Two Sector Closed Economy CGE Model: Exercises

2025

Contents

Two Sector Closed Economy CGE Model: Exercises 1
Contents 1
1. Introduction 2
2. Model 1: A First CGE Model in GAMS 3
Exercise 1 Adding Model Equations 10
Compilation Errors 13
Execution Errors 14
Dollar ($) Control Directives 16
SOLVE Statement OPTIONS 16
3. Model 1: Policy Experiments 17
Simple CGE Experiments 17
Experiment 1 18
Experiment 2 20
Experiment 3 21
4. Collecting Results 23
Calculating Derived Results 27
5. Experiments as INCLUDE Files 29
Appendices 30
Additional experiments 31
Factor Endowments: Experiment 4 31
Income distribution experiment 31

We assume that participants on this course have taken cgemod's course 'Introduction to GAMS and GAMS Studio' (http://www.cgemod.org.uk/gams_studio.html). We use techniques covered by that

1. Introduction

Writing a first CGE model in GAMS involves several stages. It is not enough to 'define' the equations. Before the equations can be defined it is necessary to declare the required sets and assign members to each set; declare and assign values to the parameters that will be used in the model; declare and assign types to the variables used in the model; and then declare the equations. Only then will defining the equations have meaning in GAMS. The model and solve statements can then be included to complete the first stage of the modelling process. The process is the same for any model in GAMS, e.g., the simple liner programming transport problem. GAMS greatly simplifies the burdens involved.

The basic principle is to formulate a model that, when solved, can replicate the database/SAM from which the parameters were calibrated and for which the variables are pertinent. This is a process whereby a series of behavioural functions are defined and the parameters that control how those functions operate are specified and a series of exogenous constraints – parameters – are specified. Given this information the model will determine the values for the variables such that they are the same as those used to calibrate the parameters.

Thereafter it is possible to start conducting experiments that ask what the economy would 'look like' in a new equilibrium if one, or more, of the parameters were caused to change. Since it is reasonable to presume, as a guiding principle, that errors **will** be made, it is therefore desirable to learn how to correct mistakes and to include several checks in the process of calibrating the model to ensure errors are more easily detected and rectified.

It is often not sensible to start building a model from scratch. Indeed, the model library included with the GAMS programming language presumes that modelers will often start, or borrow, from a previous model and adapt it to their needs. That is the principle adopted here; each of the models in this series involves sequential extensions to the previous model so that at each stage the complexity of the model increases. But rather than starting with a blank sheet of paper or a model that already runs the starting point is a part completed body of code.

2. Model 1: A First CGE Model in GAMS

In this exercise the objective is to define the equations using a series of parameters, variables and equation declarations that are supplied in a template. The template (`clmod1.gms`), which contains everything required to run the model except for the 'Equation Definitions' and the 'Model Closure' rules, is contained in a User Library along with all the material for the models in this course.

All courses by cgemod provide the computer codes used for the exercises in User Libraries. This is an efficient, simple and robust method but it requires the user to adopt some standard actions. First, you should ONLY access files in the User Libraries from GAMS Studio. Second, you should NEVER use the User Library directory as a working directory. And third, you MUST follow the method detailed for accessing the pre course exercises in 'Introduction to GAMS and GAMS Studio' (http://www.cgemod.org.uk/gams_studio.html). The method is detailed in section 11 of 'An Introduction to GAMS with GAMS Studio' (http://www.cgemod.org.uk/Intro%20to%20GAMS%20Studio.pdf); we suggest you reread this section.

File management is critical when programming. In the days before Windows and macOS computer users needed to ensure that files were stored in the correct directories. Modern computer operating systems are more casual and place less emphasis on file management until users start to programme, at which point they need to unlearn bad habits.

We employ a User Library because it allows us to ensure you see the correct files when you need them for the exercises you conduct; there are 90+ files for the modules in this course, so it is easy to make mistakes. The User Library makes it easier and quicker for you; if you are working with the wrong files, you will get confused and waste time. It also means that if/when you corrupt the User Library it can be easily reestablished.

RESIST THE TEMPTATION TO EXPLORE THE CONTENTS OF THE LIBRARY; THIS WAY LIES CONFUSION. YOU GET TO SEE AND USE ALL THE FILES.

It is our experience that most problems experienced by participants with a User Library originate from not completing the pre course exercises 'Introduction to GAMS and GAMS Studio'. Problems then arise when participants try to access library files using Windows

Explorer, and/or decide that it will be quicker to copy a file, often the wrong one, from the User Library directory, and/or fail to create new sub directories when the exercise instructions say. We cannot solve problems that corrupt the User Library; all we can do is tell you to start again. So being organised and keeping your workings for each module separate, i.e., in their own directories/folders, ensures that errors are contained.

Configuring GAMS

There are different ways different users like to work, hence there are different 'appropriate' ways to configure GAMS Studio. For all cgemod courses we use a standard configuration of GAMS Studio. This section sets out the standard configuration used by cgemod.

