DAGG Guide

December 1995: Changes for the second edition of DAGG include 3 new commands: two species of RAS command and the AGGREG command which allows the user to specify that, for example, export demand elasticities for an aggregated data file are to be weighted averages of those on the parent file, the weights being the values of exports. These instructions have also been expanded.

April 1997: Added SCALE command and note on set labelling.

May 1998: Added MATMAP and MATAGG commands, allowed '*' as well as 'x' in dim expressions. Added lengthy worked example of aggregation

May 2000: Minor revisions.

May 2001: Minor revisions.

May 2018: Package now contains DAGG64 64-bit exe for very large problems.

DAGG is a freeware program which you can download from http://www.copsmodels.com/gpmark.htm

Contents

1. Introduction 3
- 1.1. DAGG commands in brief 3
- 1.2. DAGG inputs and outputs 4
- 1.3. Running DAGG 4
- 1.4. A typical DAGG run 4
- 1.5. Interpreting error messages 6
- 1.6. Up to eight times more fun! 6
- 1.7. Preparing DAGG input files efficiently 7

2. Disaggregating data: the DUPLICATE and SPLIT commands 7
- 2.1. DUPLICATE 7
- 2.2. SPLIT 7
- 2.3. New dimensions with DAGG 8
- 2.4. Caveats on the SPLIT command 8

3. Aggregating data: the REMAP and AGGREG commands 9
- 3.1. REMAP 9
- 3.2. AGGREG 10

4. Using mapping matrices: MATMAP and MATAGG 11
- 4.1. MATMAP 12
- 4.2. MATAGG 13

5. Miscellaneous commands 13
- 5.1. COLLAPSE 13
- 5.2. TRANSPOSE 13
- 5.3. DELETE 13
- 5.4. ADD 14
- 5.5. SCALE 14
- 5.6. COPY 14
- 5.7. STOP 14

6. The RAS command 14
- 6.1. An intuitive way to think of the RAS 16
- 6.2. Other uses for the multipliers 16
- 6.3. The RASPART command 17
- 6.4. Background on RAS 18

7. Set Element Names 18

8. Worked example of DAGG-regation 20

9. Worked example of DisAGGregation 25

What is DAGG ?

DAGG is a program that facilitates many of the most common tasks involved in preparing databases for input-output or general equilibrium models.

Early versions of the ORANI model predated the invention of GEMPACK. Solution routines were coded directly in FORTRAN. Because of this, it was almost impossible to change the model. Or more precisely, it was too dangerous to change the model because bugs would inevitably be introduced which might never be noticed.

This inflexibility meant that the structure of the ORANI database was very stable. A variety of specialized programs were developed to view and maintain the database. Two such programs were the AGG and DISAG programs developed by Sutton and Strzelecki at the Industry Commission in Australia. These were programs to change the numbers of industries and commodities in an ORANI database, either by aggregation or disaggregation.

The advent of GEMPACK made altering ORANI much easier. Versions proliferated, and not all databases had the same structure. This meant that specialized programs like AGG and DISAG, which relied on a particular data format, could not be used. However, the GEMPACK language was not, in those days, rich enough to support such operations as, for example, transforming a 108x108 matrix to size 50x50 using a 108 sector to 50 sector mapping.

The DAGG tool was designed to fill that gap. It requires that input and output data be contained in the "header array" (HAR) files which are the native file format of the GEMPACK system. Otherwise, it makes no assumptions about data structure. The name DAGG denotes the combination of the functions of DISAG and AGG in one program, but recalls also a slang term from Australian agriculture.

In line with its heritage, DAGG has an old-fashioned interface. Program input takes the form of short text files. Each line in the text file describes some operation that DAGG will perform on a particular matrix. These files are called "decks", because originally each line corresponded to a single punched card. The format of the instructions takes a little effort to learn, but is capable of describing a huge variety of operations very concisely. Most DAGG users would have a small library of decks, corresponding to particular tasks, which would be recycled and adapted to new jobs.

GEMPACK has now evolved to the point where a TABLO program could be used to perform most of the tasks which DAGG can do - although an equivalent TABLO program would often be many times larger than the DAGG deck it replaced. Programs AggHAR and ViewHAR are much handier than DAGG for most aggregation tasks. However, for some more complex tasks, particularly disaggregation, DAGG still preserves its place in the armory of the GEMPACK modeller.

Mark Horridge
Centre of Policy Studies,
Melbourne, Australia

DAGG Guide

1. Introduction

The DAGG program is designed to perform the most common operations involved in aggregating or disaggregating real matrices on header array data files. Typically these operations are:

- for disaggregating parameters, one row or column is *copied* to several new row/cols, all duplicates of the original. The DUPLICATE command performs this task.
- for disaggregating flows, one row or column is *split* into several row/cols, according to a weighting scheme. The new row/cols 'add up' to the original. The SPLIT command performs this task.
- for aggregating parameters, groups of rows (or columns) are *averaged* together (using given weights) so that each group becomes one row (or column). The AGGREG command performs this task.
- for aggregating flows, groups of rows (or columns) are *added* together so that each group becomes one row (or column). The REMAP command performs this task.
- adjusting data, using the RAS command.

DAGG can be run interactively but is designed for batch operation. DAGG assumes nothing about the structure of the database. It is flexible but primitive. DAGG only modifies real arrays; character and integer data are copied unchanged. DAGG makes full allowance for 7-dimensional matrices. DAGG is a FORTRAN program which can be compiled to run on any operating system.

MODHAR can perform some of the same tasks as DAGG—but MODHAR often demands greater user effort. However, MODHAR must still be used for some operations.

1.1. DAGG commands in brief

DUPLICATE	replace a given row or column by several identical row/cols. Used in disaggregating parameters.
SPLIT	split a given row or column into several new row/cols according to given weights. Used in disaggregating flow data.
REMAP	add together rows or columns. Used in aggregating flow data. Can also re-order rows or columns.
AGGREG	average together rows or columns, using supplied weights. Used in aggregating parameter data.
TRANSPOSE	swap any two dimensions of a matrix
SCALE	multiply all elements by a scalar
COLLAPSE	aggregate a matrix across one dimension
RAS	scale a 2-dimensional matrix to add to given row and column totals
RASPART	scale part of a 2-dimensional matrix to add to given row and column totals
ADD	add a new matrix
DELETE	delete a matrix
COPY	finish, retaining unmodified matrices unchanged
STOP	finish without retaining unmodified matrices

1.2. DAGG inputs and outputs

Every DAGG run uses three files:

- The OLD file contains the original data to be disaggregated. *DAGG does not modify this file.*
- The NEW file is created by DAGG. It contains modified matrices, and, probably, most of the matrices on the OLD file which were not modified.
- The SUP file contains supplementary data used in disaggregation. *DAGG does not modify this file.*

1.3. Running DAGG

DAGG does not support the official GEMPACK method of using stored input files, nor does it automatically create a log file. However, most operating systems will fill this gap. Under DOS, the command:

```
DAGG <DAGG.INP >DAGG.LOG
```

causes DAGG to take instructions from the text file DAGG.INP and to send screen output to the file DAGG.LOG. The file DAGG.INP might resemble the input file shown below. If an error occurs, it will be reported on the log file, not on the screen. Thus it is very important to check the log file, to see that the job ran properly.

1.4. A typical DAGG run

Input to a sample DAGG run might read as follows:

```
FID87.HAR
NEWFID.HAR
EXTRA.HAR
DUPLICATE P001  112x113   2  47  3
SPLIT     H007  112x113   2  47  3  EXT1  1x3
REMAP     H022  112x113   1  EXN3 64
COLLAPSE  H020  112x113x4 2
TRANSPOSE H021  112x113x4 1  2 ! 112*113*4 works too
DELETE    D007       ! Comments like this
ADD       Y092
COPY
```

Each DAGG instruction occupies one line. The ! character and any following characters are treated as comments (ie, ignored).

The first line of the input is the name of the OLD file. Next comes the name of the NEW file. Last comes the name of the SUP file. This is compulsory even if no supplementary data is needed; in this case, name any header array file.

After the three file names, commands can be issued in any order, except that COPY and STOP appear last.

`DUPLICATE P001 112x113 2 47 3` has the effect of:

- reading matrix P001 from the OLD file and checking that it is a real matrix with dimensions 112x113,
- expanding it in the second (2) dimension, ie, adding columns,
- replacing column 47 by 3 identical columns (2 more than previously),
- writing the expanded matrix to the NEW file with the same name and header,
- marking matrix P001 as 'modified'; no item can be modified twice in the same run.

`SPLIT H007 112x113 2 47 3 EXT1 1x3` has the effect of:

- reading matrix H007 from the OLD file and checking that it is of size 112x113,
- expanding it in the second (2) dimension, ie, adding columns,
- replacing column 47 by 3 new columns (2 more than previously),
- reading matrix EXT1 from the SUP file and checking that it is a real vector length 3,
- scaling the 3 elements of EXT1 so they add to one,

- using these 3 weights to split each element of the original column 47 into 3,
- writing the expanded matrix to the NEW file with the same name and header,
- marking matrix H007 as 'modified'; no item can be modified twice in the same run.

