! Surely this is flat violation of the product-technology assumption. But is it really? "Edible fats and oils" buys lots of the products primary to Vegetable oil mills. Thus, it is entirely possible to have two bottles of chemically identical oil made of identical raw materials by identical refining processes but with one bottle made entirely in Vegetable oil mills while the oil in the other bottle was pressed in those mills and then sold to Edible fats and oils for finishing. We might call this situation “trans-market product technology.” Our algorithm gave the right answer for the Vegetable oil mills column of Recipe, that is, it combined output of products primary to the oil mills with the inputs of oil sources which this industry had.

The fourth largest discrepancy in Table 10 is for the gasoline input into Automobile renting and leasing. Use shows $1131; Recipe ups that to $1197.2; but NewUse cuts it back to $565.5. What happened? The problem is that slightly more than half of the output Auto renting is produced in Credit agencies, with a minuscule input of gasoline. When NewUse is made, more than half of the gasoline in Recipe is allocated over to Credit agencies. Here we are confronted with a failure of the product-technology assumption not because of different processes for producing the same product but because two quite different products have been called one and the same in the accounting system. The output of the Credit agencies, long-term leasing, is quite distinct from the short-term renting, which is were the gasoline was used. The best solution would be to recognize the difference of the two products. Short of that, the worst of the problem can be fixed by turning the secondary transfer from Credit agencies to Automobile rental into a primary flow. The present Recipe matrix, incidentally, is about right in the gasoline row but makes no connection between a final demand for automobile renting and leasing and the output of credit agencies.

From these five or six cases, we see that our algorithm cannot be expected to give usable results on the first try. The problems are likely to lie, however, neither in the fundamental economic reality nor in the algorithm, but in an accounting system which needs a few modifications in Use and Make to make it operational in our sense. Most importantly, the algorithm gives us the means to identify the places that need attention and a way of progressing systematically through the problems. It also provides a way of producing a final, non-negative Recipe matrix that implies a NewUse matrix close enough to the modified Use matrix that the differences can be safely ignored.

Making an input-output table requires fussing over details, and making a good Recipe matrix with the algorithm presented here is no different in this respect from any other part of the process. Use of the algorithm reveals and pinpoints problems. Moreover, the important problems are likely to be small in number. We have covered all of those causing a difference of as much as .100 between columns of Use and NewUse. To get to a Recipe table we would be ready to accept might require another week's work. But in the total effort which went into making this table, that is minuscule. Most importantly, the use of the algorithm gives us a way to work on the problems rather than just wring our hands over negatives.

In this sense, this algorithm has performed satisfactorily over many years on every U.S. table since 1958. The use of the method seems to me to deserve to become a standard part of making input-output tables and, in particular, for making product-to-product tables.

7. The Computer Program

The C++ code for this algorithm, using functions from BUMP, the Beginner's Understandable Matrix Package, for handling matrices and vectors, is given below. It is reproduced here because the code shows more clearly than the verbal or formulaic description exactly what is done. The program and the supporting BUMP code made be downloaded from the Inforum Internet site: www.inforum.umd.edu. The main program here reads in the matrices that were used in the examples. The main program for the actual calculations of the full-scale American matrices is significantly larger and has various diagnostic output, such as that shown in Table 10. It is available on request.

In using the algorithm, it is important for documenting what has been done to have a method of input of the original Use and Matrix matrices that preserves the original version at the top of the input file and introduces the modifications as over-rides later in the file. It is also important to have software,

such as ViewMat, which will show corresponding columns of several large matrices side-by-side in a scrolling grid. ViewMat is also available on the Inforum Internet site.

```
#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), NewUse(5,5);
       Use.ReadA("Use.dat");
       Make.ReadA("Make.dat");
       purify(R,Use,Make,.000001);
       R.Display("This is R");
       writemat(R,"Recipe");
       NewUse = R*(~NewUse);
       writemat(NewUse,"NewUse");
       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] && sum > 0){
                            // 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",
                row,dismax,imax);
                }
        putinrow(P,R,row);
        }
return(OK);
}
```

References

Almon, C. (1970) Investment in input-output models and the treatment of secondary products, *Input-Output Techniques, vol. 2, Applications of Input-Output Analysis,* pp.103-116 (Amsterdam, North Holland Publishing Co.)

Almon, C., Buckler, M., Horwitz, L., and Reimbold, T.,(1974) *1985, Interindustry Forecasts of the American Economy* (Lexington, Lexington Books) pp.151-154.

European system of accounts: ESA 1995, Transmission programme of data. Eurostat.

Rainer, N. and Richter, J. (1992) Some Aspects of the Analytical Use of Descriptive Make and Absorption Tables. *Economic Systems Research,* 4, pp.159 - 172

Steenge, A.E. and Konijin. A new Approach to Irreducibility in Multisectoral Models with Joint Production. *Economic Systems Research,* 4, pp 125-132

The *System of National Accounts 1993* (published by the United Nations, the World Bank, the IMF, the OECD, and the European Union)

ten Raa, Thijs, D. Chakraborty, and J.A. Small (1984) An Alternative Treatment of Secondary Products in Input-Output Analysis, *Review of Economics and Statistics,* pp. 88-97.

ten Raa, Thijs (1988) An Alternative Treatment of Secondary Products in Input-Output Analysis:Frustration", *Review of Economics and Statistics,* pp. 535-538.

Van Rijckeghem (1967) "An Exact Mehod for Determining the TechnologyMatrix in a Situation with Secondary Products," *Review of Economics and Statistics,* pp. 607-8.