

# Enhancements for GEMPACK Release 10.0-001 (April 2009)

Mark Horridge, Michael Jerie, Ken Pearson

## Contents

<b>1. BETTER SUPPORT FOR SOURCE-CODE USERS OF THE INTEL COMPILER</b>	<b>1</b>
<b>1.1 Support for Intel Fortran 11</b>	<b>1</b>
1.1.1 Installing Intel Fortran 11	1
<b>1.2 Use of SSE2 instructions</b>	<b>1</b>
<b>1.3 More aggressive optimization</b>	<b>2</b>
1.3.1 More aggressive optimization for libraries	2
1.3.2 More aggressive optimization at LTG time	3
<b>2. FASTER LU DECOMPOSITION</b>	<b>3</b>
<b>3. SPEEDUP OF EXECUTABLE-IMAGE VERSION</b>	<b>4</b>
<b>4. RELAXED LIMITS FOR SIZE-RESTRICTED GEMPACK USERS</b>	<b>4</b>
<b>5. NEW FEATURES FOR PROGRAM CMBHAR (COMBINES HAR FILES)</b>	<b>5</b>
<b>5.1 Introducing CMBHAR</b>	<b>5</b>
<b>5.2 New CMBHAR features and options</b>	<b>6</b>
5.2.1 Determining which headers appear on the output file.	6
5.2.2 Forming elements of the new set COMBINED.	6
<b>6. ENHANCEMENTS TO WINDOWS PROGRAMS</b>	<b>7</b>
<b>6.1 Analysing Closure Problems with AnalyseGE</b>	<b>7</b>
6.1.1 Using AnalyseGE Colouring to Find and Fix Closure Problems	8
<b>7. NEW COMMAND FILE STATEMENTS</b>	<b>11</b>
<b>8. INDEX</b>	<b>12</b>

## Enhancements for GEMPACK Release 10.0-001 (April 2009)

No serious bugs have been detected in the year since GEMPACK Release 10.0 (May 2008) was introduced: Release 10.0-001 includes minor bug fixes and some enhancements. These enhancements are briefly described below.

### 1. Better support for source-code users of the Intel compiler

Release 10.0-001 introduces several improvements aimed specifically at source-code users of the Intel Fortran compiler. These will be of less interest if you use the Lahey LF95 compiler. However, if you are still using the older Lahey LF90 compiler, *which will not be supported by GEMPACK 11 and later*, this section might influence you to switch to Intel.

#### 1.1 Support for Intel Fortran 11

Release 10.0-001 allows you to use Intel Fortran Version 11 (previously only Version 10 was supported). If you are already using Intel Fortran 10.1 to produce 32-bit EXEs, there is no reason to upgrade to Version 11. If you use Intel Fortran to produce 64-bit EXEs, and if your Intel Fortran is older than Version 10.1.019 (Build 20080212) you should upgrade to Version 10.1.019 or later (for 64-bit, earlier editions of IF Version 10 had a bug in the optimizer, which prevented GEMPACK from using faster-running O2 compile options; the bug was fixed for Version 10.1.019).

##### 1.1.1 Installing Intel Fortran 11

If you plan to install Version 11 of Intel Fortran to use with Release 10.0-001 of GEMPACK, please note that

- Version 11 of Intel Fortran uses a different install procedure from Version 10. In particular, the installation instructions in section 3.3 of GPD-6 (May 2008) do not apply to Intel 11.
- You can find detailed installation instructions for Intel 11 on the GEMPACK web site at <http://www.monash.edu.au/policy/gpifort.htm>

#### 1.2 Use of SSE2 instructions

SSE2 is a feature introduced by Intel for the Pentium 4 processor in 2001, and included since then in nearly all processors from any manufacturer. The Intel Fortran compiler is capable of making very effective use of SSE2 in conjunction with its /O2 optimization switch.

The problem is that an EXE containing SSE2 instructions might not work on an older PC which did not support SSE2. Intel has provided a solution: within the EXE are alternate compiled subroutines, not using SSE2, corresponding to the small number of routines that benefit from SSE2. When the EXE is launched, it detects if SSE2 is available; if not, the alternate (non-SSE2) code is used. The solution leads to a small increase in EXE size, but ensures that the EXE runs on older PCs, such as those using the Pentium 3 processor.

Release 10.0-001 applies compiler settings for Intel Fortran (10 or 11) which implement Intel's solution. If you use the Intel compiler, your TABLO-generated EXE files will run faster on the bulk of PCs that support SSE2, but will still run on PCs which do not.