`REMAP H022 112x113 1 EXN3 64` has the effect of:

- reading matrix H022 from the OLD file and checking that it is of size 112x113,
- rearranging and combining the rows (dimension 1),
- using a integer mapping vector EXN3 on the SUP file; this must be of length 112,
- checking that each of the 112 elements of EXN3 is a number between 1 and 64,
- checking that each of the numbers 1 to 64 appears *somewhere* in EXN3,
- writing the remapped matrix ,size 64x113, to the NEW file with same name and header,
- marking matrix H022 as 'modified'.

More detail on the DUPLICATE, SPLIT and REMAP commands appears below.

`COLLAPSE H020 112x113x4 2` has the effect of:

- reading matrix H020 from the OLD file and checking that it is of size 112x113x4,
- adding it up across the second (2) dimension, to obtain a matrix size 112x4,
- writing the collapsed matrix to the NEW file with the same name and header,
- marking matrix H020 as 'modified'.

`TRANSPOSE H021 112x113x4 1 2` has the effect of:

- reading matrix H021 from the OLD file and checking that it is of size 112x113x4,
- swapping dimensions 1 and 2, to obtain a matrix size 113x112x4,
- writing the transposed matrix to the NEW file with the same name and header,
- marking matrix H021 as 'modified'.

`DELETE D007` merely has the effect of:

- marking a data item (real, integer or character) as 'modified'. This prevents it being copied to the NEW file later.

`ADD Y092` just:

- transfers a data item (real, integer or character) from the SUP to the NEW file. Any data item with the same header on the OLD file is marked as 'modified'. This prevents it being copied to the NEW file later.

`COPY`

- transfers all unmodified data items (real, integer or character) from the OLD to the NEW file. The program then terminates.

`STOP`

- terminates the session, without transferring unmodified data items.

1.5. Interpreting error messages

If any error occurs during a DAGG run, the program prints a message, then terminates without saving anything. Examine the log file to discover the cause of the error. For example, the following (faulty) command attempts to add up the 3rd dimension of a two dimensional matrix:

```
COLLAPSE ORGM 3x7 3
```

This would give rise to the following log file:

```
 Command was: COLLAPSE ORGM 3x7 3
(Read real array, size 3x7, header 'ORGM'.)
Collapsing OLD array:
ORGM: Wool Sales   with dimensions    3   7
 in dimension   3 above
Error: AggDim has value    3; value should be from    1 to    2
  (ERROR RETURN FROM ROUTINE: DAGG  )
  (E- integer out of permitted range)
 (The program terminated with an error.)
```

In order to interpret the message "AggDim has value 3; value should be from 1 to 2", one has to know what "AggDim" means. Find this out by consulting Section 5.1 below, which states that the COLLAPSE command has the format:

```
COLLAPSE  OldHead  OldDims  AggDim
```

In other words, "AggDim" refers to the final parameter of the command (the dimension over which to sum).

Sometimes DAGG may abort with the message that you must increase a certain parameter in the program file DAGG.FOR. For example, you may be asked to increase the parameter ‘mmreal’ – the number of elements in each of DAGG’s 3 work arrays. Then you have to recompile the program (LTG DAGG). This will only be possible if you have a source code licence for GEMPACK.

Each of DAGG’s 3 work arrays must be large enough to contain the largest matrix used in any single DAGG operation. If you increase ‘mmreal’ too much you may find that DAGG requires more RAM memory than your PC has.

1.6. Up to eight times more fun!

A typical use of DAGG might be to split industries and commodities as follows:

- Old commodity 16 to become 5 new commodities
- Old commodity 52 to become 3 new commodities
- Old industry 14 to become 5 new industries
- Old industry 50 to become 3 new industries

Typically, two data files would be treated, corresponding to flows and parameters.

Because DAGG can never modify a given matrix twice in the same session, up to 8 DAGG runs might be needed:

1 Split row 16 where necessary on flows file
2 Split row 52 where necessary on flows file
3 Split column 14 where necessary on flows file
4 Split column 50 where necessary on flows file

- Runs 5 to 8—repeat above for parameters file

The output file from each DAGG run is the input or OLD file for the next run. Several DAGG runs can be grouped in the same batch file. Such a file (under DOS) might look like this:

```
DAGG <DAGG1.INP >DAGG1.LOG
DAGG <DAGG2.INP >DAGG2.LOG
DAGG <DAGG3.INP >DAGG3.LOG
DAGG <DAGG4.INP >DAGG4.LOG
```

1.7. Preparing DAGG input files efficiently

Run SEEHAR on your original data file to obtain a listing of headers, which may be modified in a text editor to become a DAGG input file. Or, copy-and-paste the ViewHAR contents screen into Excel, delete a few columns, sort on the dimensions columns, and save as a text file, and edit that.

2. Disaggregating data: the DUPLICATE and SPLIT commands

The DUPLICATE and SPLIT commands are useful for disaggregating data. Use DUPLICATE for parameters such as demand elasticities, where you wish several new industries (say) to have the same value as one old industry. Use SPLIT for flows data, where you wish a value for one old industry to be *divided* among several new industries.

2.1. DUPLICATE

The format of the DUPLICATE command is:

```
DUPLICATE OldHead OldDims SplitDim SplitElm NumNew
```

For example:

```
DUPLICATE H023 1x114x113 3 47 5
```

`OldHead` identifies a real matrix on the OLD file.

`OldDims` are the dimensions of the matrix as displayed by GEMPACK programs. They must *loosely* match the dimensions of the matrix on the file. *Loosely* means that unitary dimensions may be ignored: `114x113`, `1x114x113`, and `114x1x113`, are all loose matches for one another. DAGG internally formats the matrix as specified in the DUPLICATE command, ie, in this case as `1x114x113`.

In all DAGG commands you can use '*' instead of 'x'

`SplitDim` shows in which dimension the split will occur. In the example, 3 means the third of the dimensions *as they were listed in the DUPLICATE command*, ie, the 113 or column dimension.

`SplitElm` shows which elements are to be duplicated. Here, column 47 in every row will be duplicated.

`NumNew` shows how many duplicates are made, in this case 5 (4 plus the original). This means that for all rows (the 114 dimension), columns 1 to 46 are unchanged. Columns 47-51 (5 of them!) in the new matrix are all the same as column 47 in the old matrix. New column 52 is old column 48, and so on till new column 117 is old column 113

The result is a `1x114x117` matrix. Whether the source matrix was of 2R or RL type, the new matrix is stored in RL format with unitary dimensions removed, eg, as a `114x117` RL matrix.

2.2. SPLIT

The SPLIT command operates in a very similar way to the DUPLICATE command, except that old elements are allocated between corresponding entries in the new matrix, instead of simply being copied to each of those entries. The format of the SPLIT command is:

```
SPLIT OldHead OldDims SplitDim SplitElm NumNew SupHead SupDims
```

for example:

```
SPLIT H023 114x114x113 3 47 5 EXT3 114x114x5
```

The `OldHead`, `OldDims`, `SplitDim`, `SplitElm` and `NumNew` parameters operate just as in the DUPLICATE command. The 2 additional parameters determine the weighting or allocation scheme:

`SupHead` identifies a real matrix on the SUP file.

`SupDims` gives dimensions for the above matrix which must (a) *loosely* match the dimensions of the matrix on the SUP file, and (b) satisfy the following conditions:

- the number of dimensions must equal that of `OldDims` (including uniitary dimensions).
- the `SplitDim`'th dimension must equal `NumNew`.
- each of the other dimensions must equal either 1 or the corresponding dimension in `OldDims`.

These conditions need explaining. Take our example:

```
SPLIT H023 114x114x113 3 47 5 EXT3 114x114x5
```

The dimensions of the weight matrix, 114x114x5, allow for the maximum amount of disaggregation detail. Normally however we have less data than this requires, so we assume that a common split factor is used along all rows, say. This effect would be achieved by:

```
SPLIT H023 114x114x113 3 47 5 EXT3 1x114x5
```

Conversely, a common split factor along all columns would be achieved by:

```
SPLIT H023 114x114x113 3 47 5 EXT3 114x1x5
```

Note the use of the unitary dimension in the weight dimension expression to distinguish these two cases. For a global split factor we would need:

```
SPLIT H023 114x114x113 3 47 5 EXT3 1x1x5
```

The weights stored on the SUP file need not add to 1. DAGG normalizes on the appropriate dimension as required.