With settings (F7) open select the General tab and choose the following options (Figure 2.1)

1. 'Open file in current project by default' (this means that you want to open a new file in a new project it is a conscious decision).
2. Set 'File count to generate GSP file' to 2 (this means GSP files appear in Windows Explorer as soon as programme is run)
3. Set 'GSP file needs main file' to OFF (this means a a GSP file can exist without a GMS file)
4. Set the 'Open *.lst file after running GAMS' to ON.

With settings (`F7`) open select the Editor & Log tab and choose the following options (Figure 2.2)

1. Set the Font to Courier New (a standard fixed pitch font)
2. Set the Font Size to 12 (the choice here is personal, but we find 12 works for most users)

With settings (`F7`) open select the Workspace tab and choose the following options (Figure 2.3)

1. Set the Default Workspace to `C:\cgemod\error` (this means that if you make errors in loading files, they are likely to end up in the directory `C:\cgemod\error`
2. Set 'Clean-up selected workspace directories on start-up' to OFF (this reduces the risk of deleting files that you prefer keeping)

Figure 2.1 Settings: General tab

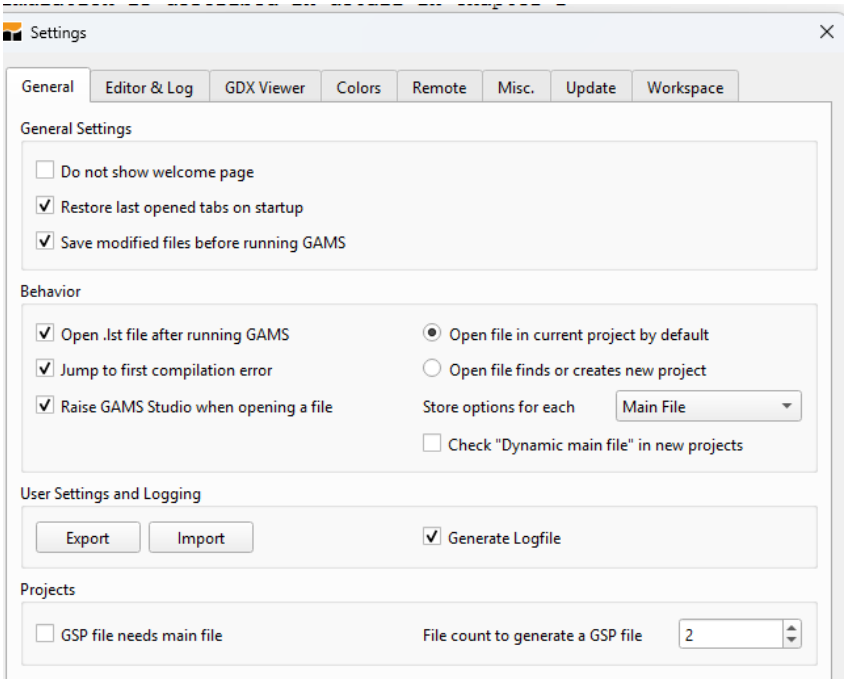


Figure 2.2 Settings: Editor & Log tab

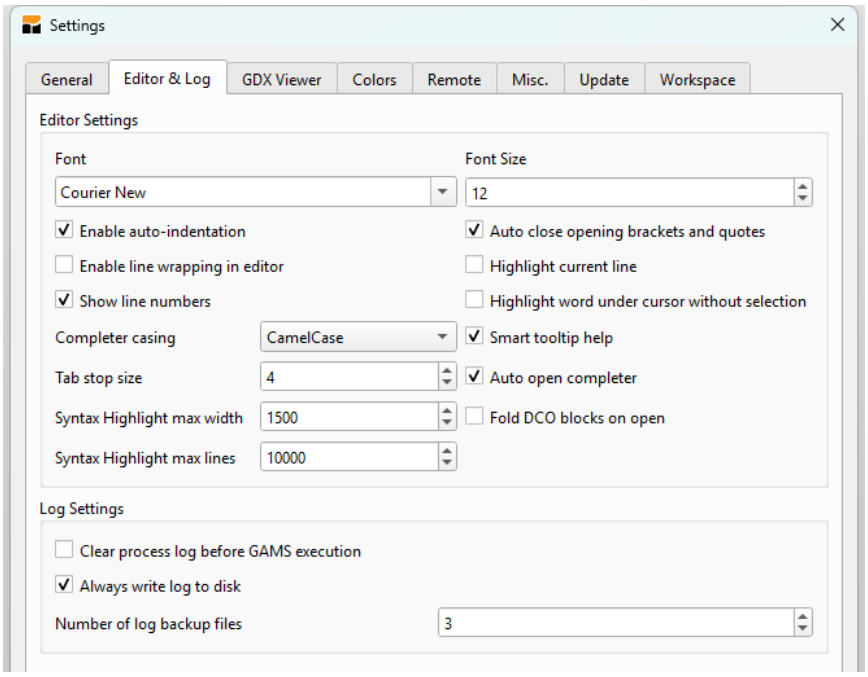
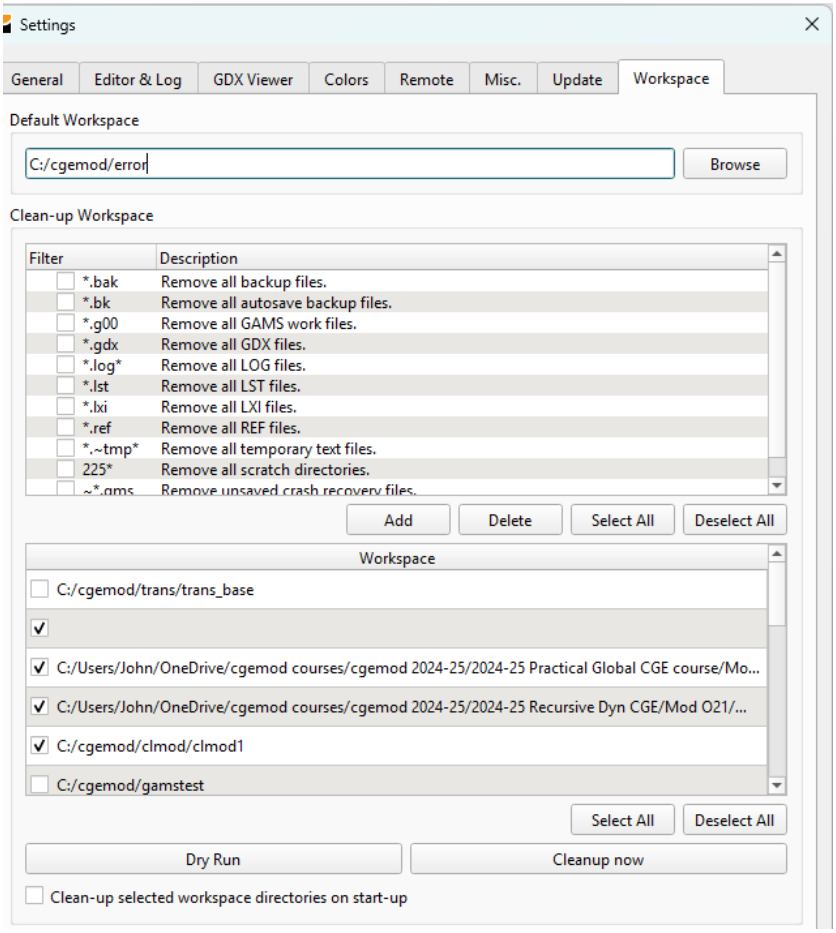
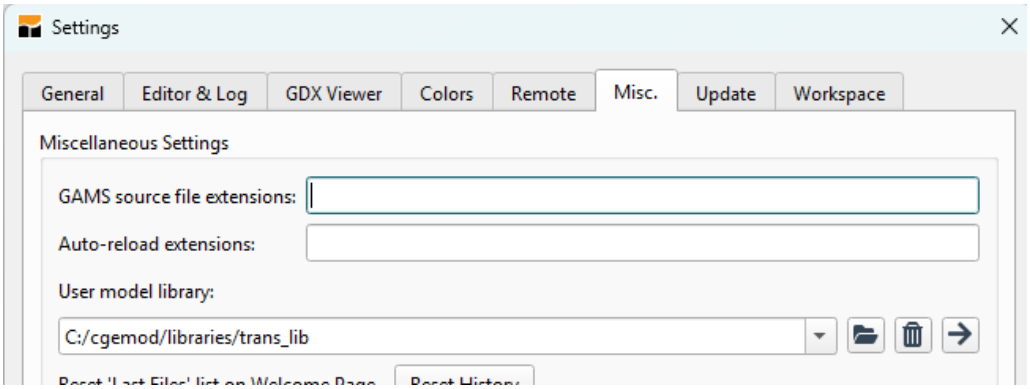


Figure 2.3 Settings: Workspace tab