As mentioned above, the advantage of SSE2 code only appears if /O2 optimization is used. We discuss /O2 optimization below and mention there the possible speedup that arises from the O2/SSE2 combination.

### 1.3 More aggressive optimization

We remind the reader that source-code GEMPACK uses the Fortran compiler at two times:

**A:** When you install GEMPACK, the program BuildGP compiles a great number of GEMPACK sub-routines to produce libraries. Included in these libraries is all the code which is used by every model, such as the LU decomposition routines used in the linear phase of GEMPACK's solution process. In principle, you only need to do this compilation once – when you install GEMPACK.

**B:** If you change your model specification (ie, edit the TAB file), you need to run TABLO again to produce Fortran code specific to your model. Then you run LTG to compile that code. Especially during model development, you may find yourself running LTG quite often.

More aggressive optimization by the compiler could result in faster running EXE files – at the expense of increasing the time for compilation. However, an increase in BuildGP's run time is not too painful, as you only need to wait once. By contrast, slower LTG times could annoy you frequently.

#### 1.3.1 More aggressive optimization for libraries

By default, Release 10.0-001 applies /O2 optimization compiler settings for Intel Fortran (10 or 11) for all the compiling that happens when you install GEMPACK (ie, case A above). In previous Releases the less aggressive /O1 optimization level was used. The new setting applies not only to the GEMPACK code libraries, but also to programs such as TABLO, GEMSIM or SLTOHT.

The combination of /O2 optimization and SSE2 instructions can result in considerable time saving during the LU (linear system solution) phase of a simulation. Some examples are shown below.

**Table 1: Run times (seconds) for 4 models and 2 levels of library optimization**

	TERM	MONASH	GTAP	MMRF4
O1 no SSE2	52	8	150	212
O2 + SSE2	28	7	103	199
% cut	-46	-5	-31	-6

TERM and GTAP gain most because they spend most time during the LU phase; this uses the same library code for every model. By contrast, MONASH and MMRF4 spend most time evaluating coefficients in equations. The model-specific code that does this is created at LTG time, which uses different compiler settings (see below). Hence, these two models gain less from library optimization.

Optimization works by doing calculations in a different, quicker, way; for example by re-ordering arithmetic operations. Since computer arithmetic is not exact, the changed calculation method could lead to very tiny changes in results. We notice these tiny changes during our testing procedures -- we re-run hundreds of simulations to check that new GEMPACK code produces the same answers as older versions. However, we have no reason to believe that results computed using optimization are either more or less accurate than those computed without optimization.

#### *Exception for early versions of 64-bit Intel Fortran 10*

Early versions of 64-bit Intel Fortran 10 (those prior to Version 10.1.019 [Build 20080212]) had a bug in the optimizer, which prevented GEMPACK from using faster-running /O2 compile options. The BuildGP installer detects this case, and applies /O1 compiler options instead. There will be a performance penalty. If you are using this older Intel Fortran for 64-bit work, you may consider updating. The problem does not apply to the 32-bit version.

#### *BAT files to change optimization level for GEMPACK libraries.*

Although we see no reason why you should wish to change the optimization level used to compile libraries, we have supplied two BAT files which do that job.

- Run O1FIG.BAT to apply the /O1 settings used in earlier GEMPACK releases.

- Run O2FIG.BAT to restore the /O2 settings which are the new default.

You need to run BuildGP again after running either of the above BAT files.

### 1.3.2 More aggressive optimization at LTG time

By default, Release 10.0-001 applies /O1 optimization compiler settings for Intel Fortran (10 or 11) for the compiling that happens when you run LTG (ie, case B above). In previous Releases no optimization was used during LTG. The effect of the new default settings is that LTG takes longer, but the model runs faster.

**Table 2: LTG and run times (seconds) for 4 models and 2 levels of LTG optimization**

Optimization	TERM		MONASH		GTAP		MMRF4	
	LTG	Run	LTG	Run	LTG	Run	LTG	Run
none	15	36	25	15	16	103	78	696
/O1	68	28	145	7	86	103	534	199

As can be seen from the table above, the balance of advantage depends very much on the model. For GTAP, moving to /O1 hardly affects run time, but LTG takes nearly 5 times as long. For MMR4, LTG time increases to 9 minutes, but run time falls by nearly the same amount.