2.3. New dimensions with DAGG

Consider the following command:

```
DUPLICATE H025 1x114 1 1 5
```

The input is a vector length `114`. DAGG duplicates this in the first (dummy) dimension, so that the output is a matrix of size `5x114`. Conversely:

```
DUPLICATE H025 114x1 2 1 5
```

prouduces a matrix of size `114x5` from the input vector. SPLIT can be used in the same way:

```
SPLIT H025 1x114 1 1 5 EXT3 5x1
```

splits each of the original `114` elements into `5` parts, producing a matrix of size `5x114`.

2.4. Caveats on the SPLIT command

A When SPLIT divides one old flow into `NumNew` new flows, the new flows sum to the old. This helps to preserve database balance—the equality of supply and demand for each commodity. However, nothing guarantees this equality for each of the individual industries and commodities formed by a SPLIT. Suppose that 2 DAGG runs were used to split 'Cereals' rows and columns into 'Wheat' and 'Barley' for a number of different matrices and that different weights were used to split different matrices. Even if originally costs equalled sales for 'Cereals', the same will not necessarily be true for 'Wheat' and 'Barley' individually, although the *total* pure profits for the two will be zero. It is up to the user to ensure that individual balance is achieved for the new industries in such a case.

B Some matrices are diagonal or have a diagonal tendency—intermediate demands, or the MAKE matrix are examples. Again suppose that 2 DAGG runs were used to split the 'Cereals' rows and columns of the MAKE matrix. into 'Wheat' and 'Barley', both times using a 1:1 split ratio. The Cereal-Cereal diagonal element would now become a square, *with equal values in all 4 corners*. If this is not wanted, MODHAR must be used to correct matters.

C Suppose you want to split several rows, requiring a separate DAGG run for each split. Life will be easier if you *split the last rows first*. That way, the number of the row you are splitting will always be the original number of that row.

3. Aggregating data: the REMAP and AGGREG commands

3.1. REMAP

The REMAP command is used for aggregating flows[1]. Groups of rows (or columns) are added together so that each group becomes one row (or column) of the new matrix. The format of the REMAP command is:

```
REMAP OldHead OldDims MapDim SupHead NewSiz
```

For example:

```
REMAP  H022  3x4  2  EXN3 3
```

The `OldHead`, and `OldDims` parameters operate just as in other commands. The 3 additional parameters determine how the matrix is remapped:

`MapDim` shows which dimension is to be remapped—here the 2nd (ie, columns).

`SupHead` must identify an integer vector on the SUP file with length equal to the `MapDim`'th component of `OldDims`.

`NewSiz` gives the new size for the `MapDim`'th dimension. This cannot be greater than the old size.

Suppose that in our example the `H022` matrix has values:

1.0	2.0	3.0	4.0
5.0	6.0	7.0	8.0
9.0	10.0	11.0	12.0

and the integer vector which will remap columns (dimension 2) has values:

1	2	3	1

This means that: old column 1 goes to new column 1
old column 2 goes to new column 2
old column 3 goes to new column 3
old column 4 goes to new column 1

So new column 1 is the sum of old columns 1 and 4, giving as output:

5.0	2.0	3.0
13.0	6.0	7.0
21.0	10.0	11.0

Make sure that every new column contains at least one old column, or an error will occur. If `NewSiz` is *equal* to the `MapDim`'th component of `OldDims`, REMAP can be used to reorder rows or columns without combining them.

[1] GEMPACK programs ViewHAR and AggHAR now do this job far more conveniently. See ViewHAR online help.

3.2. AGGREG

The AGGREG command is used to aggregate parameters[2]. Groups of rows (or columns) are *averaged* together (using given weights) so that each group becomes one row (or column) of the new matrix. Apart from the need to specify the weighting scheme, AGGREG works much as REMAP. The format of the AGGREG command is:

```
AGGREG OldHead OldDims MapDim SupHead NewSiz WgtHead WgtDims
```

For example:

```
AGGREG  H022  3x4  2  EXN3 3 WGHT 1x4
```

The `OldHead`, and `OldDims` parameters operate just as in other commands. The remaining parameters determine how the matrix is aggregated:

`MapDim`, `SupHead,` and `NewSiz` work just as in the `REMAP` command.

`WgtHead` identifies a real matrix on the SUP file, used for weights.These weights need not add to 1. DAGG normalizes them on the `MapDim`'th dimension automatically.

`WgtDims` gives dimensions for the above matrix which must (a) *loosely* match the dimensions of the matrix at `WgtHead`, and (b) satisfy the following conditions:

- the number of dimensions must equal that of `OldDims` (including unitary dimensions).
- the Map`Dim`'th dimension must equal the corresponding dimension in `OldDims`.
- each of the other dimensions must equal either 1 or the corresponding dimension in `OldDims`.

Suppose that in our example the `H022` matrix and EXN3 vector had the same values as in the example for the REMAP command. Suppose that the weight vector had values:

3.0 2.0 4.0 1.0

This means that: 3/4 (= 3/(3+1)) of old column 1 goes to new column 1
all of old column 2 goes to new column 2
all of old column 3 goes to new column 3
1/4 (= 1/(3+1)) of old column 4 goes to new column 1

So new column 1 is an *average* of old columns 1 and 4, giving as output:

1.75 2.0 3.0
5.75 6.0 7.0
9.75 10.0 11.0

Make sure that for each group, the weights do not add to zero, or an error will occur.

[2] GEMPACK program AggHAR now does this job far more conveniently. See ViewHAR online help.

4. Using mapping matrices: MATMAP and MATAGG

REMAP and AGGREG use an integer mapping vector to aggregate flows or parameters. With this method, each old element is mapped to just one new element (although each new element may be the sum (or average) of several old elements).

MATMAP and MATAGG use an alternative way to specify mappings, mapping matrices. Consider the following scheme to aggregate imports of 3 old commodities to 2 new commodities:

Old commodity 1 goes 80% to new commodity A, 20% to new commodity B

Old commodity 2 goes 100% to new commodity B

Old commodity 3 goes 30% to new commodity A, 70% to new commodity B

We could represent this scheme by a matrix mapping old coms 1-3 to new coms A-B thus:

Import Mapping	New com A	New com B	Share Total
Old com 1	0.8	0.2	1
Old com 2	1		1
Old com 3	0.3	0.7	1

The above mapping could not be achieved using the REMAP command. On the other hand, any mapping possible with REMAP can be represented as a mapping matrix which contains just one entry per row.

A mapping matrix may also contain enough information for many SPLIT operations. As long as the supplementary splitting shares are vectors rather than matrices, we can represent them in a mapping matrix. Consider the following scheme to distribute wages for 4 old industries 1-4 to 6 new industries.

Wages Mapping	New ind A	New ind B	New ind C	New ind D	New ind E	New ind F	Share Total
Old ind 1	0.8	0.2					1
Old ind 2			1				1
Old ind 3				1			1
Old ind 4					0.7	0.3	1

We could achieve the same effect by 2 SPLIT commands, first splitting row 4 70/30, then splitting row 1 80/20. As long as the mapping matrix has just one entry per column, it is equivalent to a series of SPLIT commands. Now suppose we amended the wage mapping scheme as follows:

Amended Wages Mapping	New ind A	New ind B	New ind C	New ind D	New ind E	New ind F	Share Total
Old ind 1	0.8	0.2					1
Old ind 2			0.6	0.4			1
Old ind 3			0.9	0.1			1
Old ind 4					0.7	0.3	1

Neither SPLIT nor REMAP could be used to specify the way that old inds 2 and 3 are distributed over new inds C and D.

DAGG provides two commands for using mapping matrices: MATMAP (for flows) and MATAGG (for parameters). Use MATMAP to specify mappings which are not possible using REMAP or SPLIT, or use it to perform a number of simple SPLITs simultaneously. Similarly, use

MATAGG to specify mappings which are not possible using AGGREG or DUPLICATE, or use it to perform a number of DUPLICATEs simultaneously.

4.1. MATMAP

The MATMAP command is used for aggregating or disaggregating flows. Each old row (or column) is distributed amongst various rows (or columns) of the new matrix. The format of the MATMAP command is:

```
MATMAP OldHead OldDims MapDim SupHead NewSiz
```

For example:

```
MATMAP  H022  3x4  2  M423 3
```

The `OldHead`, and `OldDims` parameters operate just as in other commands. The 3 additional parameters determine how the matrix is remapped:

`MapDim` shows which dimension is to be remapped—here the 2nd (ie, columns).

`SupHead` must identify a real mapping matrix on the SUP file with as many rows as the `MapDim`'th component of `OldDims` and with `NewSiz` columns.

`NewSiz` gives the new size for the `MapDim`'th dimension. This could be greater, less, or equal to the old size.

Suppose that in our example the `H022` matrix has values:

1.0	2.0	3.0	4.0
5.0	6.0	7.0	8.0
9.0	10.0	11.0	12.0

and the mapping matrix M423 (maps 4 to 3) which will remap columns (dimension 2) has values:

1.0	0.0	0.0
0.2	0.8	0.0
0.0	0.3	0.7
0.0	0.0	1.0

This means that: old column 1 goes to new column 1
old column 2 goes 20% to new column 1, 80% to new column 2
old column 3 goes 30% to new column 2, 70% to new column 3
old column 4 goes to new column 3

Or, if you like: new column 1 is all of old column 1 + 20% of old column 2
new column 2 is 80% of old column 2 + 30% of old column 3
new column 3 is 70% of old column 3 + all of old column 4

The output matrix has only 3 columns and has the same row sums as the original:

1.40	2.50	6.10
6.20	6.90	12.90
11.00	11.30	19.70

In the examples above, the row sums of the mapping matrix are one, and this certainly is an aid to clarity. It is not however essential, since DAGG converts the mapping matrix to row shares automatically. But you must make sure that every row in the mapping matrix contains at least one entry, and that the original (unscaled) row sums are greater than zero. Individual negative elements are allowed, although DAGG warns of these.

4.2. MATAGG

The MATAGG command is used for aggregating or disaggregating parameters. Each old row (or column) is averaged amongst various rows (or columns) of the new matrix. The format of the MATAGG command is:

```
MATAGG OldHead OldDims MapDim SupHead NewSiz WgtHead
```

For example:

```
MATMAP  P027  4  1  M423 3 IMPS
```

The parameters are the same as for MATMAP except for the additional, last, header WgtHead. This must refer to a real vector matrix of weights of the same length as the `MapDim`'th component of `OldDims` (4 in this case).

5. Miscellaneous commands

5.1. COLLAPSE

This command aggregates a matrix over one dimension. The format is:

```
COLLAPSE  OldHead  OldDims  AggDim
```

The `OldHead`, and `OldDims` parameters operate just as in other commands. `AggDim` is an integer which shows which dimension is to be aggregated. For example, the command:

```
COLLAPSE  H020  112x113x4  2
```

has the effect of:

- reading matrix H020 from the OLD file and checking that it is of size 112x113x4,
- adding it up across the second (2) dimension, to obtain a matrix size 112x4,
- writing the collapsed matrix to the NEW file with the same name and header,
- marking matrix H020 as 'modified'.

5.2. TRANSPOSE

This command transposes two dimensions of a matrix. The format is:

```
TRANSPOSE  OldHead  OldDims  TranDim1 TranDim2
```

The `OldHead`, and `OldDims` parameters operate just as in other commands. `TranDim1` and `TranDim2` are integers which show which dimensions are to be transposed. For example, the command:

```
TRANSPOSE  H021  112x113x4  1  2
```

has the effect of:

- reading matrix H021 from the OLD file and checking that it is of size 112x113x4,
- swapping dimensions 1 and 2, to obtain a matrix size 113x112x4,
- writing the transposed matrix to the NEW file with the same name and header,
- marking matrix H021 as 'modified'.

5.3. DELETE

This command 'deletes' a data item (real, integer or character). The format is:

```
DELETE  OldHead
```

The `OldHead` parameter identifies a matrix which will be exempted from the action of a later `COPY` command. For example, the command:

```
DELETE  D007
```

merely has the effect of marking the data item with header `D007` as 'modified'. This prevents it being copied to the NEW file later.

5.4. ADD

The command

```
ADD  OldHead
```

just transfers a data item (real, integer or character) from the SUP to the NEW file. Any data item with the same header on the OLD file is marked as 'modified'. This prevents it being copied to the NEW file later.

5.5. SCALE

The command

```
SCALE  OldHead  Numer  Denom
```

has the effect of:

- reading matrix header OldHead from the OLD file,
- multiplying every element of the matrix by the ratio Numer/Denom. Note that Numer and Denom must be integers.
- writing the scaled matrix to the NEW file with the same name and header,
- marking matrix OldHead as 'modified'.

5.6. COPY

The command

```
COPY
```

transfers all unmodified data items (real, integer or character) from the OLD to the NEW file. The program then terminates.

5.7. STOP

The command

```
STOP
```

terminates the session, without transferring unmodified data items.

6. The RAS command

In preparing input-output data, the need often arises to adjust some matrix A so that it sums to given row and column totals. The RAS method meets this need. Given an original matrix A(i,j), size r×c, and target vectors of row totals R(i), size r, and column totals C(j), size c, the RAS attempts to find a new, similar, matrix B(i,j) such that:

$$\sum_i B(i,j) = C(j) \quad j = 1,...,c$$

$$\sum_j B(i,j) = R(i) \quad i = 1,...,r$$

The new matrix B(i,j) is related to the original A(i,j) via:

$$B(i,j) = rm(i).cm(j).A(i,j) \qquad i = 1,...,r \qquad j = 1,...,c$$

where rm(i) is a vector of row multipliers and cm(j) is a vector of column multipliers.

The RAS may be appropriately used to eliminate small inconsistencies that have arisen during data manipulation, or that may be traced to the use of data from several, mutually inconsistent, sources. Unfortunately, it will also smooth away gross errors due to blunders by the practitioner. Hence, it is vitally important to ensure that the RAS is performing small necessary

adjustments to the data, rather than hiding the evidence of human carelessness. For this reason DAGG prints out an extensive report of the RAS process. *This report should be carefully checked*.

The format of the RAS command is:

```
RAS OldHead OldDims RowHead Nrows ColHead NCols
```

For example:

```
RAS BAS1 73x40 RTGT 73 CTGT 40
```

`OldHead` identifies the real matrix on the OLD file which will be RASsed.

`OldDims` are the dimensions of the matrix at `OldHead`.

`RowHead` identifies the row target vector, which must be on the SUP file.
`NRow` is the length of the row target vector; it must be equal to the number of rows in the matrix.
`ColHead` identifies the column target vector, which must be on the SUP file.
`NCol` is the length of the column target vector; it must be equal to the number of columns in the matrix.

Three items are written to the output file:

`OldHead` contains the adjusted matrix.

`RowHead` contains the vector of row multipliers.

`ColHead` contains the vector of column multipliers.

The RAS follows an iterative procedure. Initially all elements of the row multipliers rm(i) and the column multipliers cm(j) are set to one. Then the following two steps are repeated up to 20 times.

(a) Using the current row multipliers, new column multipliers are chosen so that:

$$\sum_i B(i,j) = C(j) \qquad j = 1,...,c$$

i.e.: $$\sum_i rm(i).cm(j).A(i,j) = C(j) \qquad j = 1,...,c$$

i.e.: $$cm(j) = C(j) / \sum_i rm(i).A(i,j) \qquad j = 1,...,c$$

(b) Using the current column multipliers, new row multipliers are chosen so that:

$$\sum_j B(i,j) = R(i) \qquad i = 1,...,r$$

i.e.: $$\sum_j rm(i).cm(j).A(i,j) = R(i) \qquad i = 1,...,r$$

i.e.: $$rm(i) = R(i) / \sum_j cm(j).A(i,j) \qquad i = 1,...,r$$

Each of steps (a) and (b) disturbs the equality brought about by the previous step. However, successive adjustments to the row and column multipliers should become much smaller at every step. Usually, 5 iterations of (a) and (b) are sufficient to compute rm(i) and cm(j) (and thus B(i,j)) to a high degree of accuracy.

Failure of the RAS to converge within 5 or 10 iterations normally means that the procedure has been misused. Non-convergence can arise from four causes:

(a) The sum of the target row totals is not equal to the sum of the target column totals. To avoid this problem the DAGG RAS procedure starts by computing both sums and the average of the two. Both target vectors are then uniformly scaled so that their sum is equal to this average.

(b) Some of the target row and column totals are vastly different from the row and column totals of the original matrix. To signal this problem, the RAS report prints out:

- the original row and column totals

- the target row and column totals
- the ratios of original row and column totals to the target—hopefully between 0.2 and 5.

(c) Some of the elements of the original matrix are negative. This can sometimes prevent convergence, unless the negative element has an absolute value which is minute compared to other members of the same row and column. The RAS report prints out a list of all negative elements of the original matrix which have absolute value greater than 5% of either of the corresponding row or column totals.

(d) Singletons—elements which are the sole non-zero entry in both their row and their column—force the RAS to scale a single number so that it equals both row and column targets. If the two targets differ at all, this will be impossible. Similarly, elements which are nearly as big as both their row and their column totals cause the RAS to perform poorly. To signal this problem the RAS report prints out a list of all elements in the original matrix which have absolute value greater than 50% of either of the corresponding row or column totals. There are two main remedies for the singleton problem:

- ensure target row and column totals coresponding to singletons are the same.
- avoid singletons by 'smearing' the original matrix. This could be done by merely adding one to each element of the original matrix. For a more sophisticated approach, form the default apportionment matrix:

  $$D(i,j) = R(i).C(j)/T \quad \text{where } T = \sum_j C(j) = \sum_i R(i)$$

  This is the matrix that would arise from RASsing a matrix filled with ones. Replace the original matrix A with an average of A and D: e.g., $A(i,j) \Leftarrow (A(i,j) + D(i,j))/2$.