The run time savings will be more useful if a particular EXE is run many times. That situation is normal for the multiperiod recursive-dynamic models which are becoming more popular and which are often run under RunDynam (or RunMONASH etc). For example, a 30-year MMR4 simulation might require 90 separate simulations. In this case the /O1 setting, which runs 3 times faster, is obviously much better. On the other hand, if you were adding new code to TERM, and had to run LTG frequently, the /O1 setting could be irritating.

#### *BAT files to change LTG optimization level*

Since you will want to make the choice yourself, we have supplied two BAT files to help.

- Run LTGfigo0.BAT to use no optimization during LTG (the old default).
- Run LTGfigo1.BAT to use /O1 optimization during LTG (the new default).

You do **NOT** need to run BuildGP again after running either of the above BAT files. You might choose to switch off optimization while developing a model (ie, running LTG frequently), then run LTGfigo1.BAT as you start running more simulations.

## 2. Faster LU decomposition

GEMPACK uses sparse matrix techniques during the linear solution (LU) phase of model solving. As the linear system is solved, 'fill-in' occurs, and the remaining part of the LHS matrix becomes less sparse. At a certain point, a switch is made to simpler routines which take no account of sparsity.

Up to now, this switch has occurred when the LHS matrix became 50% non-zero. We experimented with the trigger value, and found that a value of 40% (ie, switching to dense processing earlier) reduced solution time in many cases but did not increase solution time for any of the models that we tried. So this new 40% value has been set as the default in Release 10.0-001.

For example, the time for a Johansen solution of the TERM model with 32 sectors and 27 regions was reduced from 56 to 48 seconds. The effect was stronger when the "iz1=no;" option had been specified in the CMF file.

For some other models, especially models which spend little time in the LU phase, there may be little speed improvement.

This change may alter numerical results very slightly. If you want to revert to the older, 50%, threshold, include the line:

```
debug options = 207 ;
```

in your CMF file.

### 3. Speedup of Executable-image Version

Many of the EXE files supplied with the Executable-image Version of GEMPACK are compiled with the Intel compiler – and for this release we have used the faster compile settings described above. The speedup is more noticeable on larger models which spend most time in the LU phase. For example, GEMSIM solve time for a 30-sector, 30-region, 3-5-7 Gragg GTAP simulation was reduced from 200 seconds to 139 seconds. Speedup in other situations will probably be less pronounced, but still noticeable.

### 4. Relaxed limits for size-restricted GEMPACK users

GEMPACK imposes a size limit on the model you can solve in two cases:

- if you are using GEMSIM with a Limited Executable-Image Licence.
- if you have no GEMPACK licence, and you are running a model-specific EXE file (TABLO-generated program EXE) created by someone else who has a Source-Code licence.

The limits in both these cases are the same, and have been significantly relaxed for Release 10.0-001. Old and new limits are shown in the table below.<sup>1</sup>

The new limits are chosen so that you could solve a typical single region model with up to 40 sectors, or a multi-region model (like GTAP) with 12 sectors and 12 regions.

If you have no licence and are running a model-specific EXE file created by someone else who used an earlier GEMPACK version, the previous lower limits apply.

In older GEMPACK documentation, the limits below were described as defining "medium-sized" models.

**Table 3: Limits for size-restricted GEMPACK users**

Description	Limit for GEMPACK Release 10.0 and earlier	Limit for GEMPACK Release 10.0-001
Endogenous variables (condensed plus backsolved)	30,000	75,000
All variables (condensed plus backsolved)	35,000	120,000
Endogenous variables in the condensed system	10,000	35,000
Total variables in the condensed system	16,000	70,000
Nonzeros on the LHS at any step	75,000	175,000
Components of vector variables with size >40	1000	1000
Total number of reals and integers used for coefficients (pre-sim and updated)	250,000	300,000

<sup>1</sup> The "old limits" are as listed in section 6.2.1 of the Release 10.0 version (May 2008) of GPD-7.

## 5. New features for program CMBHAR (combines HAR files)

### 5.1 Introducing CMBHAR

Multi-period, recursive-dynamic CGE models, such as MONASH or Dynamic-GTAP, are solved for a number of years in sequence. A solution file (of year-to-year % changes) and updated data files are produced for each year.