6.1. An intuitive way to think of the RAS

The RAS procedure is often explained (and implemented) in another, perhaps more intuitive, way. Again starting with an initial matrix and given row and column targets, the following procedure is repeated:

Repeat:

- Scale each row of the matrix so that it adds up to the corresponding row target.
- Scale each column of the matrix so that it adds up to the corresponding column target.

until both rows and columns add up to the targets.

Supposing that there were no error, and imagining that computers were perfectly accurate, the above procedure would give rise to just the same scaled matrix as does the procedure used by DAGG. Nevertheless, the DAGG procedure is preferable, because it provides, in the form of the row and column multipliers, an accurate account of how cruelly the original data had to be tortured in order to meet the targets. Remember, every RAS poses the modeller with one of these two questions:

- Why didn't my RAS converge?
- My RAS converged, but have I done violence to the data?

DAGG attempts to provide sufficient diagnostics to answer either of these questions. Simpler routines, which can be easily implemented on a spreadsheet, usually provide insufficient diagnostics.

6.2. Other uses for the multipliers

Apart from their diagnostic value, the multipliers have further uses. They can be used to scale groups of matrices. For example, we may wish to scale a basic values matrix and a tax matrix so that the sum of these two matrices—the producer price matrix—has given row and column totals.

To do this, we would form the producer price matrix by adding the basic values and tax matrices together; then RAS the producer price matrix. The row and column multipliers would then be used to scale the basic values and tax matrices.

6.3. The RASPART command

Sometimes we only want to RAS part of a matrix. For example, in ensuring that costs = sales for each industry we might prepare a matrix resembling the following:

	Intermediate h columns	Final Demand 4 columns	Row Totals = Sales	
Domestic g rows	A			Row Targets g rows
Imported g rows				
Capital g rows				
Labour g rows				
Col Totals = Costs				
	Col Targets h columns			

Suppose we have g row targets for sales and h column targets for costs—these might be taken from the row and column sums of the MAKE matrix. We wish to scale the first g rows of the matrix to sum to the g sales targets, and to scale the first h columns of the matrix to sum to the h cost targets. Only the intermediate domestic demand will be scaled to meet both targets. Wages, for example, will be scaled only by cost multipliers; domestic final demands only by sales multipliers; and imported final demands will not be scaled at all.

To meet this need, the RASPART command allows the user to scale only the *first* a rows and b columns of a matrix size r×c. The values of a and b are assumed to be the sizes of the row and column target vectors.

The format of the RASPART command is the same as the RAS command:

```
RASPART OldHead OldDims RowHead Nrows ColHead NCols
```

For example:

```
RASPART BAS1 73x40 RTGT 70 CTGT 35
```

`OldHead` identifies the real matrix on the OLD file which will be RASsed.

`OldDims` are the dimensions of the matrix at `OldHead`.

`RowHead` identifies the row target vector, which must be on the SUP file.
`NRow` is the length of the row target vector; it must be equal to *or less than* the number of rows in the matrix.
`ColHead` identifies the column target vector, which must be on the SUP file.
`NCol` is the length of the column target vector; it must be equal to *or less than* the number of columns in the matrix.

Three items are written to the output file:

`OldHead` contains the adjusted matrix.

`RowHead` contains the vector of row multipliers.

`ColHead` contains the vector of column multipliers.

If both row targets and column targets are the same length as the corresponding matrix dimensions, an error will occur. Use the the RAS command if this is what you want to do.

Apart from the length of the target vectors, the RASPART command works just as the RAS command. The only other difference is that the two target vectors need not have the same sum. Consequently, the target vectors are not scaled.

6.4. Background on RAS

Historically, the RAS has been used to 'update' input-output databases, when fresher data are available which, however, are less detailed than the original. For example, we might have a full database for 1990 and only partial data (such as column and row sums) for 1995. RAS can be used to produce a synthetic database for 1995.

Eventually, full data for 1995 becomes available. The question then arises: how well does the RAS predict actual changes in input-output coefficients? This question is addressed by the book:

Biproportional Matrices and Input-Output Change
Michael Bacharach
Cambridge University Press 1970

which also includes much more information about the RAS. Bacharach discusses under which conditions the RAS converges. Also, he shows how the RAS formulae could be deduced from a statistical minimization problem. The obvious minimization criterion:

$$S = \sum_i \sum_j A(i,j) \left([B(i,j)-A(i,j)]/A(i,j)\right)^2$$

i.e., the weighted sum of proportional deviations, allows cells to become negative, and so is unsatisfactory. Instead the following minimand, derived from Theil's entropy concept, gives rise to the RAS transformation:

$$S = \sum_i \sum_j B(i,j) \log\{\frac{B(i,j)}{A(i,j)}\}$$

Notice that the weights B(i,j) are from the final matrix.

Another interesting paper is *Entropy Theory and RAS are Friends* by Robert McDougall, a GTAP Working paper which you can download from

http://www.agecon.purdue.edu/gtap/wkpapr/index.htm

7. Set Element Names

GEMPACK usually stores set element names with each real matrix in Header Array Files. Header Array Files created by a TABLO-generated program normally contain such labelling data. DAGG does not preserve such labels (they were not invented when DAGG was developed). However

- GEMPACK programs never require the set labels: they are optional.
- For model data files you can restore the labels as follows. Run a simulation which creates an updated data file, using as input the unlabelled file produced by DAGG. Have a shock of zero value. The output updated data should be numerically identical to the input file (check this with CMPHAR) but will have the set element labels. (NB: headers not used by the model are removed by this process).
- ViewHAR can be used to apply set element names to arrays.
- TABLO-generated programs created by recent versions of GEMPACK may insist that set element names stored with each header on a file exactly match the set elements listed in the TAB file. This is a safety feature -- sometimes useful, sometimes very annoying. You can switch it off by including the line:

```
check-on-read elements = warn;
```

in your CMF file.

8. Worked example of DAGG-regation

In this section we go through the steps involved in aggregating an entire model data base[3]. The original data was for a Taiwanese version of the ORANIG model which had 170 commodities, 160 industries, 8 margins and 6 occupations. The aggregated data base is to have 39 commodities and industries, 2 margin commodities and 1 occupation. We start by preparing a text file (TEMPLATE.DAG) listing the real data items as follows:

```
1ARM    COM               !*SIGMA1: Intermediate Armington
2ARM    COM               !*SIGMA2: Investment Armington
3ARM    COM               !*SIGMA3: Households Armington
P018    COM               !*EXP_ELAST: Trad Export Elasticities
XPEL    COM               !*EPS: Expenditure Elasticities
P027    IND               !*QCOEF: Gross/Net Rate of Return
P028    IND               !*SIGMA1PRIM: Primary Factor Sigma
SCET    IND               !*SIGMA1OUT: Output Sigma
SLAB    IND               !*SIGMA1LAB: Labour Sigma

0TAR    COM               ! V0TAR: Tariff Revenue
4BAS    COM               ! V4BAS: Exports Basic
4TAX    COM               ! V4TAX: Exports Tax
MAKE    COM*IND           ! MAKE: Multiproduct Matrix
4MAR    COM*MAR           ! V4MAR: Exports Margins
3BAS    COM*SRC           ! V3BAS: Households Basic
3TAX    COM*SRC           ! V3TAX: Households Tax
5BAS    COM*SRC           ! V5BAS: Government Basic
5TAX    COM*SRC           ! V5TAX: Government Tax
6BAS    COM*SRC           ! V6BAS: Inventory Changes
1BAS    COM*SRC*IND       ! V1BAS: Intermediate Basic
1TAX    COM*SRC*IND       ! V1TAX: Intermediate Tax
2BAS    COM*SRC*IND       ! V2BAS: Investment Basic
2TAX    COM*SRC*IND       ! V2TAX: Investment Tax
1MAR    COM*SRC*IND*MAR   ! V1MAR: Intermediate Margins
2MAR    COM*SRC*IND*MAR   ! V2MAR: Investment Margins
3MAR    COM*SRC*MAR       ! V3MAR: Households Margins
5MAR    COM*SRC*MAR       ! V5MAR: Government Margins
1CAP    IND               ! V1CAP: Capital
1LND    IND               ! V1LND: Land
1OCT    IND               ! V1OCT: Other Costs
1LAB    IND*OCC           ! V1LAB: Labour
```

Text File: TEMPLATE.DAG

The above file was produced by opening the original database in ViewHAR and pasting the contents page to Excel. Some editing was done there before saving the sheet to a text (CSV) file. A text editor was then used to produce the result shown above. Although not essential, it adds clarity to arrange the rows in a certain order:

- Parameters, which must be averaged, are separated from flows, which must be added.
- Both flows and parameter have been sorted (in Excel) according to dimension. Scalar items don't need to be aggregated and so need not appear in the file.

Such a template can be reused as the starting point of many DAGG operations.

Since we plan to aggregate in COM, IND and MAR dimensions, which are used simultaneously by headers 1MAR and 2MAR and since each DAGG job can only work on one dimension at a time, there will have to be 3 DAGG jobs. We decide to arrange them as follows:

1 job DGIND which aggregates the IND dimension;
2 job DGCOM which aggregates the COM dimension; and
3 job DGMAR which aggregates the MAR and OCC dimensions.

MAR and OCC can be done together because no header uses both dimensions.

---

[3] GEMPACK program AggHAR now does this job far more conveniently. See ViewHAR online help. Thus the exercise is mainly of historical interest. For **dis**aggregation, there is still no obvious substitute to DAGG.

When aggregating, it does not normally matter in which order the dimensions are treated. When **dis**aggregating, it normally does matter, and you should think ahead carefully. A good scheme is to first split the dimension where most data is available. Often, we have detailed sales data which are used to split the COM dimension. The totals of the split sales then provide weights for a fairly simple split in the IND (costs)dimension.

The next step is to prepare the SUPP file of supplementary data used by all 3 DAGG jobs. This will contain:

- an integer header IMAP, consisting of 160 integers between 1 and 39 showing to which of the 39 new industries each of the 160 old industries belongs.
- an integer header CMAP, consisting of 170 integers between 1 and 39 showing to which of the 39 new commodities each of the 170 old commodities belongs.
- an integer header MMAP, consisting of 8 integers between 1 and 2 showing to which of the 2 new margin commodities each of the 8 old margin commodities belongs. This needs elaboration: First, every new commodity to which any old margin commodity is mapped will need to be a new margin commodity. Thus the mapping in CMAP determines the number of new margin commodities. Second, MMAP tells, for example, that the 3rd original margin (which might have been old commodity 100) maps to the 1st new margin (which might be new commodity 20). So MMAP(3) = 1. Neither “100” nor “20” appear in the MMAP vector.

ViewHAR can be used to create 3 integer vectors of the appropriate sizes. Paste numbers from Excel into the three new headers. Save the result as DAGSUP.HAR.

In addition to the 3 integer mapping vectors, we need real vectors of weights to aggregate the elasticities which are listed in the left hand columns below . For the ORANIG model, this need has been anticipated, and each time the model is run a summary file is produced containing the weight vectors listed in the right hand columns below. We can simply use ViewHAR to import these headers from the summary file into DAGSUP.HAR.

Elasticities			Weights		
Head	Size	Description	Head	Size	Description
1ARM	COM	Armington intermed	1PUR	COM	Interm.Usage by com at PP
2ARM	COM	Armington investment	2PUR	COM	Invest.Usage by com at PP
3ARM	COM	Armington households	3PUR	COM	Consumption at Purch.Prices
P018	COM	Export Elas	4PUR	COM	Exports at Purchasers Prices
P027	IND	QCoeff	1CAP	IND	Capital Rentals
P028	IND	CES prim factors	VLAD	IND	Industry Factor Cost
SCET	IND	output sigma	1TOT	IND	Industry Output
SLAB	IND	CES for lab skill	LAB1	IND	Industry Wages
XPEL	IND	Hhold Expend Elas	3PUR	COM	Consumption at Purch.Prices

In general there are no rigid rules governing the choice of weights. For example, some people might have used import weights (rather than dom+imp usage as is suggested above) to aggregate the Armington elasticities[4]. On the other hand, where elasticities must satisfy a condition, weights must be carefully chosen. For example, the expenditure-share-weighted average of the expenditure elasticities should equal 1. The use of expenditure weights (3PUR) to aggregate these elasticities ensures that this adding-up property will be preserved in the aggregate database (the proof of this is left as an exercise).

Now that DAGSUP.HAR contains the weights and integer mapping vectors that we will need, it is time to prepare the input files for each of the three DAGG jobs. From the DOS command line, type:

```
find "IND" template.dag >dgind.inp
```

This has the effect of creating file DGIND.IMP, consisting of all lines in TEMPLATE.DAG that mention “IND”. Similarly extract all lines containing COM and MAR:

```
find "COM" template.dag >dgcom.inp
```

[4] Even better might be the square root of (import usage x domestic usage). Can you see why?

```
find "MAR" template.dag >dgmar.inp
```

We have now produced the three files listed below.

DGIND.INP:

```
P027    IND              !*QCOEF: Gross/Net Rate of Return
P028    IND              !*SIGMA1PRIM: Primary Factor Sigma
SCET    IND              !*SIGMA1OUT: Output Sigma
SLAB    IND              !*SIGMA1LAB: Labour Sigma
MAKE    COM*IND          ! MAKE: Multiproduct Matrix
1BAS    COM*SRC*IND      ! V1BAS: Intermediate Basic
1TAX    COM*SRC*IND      ! V1TAX: Intermediate Tax
2BAS    COM*SRC*IND      ! V2BAS: Investment Basic
2TAX    COM*SRC*IND      ! V2TAX: Investment Tax
1MAR    COM*SRC*IND*MAR  ! V1MAR: Intermediate Margins
2MAR    COM*SRC*IND*MAR  ! V2MAR: Investment Margins
1CAP    IND              ! V1CAP: Capital
1LND    IND              ! V1LND: Land
1OCT    IND              ! V1OCT: Other Costs
1LAB    IND*OCC          ! V1LAB: Labour
```

DGCOM.INP:

```
1ARM    COM              !*SIGMA1: Intermediate Armington
2ARM    COM              !*SIGMA2: Investment Armington
3ARM    COM              !*SIGMA3: Households Armington
P018    COM              !*EXP_ELAST: Trad Export Elasticities
XPEL    COM              !*EPS: Expenditure Elasticities
0TAR    COM              ! V0TAR: Tariff Revenue
4BAS    COM              ! V4BAS: Exports Basic
4TAX    COM              ! V4TAX: Exports Tax
MAKE    COM*IND          ! MAKE: Multiproduct Matrix
4MAR    COM*MAR          ! V4MAR: Exports Margins
3BAS    COM*SRC          ! V3BAS: Households Basic
3TAX    COM*SRC          ! V3TAX: Households Tax
5BAS    COM*SRC          ! V5BAS: Government Basic
5TAX    COM*SRC          ! V5TAX: Government Tax
6BAS    COM*SRC          ! V6BAS: Inventory Changes
1BAS    COM*SRC*IND      ! V1BAS: Intermediate Basic
1TAX    COM*SRC*IND      ! V1TAX: Intermediate Tax
2BAS    COM*SRC*IND      ! V2BAS: Investment Basic
2TAX    COM*SRC*IND      ! V2TAX: Investment Tax
1MAR    COM*SRC*IND*MAR  ! V1MAR: Intermediate Margins
2MAR    COM*SRC*IND*MAR  ! V2MAR: Investment Margins
3MAR    COM*SRC*MAR      ! V3MAR: Households Margins
5MAR    COM*SRC*MAR      ! V5MAR: Government Margins
```

DGMAR.INP:

```
4MAR    COM*MAR          ! V4MAR: Exports Margins
1MAR    COM*SRC*IND*MAR  ! V1MAR: Intermediate Margins
2MAR    COM*SRC*IND*MAR  ! V2MAR: Investment Margins
3MAR    COM*SRC*MAR      ! V3MAR: Households Margins
5MAR    COM*SRC*MAR      ! V5MAR: Government Margins
```

A little work with a text editor uses the above files to produce the 3 final input "decks" shown on the next page. Recalling that the first 3 lines of each deck name the OLD, NEW, and SUPP files, we see that the information flows through the three jobs thus:
`taiwn170.har` →Job1:IND→ `temp1.har` →Job2:COM→ `temp2.har` →Job3:MAR→ `agg39.har`

Search-and-replace was used to change COM, IND and MAR to the appropriate sizes. Of course, the sizes change as the jobs progress!

	IND	COM	MAR
DGIND.INP	160	170	8
DGCOM.INP	39	170	8
DGMAR.INP	39	39	8

REMAP and AGGREG commands are used everywhere with the exception of the COLLAPSE command in DGMAR.INP, used to eliminate the OCC dimension.