ViewSOL and other programs built into the RunDynam interface automatically combine solution files from different years so that time trends for variables can be displayed. But to see time trends in the data files has been harder, since the process has been less automated. One reason is that traditionally the data files mainly contain flow values measured in current prices. Year-to-year comparisons of these flows are not very informative, as differences conflate the effects of price and quantity changes.

However, where the model database contains matrices measured in physical units, year-to-year comparison becomes interesting. For example, databases for environment-oriented CGE models often tabulate tonnes of CO<sub>2</sub>, petajoules of energy, gigalitres of water, or hectares of land. We may reasonably wish to see how these quantities evolve over time.

For some time the command-line program CMBHAR has been available for this task. Briefly, CMBHAR reads a sequence of header array (HAR) files, usually produced as updates by a sequence of model simulations. The assumption is that all input files contain same-sized arrays at the same headers -- only the numerical values are different. CMBHAR outputs a single header array file with the same headers as the input files. Each output array has an additional "time" dimension, corresponding to the sequence of input files. Hence the output file includes all the information from the input files.

For example, suppose each input file had a header "EMIT" containing a matrix dimensioned GAS\*IND that showed how much of several gases are emitted by each industry. Header "EMIT" in the output file produced by CMBHAR would contain an array dimensioned GAS\*IND\*COMBINED. Each element of the new set COMBINED corresponds to one of the input files, and, very often, to a particular year.

CMBHAR has suffered from a lack of publicity and from an annoying glitch: it refused to work whenever the first HAR file of the sequence contained a real header not present in one or more subsequent files. This case often occurred, since year-0 model data files often contain headers which are not used by the model and so are not transmitted to updated data files.

Release 10.0-001 fixes this glitch: now CMBHAR will concatenate all real headers (of matching size) that appear on every file. Other headers that appear on only some input files are either ignored (if CMBHAR option CHO is used) or, if CMBHAR option AHO is used, have a special value (currently -99,999.0) added to fill in gaps in the combined array.

In addition CMBHAR will be available from within RunDynam (from Version 3.41), making it easy to see how model data has evolved over time.

### 5.2 New CMBHAR features and options

The new features of CMBHAR (version 2.4, March 2009) are:

- By default (option AHO), all real headers on input header array files are put on the output file. Option CHO is provided so that only headers common to all input files are put on the output file.
- Elements of the set COMBINED are formed from the names of the input header array files. Two new options (SNE and NEP) are provided to allow the user to either specify each element, or, use a rule for element name formation.
- The new option SCN (Specify Combined set Name) allows the user to specify the set name for the new "time" dimension instead of using the default name "COMBINED".

- When the user does not use SCN to specify a new set name then the default name "COMBINED" is used. If an output header is found to already have a dimension labelled with set COMBINED then the new "time" dimension will be labelled with new set COMBINEDX where X is the smallest integer which makes COMBINEDX a new unique set name.

### 5.2.1 Determining which headers appear on the output file.

Now by default all real headers on all input files are included on the output file. Headers which are not found on all files are combined with the dummy real value -99,999.0 filling the dimensions which correspond to the missing headers.

If you choose the CHO option (Common Headers on Output file), only those real headers which are common to all input files are combined and put on the output file. Headers which are missing from one or more of the input files are skipped.

### 5.2.2 Forming elements of the new set COMBINED.

CMBHAR now tries to form the COMBINED set element names by taking the names of the input header array files (truncated to 12 characters if necessary), and replacing illegal element name characters with the '@' character. For example the input file basb-2001.sl4 will result in the set element basb@2001 being formed. Note that set elements must not be longer than 12 characters and consist of letters (A to Z, a to z), digits (0 to 9), underscores '\_' and the character '@', and must commence with a letter (see section 4.2.1 of GPD-2). If truncation and character replacement leads to duplicate element names being formed then the elements of COMBINED are taken as t1, t2, t3 etc.

Alternative methods of forming the elements of the COMBINED set are provided by the SNE and NEP options.

If you select option SNE (Specify New set Elements), you will be prompted for the associated element name after you specify each input header array file name.

If you select option NEP (New set Elements from common Prefix) the new set elements will be formed from a common prefix and an automatically incremented integer suffix. For example the set elements

year\_2001, year\_2002, ... ,year\_2010

are formed from the common prefix string "year\_" with an integer suffix starting with "2001". Consecutive elements are formed by incrementing the previous integer suffix by 1. You will be prompted for a common prefix string, a starting integer suffix, and a common increment for the suffix.

## 6. Enhancements to Windows programs

Various small enhancements have been made to Windows programs: use each program's Help ... What's New menu command to get details. In particular:

### ViewHAR

There are two new buttons in the Set Library Window : **Rebuild Set Library** recreates the set library, using only sets from the currently open file; **Rename Set** allows you to change the name of a set.

From the Contents view you can now right-click on a real matrix and select **Change Values**. Three options are presented:

- (a) set all elements to a common value
- (b) set minute or zero elements to a small number
- (c) set tiny elements to zero

A new option, **AccFlex**, has been added to the Number of Decimal Places choice. The AccFlex (Accurate Flexible) format is similar to Flex, but it shows 7 significant figures and drops down to scientific notation for very tiny numbers

## TABmate

**Edit..ReArrange selection** command lets you re-wrap text.

## ViewSOL

Improvements to Time Series Mode. There is a more helpful message when percent change variables change sign. And ViewSOL now automatically filters out larger variables to keep total (all-solution) memory needs within 500 MB

## 6.1 Analysing Closure Problems with AnalyseGE

If you have a bad closure (wrong number of exogenous variables) or a singular LHS matrix (either numerically singular or structurally singular), you may find it useful to be able to load the closure into AnalyseGE. The colouring of variables there may help you find and fix the problem.

We are grateful to James Giesecke who suggested that we add this feature. He has often fixed closure problems by colouring variables in equations (even doing that by hand before AnalyseGE came along). In section 6.1.1 below, James describes some of the ways he uses the colouring in AnalyseGE to find and fix closure problems.

Suppose that you have a simulation which ends with a bad exogenous/endogenous count or with a singular matrix. If you want to load the closure into AnalyseGE, you can add the statement

```
simulation = analyse-closure ;
```

to your Command file and run the simulation again. In fact no simulation will be run but you will produce a Solution file and an SLC file. You can load these into AnalyseGE and you will see the normal colouring of exogenous, endogenous and shocked variables. Then you can follow the suggestions in section 6.1.1 below to try to find and fix your closure problem. Once fixed, change the closure in your Command file, remove the "simulation = analyse-closure ;" statement and run the simulation again.

You should be aware of some things about the Solution and SLC file produced when you have the above "simulation = analyse-closure ;" statement in your Command file.

- The Solution file is rather odd because the values of all endogenous variables are zero. [A simulation is not really done. Pretending that all endogenous variables are zero is just our way of fooling AnalyseGE and ViewSOL that this is a genuine Solution file.]
- The values on the SLC file are the pre-simulation values for all Coefficients as usual. You may find that these values help you to identify why the LHS matrix is singular.
- Most importantly, the closure on the Solution file and reflected in the colouring of variables in AnalyseGE is correct.

### 6.1.1 Using AnalyseGE Colouring to Find and Fix Closure Problems

Here are suggestions from James Giesecke for using the colouring in AnalyseGE to find and fix closure problems – either bad count (wrong number of exogenous variables) or singular LHS Matrix (numerically singular or structurally singular).

Problems finding an appropriate model closure sometimes occur when building a new model from scratch, or when adding lengthy and detailed new theory to an existing model. It is a less common problem when doing routine simulations with an existing model, such as, for example, ORANI-G. Nevertheless, since ORANI-G is familiar to many GEMPACK users, and since it is among the example models packaged with the standard GEMPACK installation, we shall use it below to illustrate the types of closure issue that may arise during model development.

Under the default options of AnalyseGE, exogenous variables are italic red, and endogenous variables green. In the examples that follow, exogenous variables are displayed as bold.

Excerpt 31 (see below) is the ORANI-G investment theory. ORANI-G provides several options for determining industry-specific investment. A common choice is to relate industry-specific investment to industry-specific rates of return, while holding national investment exogenous. To implement this choice, we must impose the following closure on this excerpt:

Endogenous: ggro, gret, x2tot, p1cap, p2tot, finv2, finv3, invslack, f2tot.

Exogenous: x1cap, finv1, x2tot\_i, x3tot.