DGIND.INP:

```
taiwn170.har
temp1.har
dagsup.har
AGGREG P027 160     1 IMAP 39 1CAP 160 ! QCoeff
AGGREG P028 160     1 IMAP 39 VLAD 160 ! CES prim factors
AGGREG SCET 160     1 IMAP 39 1TOT 160 ! output sigma
AGGREG SLAB 160     1 IMAP 39 LAB1 160 ! CES for lab skill
REMAP  MAKE 170x160       2 IMAP 39  ! MAKE
REMAP  1BAS 170x2x160     3 IMAP 39  ! Intermediate Basic
REMAP  1TAX 170x2x160     3 IMAP 39  ! Intermediate Tax
REMAP  2BAS 170x2x160     3 IMAP 39  ! Investment Basic
REMAP  2TAX 170x2x160     3 IMAP 39  ! Investment Tax
REMAP  1MAR 170x2x160x8   3 IMAP 39  ! Margins Intermediate
REMAP  2MAR 170x2x160x8   3 IMAP 39  ! Margins Investment
REMAP  1CAP 160           1 IMAP 39  ! Capital
REMAP  1LND 160           1 IMAP 39  ! Land
REMAP  1OCT 160           1 IMAP 39  ! Other Costs
REMAP  1LAB 160x6         1 IMAP 39  ! Labour
COPY
```

DGCOM.INP:

```
temp1.har
temp2.har
dagsup.har
AGGREG  1ARM    170 1 CMAP 39 1PUR 170 ! Armington intermed
AGGREG  2ARM    170 1 CMAP 39 2PUR 170 ! Armington investment
AGGREG  3ARM    170 1 CMAP 39 3PUR 170 ! Armington households
AGGREG  P018    170 1 CMAP 39 4PUR 170 ! Export Elas
AGGREG  XPEL    170 1 CMAP 39 3PUR 170 ! Hhold Expend Elas
REMAP   0TAR    170         1 CMAP 39  ! Tariff revenue
REMAP   4BAS    170         1 CMAP 39  ! Exports Basic
REMAP   4TAX    170         1 CMAP 39  ! Export Tax
REMAP   MAKE    170x39      1 CMAP 39  ! MAKE
REMAP   4MAR    170x8       1 CMAP 39  ! Margins Exports
REMAP   3BAS    170x2       1 CMAP 39  ! Households Basic
REMAP   3TAX    170x2       1 CMAP 39  ! Households Tax
REMAP   5BAS    170x2       1 CMAP 39  ! Government Basic
REMAP   5TAX    170x2       1 CMAP 39  ! Government Tax
REMAP   6BAS    170x2       1 CMAP 39  ! Inventories Basic
REMAP   1BAS    170x2x39    1 CMAP 39  ! Intermediate Basic
REMAP   1TAX    170x2x39    1 CMAP 39  ! Intermediate Tax
REMAP   2BAS    170x2x39    1 CMAP 39  ! Investment Basic
REMAP   2TAX    170x2x39    1 CMAP 39  ! Investment Tax
REMAP   1MAR    170x2x39x8  1 CMAP 39  ! Margins Intermediate
REMAP   2MAR    170x2x39x8  1 CMAP 39  ! Margins Investment
REMAP   3MAR    170x2x8     1 CMAP 39  ! Margins Households
REMAP   5MAR    170x2x8     1 CMAP 39  ! Margins Government
COPY
```

DGMAR.INP:

```
temp2.har
agg39.har
dagsup.har
REMAP  4MAR 39x8         2 MMAP 2 ! Margins Exp
REMAP  1MAR 39x2x39x8    4 MMAP 2 ! Margins Intermediate
REMAP  2MAR 39x2x39x8    4 MMAP 2 ! Margins Investment
REMAP  3MAR 39x2x8       3 MMAP 2 ! Margins Hou
REMAP  5MAR 39x2x8       3 MMAP 2 ! Margins Gov
COLLAPSE 1LAB   39x6  2           ! aggregate OCCs
COPY
```

To run the first deck, you would type from the DOS command line:

```
dagg <dgind.inp
```

A host of messages appear, which you could trap in a log file:

```
dagg <dgind.inp >dgind.log
```

But we cannot be satisfied until we have *mechanized the whole job*! So we construct a little batch file like that below:

DG39.BAT:

```
REM job to aggregate large Taiwan database to 39 sectors
del temp1.har
del temp2.har
del agg39.har
del *.log
dagg <dgind.inp >dgind.log
if errorlevel 1 goto err
dagg <dgcom.inp >dgcom.log
if errorlevel 1 goto err
dagg <dgmar.inp >dgmar.log
if errorlevel 1 goto err
dir/od *.har
echo job done OK
goto endbat
:err
dir/od *.log
echo JOB FAILED !!!!!!!!! Examine last log !!!!!!!!!!!!!
:endbat
```

Type `dg39.bat` to run all 3 DAGG jobs.

Creating a batch file has several important advantages:

- It makes it easy to repeat a whole sequence of tasks, whilst eliminating the possibility that we may inadvertently omit one of the steps along the way.
- It serves as a permanent record of what was done. In 6 month's time, today's data job will be merely a directory full of unidentified files. We can look at BAT files like the above to reconstruct what was done.

DG39.BAT illustrates nearly all you need to know about batch files:

- The `REM` line (means "remark') is nothing but a comment.
- The next lines delete all the files that we expect our jobs to produce. That way, if any of these files exist after the BAT has run we know that the BAT produced them just now – they are not lying around from a previous run. It becomes impossible for, say, DGMAR to run if DGCOM failed. We won't be confused by obsolete log files.
- Like any GEMPACK program, if DAGG finishes correctly it sets the DOS system variable "`errorlevel`" to 0. If DAGG aborts with a fatal error, `errorlevel` is set to greater than 0. The condition "`if errorlevel 1`" is true if `errorlevel` is 1 or more (this is a DOS quirk). So control passes to line "`:err`" (near the end of the file) if there is any problem. If there is no problem the next command is executed.
- If all goes well, the command `dir/od *.har` lists all HAR files in order of creation date – we would expect to see AGG39.HAR at the bottom of the list. Finally the `ECHO` command posts a cheery message to the screen.
- If any job failed, `dir/od *.log` lists all LOG files in order of creation date. If the last file is DGCOM.LOG we know that the job failed at the DGCOM stage: the bottom of the log file will contain a clue as to why.

The final stage of the aggregation task will be to restore the set labelling information removed by DAGG. As explained in Section 7, this can be done by using the aggregated file as input to a model simulation with zero shock. The updated data will be labelled but will have the same numbers. The model should also (if well designed) compute and store various check totals and summary data. These should be examined to see if the DAGG jobs worked as intended.

9. Worked example of DisAGGregation

In this section we go through the steps involved in disaggregating an entire model database. We illustrate the procedure for a miniature 4-sector 1990 Indonesian database. The structure of that database is identical to that of the ORANIG database discussed in the previous section. Indeed, this section assumes that you have read the previous section on aggregating such a database.

Our task is to disaggregate the "manufacturing" sector into two parts, "heavy" and "light". There is some data from 1987 about the different sales patterns of these two subsectors as follows:

Excerpt from 1987 Commodity Details Matrix:
Demands for Domestically-produced Manufactures, Producer Prices

	Agricult	Mining	Manfact	Services	H'holds	Govt	Invest	Stocks	Exports	Total
Heavy	1709	957	30549	822	6095	428	12003	-457	8414	60521
Light	2563	638	13093	7397	54857	1710	28008	-685	17880	125460
Total	4272	1595	43642	8219	60953	2138	40011	-1142	26294	185981

For imports we only know that 70% of imports of manufactures are "light". There is no information which would enable us to assign a different cost structure to heavy and light subsectors.

The miniature database has 4 sectors, 1 margin (Services) and only 1 occupation. Since no margin commodity is being split, only 2 DAGG jobs are needed; one to split the Manufacturing commodity, and one to split the corresponding industry. The order of work is to:

(a) use the above data to construct a series of 6 splitting weight matrices for Intermediate, Investment, Household, Exports, Government and Stocks flows. These weight matrices are stored in a HAR file with headers COM1, COM2, .., up to COM6. We also need 2 more vectors of length 2: one (IMPS) to split the tariff duty vector in proportion to total imports, and one (IND0) to split the commodity row in the MAKE matrix in proportion to total sales. However, the proportions for IND0 are unknown at this stage: for now we assume a 1:1 proportion.

(b) In DAGG, use these weight matrices to split the Manufacturing (commodity) row in all relevant matrices. Domestic and import flows will be treated simultaneously. The DUPLICATE command will be used to stretch parameter vectors of commodity length. SPLIT and DUPLICATE commands are covered in Section 2.

(c) In DAGG, use the same IND0 splitting vector, length 2, to SPLIT the Manufacturing (industry) column in all relevant matrices. In the first instance we assume that both industries are of equal size. The DUPLICATE command will be used to stretch parameter vectors of industry length.

(d) Run the enlarged, 5x5 database through the model; this will calculate and display the value of costs and sales for heavy and light subsectors [simulation results will be invalid, since industry costs will not equal commodity sales. However, the computed values for sales are not affected by our (false) assumption about costs.

(e) Update the values in the IND0 vector to reflect the computed values of Heavy and Light sales. Then repeat steps (a) to (d), this time splitting the Manufacturing (industry) column in proportion to the computed total sales values.

(f) Manually modify the MAKE matrix so that it becomes diagonal (see Section 2.4).

(g) Repeat step (d), using a zero shock. This time costs should equal sales for the two new commodities. Also the updated database, whilst containing the same numbers as the input data, will contain the set labelling information (row and column labels) that DAGG strips away.