When presented as a simple list of endogenous and exogenous variables, the economic meaning of this closure is somewhat opaque. But when viewed in AnalyseGE, the economic sense is clear. Example 1 illustrates the closure as it appears in AnalyseGE. The exogenous variables are shown as bold in Example 1 below (they would be red in AnalyseGE). p1cap and p2tot are determined elsewhere in the model, hence E\_gret must determine gret. With gret thus explained, with finv1 exogenous, Equation E\_finv1 must be determining ggro(i). Hence, with x1cap exogenous, Equation E\_ggro determines industry investment, x2tot. The endogenous status of invslack in equation E\_finv1 makes sense when we see that aggregate investment (x2tot\_i) is exogenous. With aggregate investment exogenous, it is obvious from E\_f2tot that f2tot must be endogenous. With investment determined by rates of return, it is clear that equations E\_finv2 and E\_finv3 must be rendered inoperative via endogenous determination of finv2 and finv3. Interpreting a valid closure is easy with AnalyseGE. In the same way, so too is interpreting an invalid closure.

**EXAMPLE 1: A valid closure**

```
! Excerpt 31 of TABLO input file: !
! Investment equations !

Variable
(all,i,IND) ggro(i) # Gross growth rate of capital = Investment/capital #;
(all,i,IND) gret(i) # Gross rate of return = Rental/[Price of new capital] #;
(all,i,IND) finv1(i) # Shifter to enforce DPSV investment rule #;
(all,i,IND) finv2(i) # Shifter for "exogenous" investment rule #;
(all,i,IND) finv3(i) # Shifter for longrun investment rule #;
invslack # Investment slack variable for exogenizing aggregate investment #;
f2tot # Ratio, investment/consumption #;

Equation
E_ggro (all,i,IND) ggro(i) = x2tot(i) - x1cap(i);
E_gret (all,i,IND) gret(i) = p1cap(i) - p2tot(i);

Equation E_finv1 # DPSV investment rule #
(all,i,IND) ggro(i) = finv1(i) + 0.33*[2.0*gret(i) - invslack];

Equation E_finv2 # Alternative rule for "exogenous" investment industries #
(all,i,IND) x2tot(i) = x2tot_i + finv2(i);

Equation E_finv3 # Alternative long-run investment rule #
(all,i,IND) ggro(i) = finv3(i) + invslack;

Equation E_f2tot x2tot_i = x3tot + f2tot;
```

Example 2 presents an invalid closure. The full exogenous / endogenous variable count is correct, but the LHS matrix is structurally singular. Our list of endogenous and exogenous variables is now:

Endogenous: ggro, gret, x2tot, p1cap, p2tot, finv3, invslack, f2tot.

Exogenous: finv1, finv2, x2tot\_i, x3tot, x1cap.

Patient inspection of the simulation's full endogenous / exogenous variable list will reveal the error in time. However, finding the error is often faster via inspection of the invalid closure within AnalyseGE. Example 2 illustrates the invalid closure. The simultaneous exogeneity of finv1 and finv2 is immediately clear. If we understand the theory of our model, we will quickly conclude that we have activated

two competing theories for the determination of  $x2tot$ <sup>2</sup>. If we are less sure of the model's theory, we can track down the problem by tracing economic causation, given the incorrect closure. The AnalyseGE display immediately suggests that equation E\_finv2 is a good place to start: with  $x2tot_i$  and  $finv2$  exogenous, E\_finv2 must be determining  $x2tot$ . We see that  $x2tot$  also appears in E\_ggro. With  $x1cap$  exogenous, it seems that E\_ggro must be determining  $ggro$ . Since  $ggro$  appears on the LHS of E\_finv1, it appears that E\_finv1 must be determining  $gret$ . But this cannot be correct, since  $gret$  is determined by E\_gret (and we understand that  $p1cap$  and  $p2tot$  are tied-down elsewhere in the model). We conclude that either of  $finv1$  or  $finv2$  can be exogenous, but not both<sup>3</sup>.

**EXAMPLE 2: An invalid closure: Structurally singular LHS Matrix**

```
! Excerpt 31 of TABLO input file: !
! Investment equations !

Variable
(all,i,IND) ggro(i) # Gross growth rate of capital = Investment/capital #;
(all,i,IND) gret(i) # Gross rate of return = Rental/[Price of new capital] #;
(all,i,IND) finv1(i) # Shifter to enforce DPSV investment rule #;
(all,i,IND) finv2(i) # Shifter for "exogenous" investment rule #;
(all,i,IND) finv3(i) # Shifter for longrun investment rule #;
invslack # Investment slack variable for exogenizing aggregate investment #;
f2tot # Ratio, investment/consumption #;

Equation
E_ggro (all,i,IND) ggro(i) = x2tot(i) - x1cap(i);
E_gret (all,i,IND) gret(i) = p1cap(i) - p2tot(i);

Equation E_finv1 # DPSV investment rule #
(all,i,IND) ggro(i) = finv1(i) + 0.33*[2.0*gret(i) - invslack];

Equation E_finv2 # Alternative rule for "exogenous" investment industries #
(all,i,IND) x2tot(i) = x2tot_i + finv2(i);

Equation E_finv3 # Alternative long-run investment rule #
(all,i,IND) ggro(i) = finv3(i) + invslack;

Equation E_f2tot x2tot_i = x3tot + f2tot;
```

Example 3 (overleaf) is a closure with an insufficient number of exogenous variables. Our list of endogenous and exogenous variables is:

Endogenous:  $ggro$ ,  $gret$ ,  $x2tot$ ,  $p1cap$ ,  $p2tot$ ,  $finv1$ ,  $finv2$ ,  $finv3$ ,  $invslack$ ,  $f2tot$ .

Exogenous:  $x2tot_i$ ,  $x3tot$ ,  $x1cap$ .

Again, careful inspection of the simulation's full list of endogenous and exogenous variables will uncover the error in time. However, inspection of the closure via AnalyseGE quickly shows that nothing is determining  $x2tot$ : we have failed to choose an investment theory. That is, all elements of  $finv1$ ,  $finv2$  and  $finv3$  are endogenous.

<sup>2</sup> Less obviously (given the TABLO excerpt presented here) with  $x1cap$  endogenous, the model cannot find market clearing rental rates or cost-minimising capital inputs. We have over-determined investment and under-determined industry-specific capital markets.

<sup>3</sup> This solves one half of our closure problem. Since we began the example by noting that the exogenous / endogenous variable count is valid, there remains the task of finding the endogenous variable that should be exogenous. In the present example, we would have to look outside Excerpt 31 for that variable.

**EXAMPLE 3: Insufficient number of exogenous variables**

! Excerpt 31 of TABLO input file: !  
! Investment equations !

## Variable

```
(all,i,IND) ggro(i) # Gross growth rate of capital = Investment/capital #;
(all,i,IND) gret(i) # Gross rate of return = Rental/[Price of new capital] #;
(all,i,IND) finv1(i) # Shifter to enforce DPSV investment rule #;
(all,i,IND) finv2(i) # Shifter for "exogenous" investment rule #;
(all,i,IND) finv3(i) # Shifter for longrun investment rule #;
invslack # Investment slack variable for exogenizing aggregate investment #;
f2tot # Ratio, investment/consumption #;
```

## Equation

```
E_ggro (all,i,IND) ggro(i) = x2tot(i) - x1cap(i);
E_gret (all,i,IND) gret(i) = plcap(i) - p2tot(i);
```

## Equation E\_finv1 # DPSV investment rule #

```
(all,i,IND) ggro(i) = finv1(i) + 0.33*[2.0*gret(i) - invslack];
```

## Equation E\_finv2 # Alternative rule for "exogenous" investment industries #

```
(all,i,IND) x2tot(i) = x2tot_i + finv2(i);
```

## Equation E\_finv3 # Alternative long-run investment rule #

```
(all,i,IND) ggro(i) = finv3(i) + invslack;
```

Equation E\_f2tot **x2tot\_i** = **x3tot** + f2tot;**7. New command file statements**

Simulation = YES|no|analyse-closure ; ! "analyse-closure is new - see section 6.1

## 8. INDEX

### A

AnalyseGE  
Analysing Closure Problems, 9

### C

Closure  
Wrong number of exogenous variables, 9, 10  
CMBHAR  
Combines HA files, 7

### G

Giesecke, James, 9, 10

### I

Intel compiler  
Faster runs, 4  
Installing Version 11, 3  
optimisation defaults, 4  
Introductory licence  
Size limits on TG-programs, 6

### L

Limited Exe-Image Version  
Size limits, 6

LU decomposition  
When switch to full processing, 5

### N

Numerically singular LHS Matrix  
Loading closure into AnalyseGE, 9, 10

### O

Optimisation  
Intel compiler, 4

### R

RunDynam, RunMONASH etc, 5

### S

Singular LHS Matrix  
Numerically singular, 9, 10  
Structurally singular, 9, 10  
Size limits  
Limited Exe-Image Version, 6  
TG-programs, 6  
SSE2 instructions, 3  
Structurally singular LHS Matrix  
Loading closure into AnalyseGE, 9, 10