Running through steps (a) to (e) twice takes advantage of the fact that the main TAB file for the model already contains facilities to compute total sales of each commodity. Alternatively, we could construct a special TAB file which would calculate sales of heavy and light from the 5-commodity, 4-industry database which is produced by step (b). That would be more effort.

DAGG decks DGCOM.INP and DGIND.INP

Here are the two DAGG instruction files which do the job.

File DGCOM.INP

```
model.har
temp.har
dagsup.har
SPLIT   1BAS  4*2*4    1 3 2  COM1 2*2*4    ! V1BAS Intermediate Basic
SPLIT   1MAR  4*2*4*1  1 3 2  COM1 2*2*4*1  ! V1MAR Intermediate Margins
SPLIT   1TAX  4*2*4    1 3 2  COM1 2*2*4    ! V1TAX Intermediate Tax
SPLIT   2BAS  4*2*4    1 3 2  COM2 2*2*1    ! V2BAS All-ind Investment Basic
SPLIT   2MAR  4*2*4*1  1 3 2  COM2 2*2*1*1  ! V2MAR All-ind Investment Margins
SPLIT   2TAX  4*2*4    1 3 2  COM2 2*2*1    ! V2TAX All-ind Investment Tax
SPLIT   3BAS  4*2      1 3 2  COM3 2*2      ! V3BAS Households Basic
SPLIT   3MAR  4*2*1    1 3 2  COM3 2*2*1    ! V3MAR Households Margins
SPLIT   3TAX  4*2      1 3 2  COM3 2*2      ! V3TAX Households Tax
SPLIT   4BAS  4        1 3 2  COM4 2        ! V4BAS Exports Basic
SPLIT   4MAR  4*1      1 3 2  COM4 2*1      ! V4MAR Exports Margins
SPLIT   4TAX  4        1 3 2  COM4 2        ! V4TAX Exports Tax
SPLIT   5BAS  4*2      1 3 2  COM5 2*2      ! V5BAS Government Basic
SPLIT   5MAR  4*2*1    1 3 2  COM5 2*2*1    ! V5MAR Government Margins
SPLIT   5TAX  4*2      1 3 2  COM5 2*2      ! V5TAX Government Tax
SPLIT   6BAS  4*2      1 3 2  COM6 2*2      ! V6BAS Inventory Changes
SPLIT   MAKE  4*4      1 3 2  IND0 2*1      ! MAKE MAKE multiproduction matrix
SPLIT   0TAR  4        1 3 2  IMPS 2        ! V0TAR Tariff Revenue
DUPLICATE  2ARM  4 1 3 2  ! SIGMA2 Investment Armington
DUPLICATE  3ARM  4 1 3 2  ! SIGMA3 Households Armington
DUPLICATE  1ARM  4 1 3 2  ! SIGMA1 Intermediate Armington
DUPLICATE  P018  4 1 3 2  ! EXP_ELAST Traditional Export Elasticities
DUPLICATE  XPEL  4 1 3 2  ! EPS Household Expenditure Elasticities
COPY
```

File DGIND.INP

```
temp.har
model5.har
dagsup.har
SPLIT   2BAS  5*2*4   3 3 2 IND0  1*1*2     ! V2BAS All-ind Investment Basic
SPLIT   2TAX  5*2*4   3 3 2 IND0  1*1*2     ! V2TAX All-ind Investment Tax
SPLIT   1TAX  5*2*4   3 3 2 IND0  1*1*2     ! V1TAX Intermediate Tax
SPLIT   1BAS  5*2*4   3 3 2 IND0  1*1*2     ! V1BAS Intermediate Basic
SPLIT   1MAR  5*2*4*1 3 3 2 IND0  1*1*2*1   ! V1MAR Intermediate Margins
SPLIT   2MAR  5*2*4*1 3 3 2 IND0  1*1*2*1   ! V2MAR All-ind Investment Margins
SPLIT   1OCT  4       1 3 2 IND0  2         ! V1OCT Other Costs
SPLIT   1CAP  4       1 3 2 IND0  2         ! V1CAP Capital
SPLIT   1LND  4       1 3 2 IND0  2         ! V1LND Land
SPLIT   1LAB  4*1     1 3 2 IND0  2*1       ! V1LAB Labour
SPLIT   MAKE  5*4     2 3 2 IND0  1*2       ! MAKE  multiproduction matrix
DUPLICATE  P028  4 1 3 2 ! SIGMA1PRIM Primary Factor Sigma
DUPLICATE  SLAB  4 1 3 2 ! SIGMA1LAB Labour Sigma
DUPLICATE  SCET  4 1 3 2 ! SIGMA1OUT Output Sigma
COPY
```

As explained in Section 8, these files were produced via the commands:

```
find "COM" template.dag >dgcom.inp
find "IND" template.dag >dgind.inp
```

and then using search-and-replace commands in a text editor. The jobs are run from the DOS command line thus:

```
dagg <dgcom.inp >dgcom.log
dagg <dgind.inp >dgind.log
```

Selected lines from the DAGG input files are explained below.

Using the DUPLICATE command

The command to stretch the vector of Armington elasticities is:

```
DUPLICATE  1ARM  4 1 3 2  ! SIGMA1 Intermediate Armington
```

The sequence 4 1 3 2 means:

4 original length of 1ARM

1 split in 1st dimension (there is only one in this case)

3 split 3rd element (Manufacturing)

2 into 2 parts, each identical to the original.

All following the "!" is a comment, which is ignored.

Using the SPLIT command; constructing commodity splitting matrices

The command to stretch the vector of exports at basic values is:

```
SPLIT   4BAS  4 1 3 2  COM4 2        ! V4BAS Exports Basic
```

COM4 is a vector length 2 with elements {8414, 17880}, drawn from the data above. DAGG will use the ratio of these two numbers to split the original export flow of manufactures. The sequence 4 1 3 2 has the same meaning as for the DUPLICATE command.

The same weights are used to split the V4MAR matrix of export margins (note there is only 1 margin commodity).

```
SPLIT   4MAR  4*1  1 3 2  COM4 2*1      ! V4MAR Exports Margins
```

Notice how the dimensions of COM4 have been padded with "*1" so that the weight matrix has the same number of dimensions as the matrix to be split. In the dimension to be split, the weight matrix has the same length as the number of new elements; the other dimensions must either be the same as those of the matrix to be split, or be 1 (meaning weights are unchanging within that dimension).

The original vector of government demands has dimensions 4*2 (COM*SRC), and we want to split domestic and imports differently. The weight matrix COM5 is:

	dom	imp
Heavy	428	3
Light	1710	7

It is important that the order of the dimensions in the weight matrix is the same as that in the matrix to be split: the weight matrix has dimensions NEWCOM*SRC. If we inadvertently used the transpose of the above 2*2 matrix as our weights, DAGG would not notice, but the results would be wrong. DAGG uses only the column proportions; the "imp" could equally have contained {30,70}. The DAGG command to split government demands is:

```
SPLIT   5BAS  4*2      1 3 2  COM5 2*2      ! V5BAS Government Basic
```

While margins on government demands are split by:

```
SPLIT   5MAR  4*2*1    1 3 2  COM5 2*2*1    ! V5MAR Government Margins
```

5MAR is of size COM*SRC*MAR while COM5 is originally of size COM*SRC. By padding the dimensions of COM5 at right with "*1", we indicate to DAGG that the two dimensions of COM5 are to be associated with the first two dimensions of 5MAR.

To split intermediate demands, size COM*SRC*IND, we use a full weight matrix COM1, size NEWCOM*SRC*IND (order of dimensions is important).

The original matrix of investment demands has dimensions 4*2*4 (COM*SRC*IND), and we want to split domestic and imports differently, but apply the same splitting ratios over all IND. The weight matrix COM2 is:

	dom	imp
Heavy	12003	3
Light	28008	7

and the DAGG command to split basic investment demands is:

```
SPLIT   2BAS  4*2*4    1 3 2  COM2 2*2*1    ! V2BAS All-ind Investment Basic
```

Again, the final "*1" in the dimensions of COM2 indicates that each industry's investment is to be split in the same way.

Turning to the second file, DGIND.INP, we see that the same splitting vector, IND0, is used throughout. This reflects our ignorance of any difference in the cost structures of the two new sectors. Also, we assume that investment in the subsectors is apportioned according to output. As explained previously, we initially set the IND0 weights to {1,1}, replacing these figures later with sales figures for the new sectors.

The command to split the industry dimension of intermediate demands is:

```
SPLIT   1BAS  5*2*4   3 3 2 IND0  1*1*2     ! V1BAS Intermediate Basic
```

1BAS is of size COM*SRC*IND - notice how COM is now 5 (since COM dimensions have already been split). The sequence 3 3 2 means:

3 split in 3rd (IND) dimension

3 split 3rd element (Manufacturing)

2 into 2 parts, each identical to the original.

The dimensions of IND0, NEWIND (=NEWCOM), have been padded at the left with "1"s, meaning that the splitting weights do not depend on COM or SRC.

<end of document>