With settings (`F7`) open select the Misc tab. This tab will need updating as you add more User Model libraries with different cgemod course. In Figure 2.4 it is set for `trans_lib`.

Figure 2.4 Settings: Misc tab



Establishing the User Library

Now establish the User Library that will be used for the modules of the Practical CGE course and tell GAMS Studio where the User Library will be found (see section 11 of 'Introduction to GAMS and GAMS Studio'). Carry out the following actions

1. Create a master directory for all your course materials, we will assume the directory is `C:\cgemod`
2. Create the following sub directories: `C:\cgemod\downloads` (to preserve all codes downloaded for courses), `C:\cgemod\cgemod_lib` (to contain all the User Model Libraries used by for courses), and `C:\cgemod\cgemod_lib\pract_lib` (the User Model Libraries for this course).
3. Then download the zip file `pract_lib.zip` from the website (it is one of the files in the folder of downloads for Topic P1:2) into your downloads directory (`C:\cgemod\downloads`).
4. Unzip the contents of the file `pract_lib.zip` into the library directory `C:\cgemod\cgemod_lib\pract_lib`. WinZip may by default unzip these files into a directory, often called `practical`. If so, you will need to copy these files and paste them into your library directory `C:\cgemod\cgemod_lib\pract_lib`. The files in this directory must NOT be in sub directories.[1]
5. Open Studio settings - `File>Settings` or `F7` or the `Settings Wheel` on the toolbar and go to the `Misc` tab.
6. Change the path for the User Model library from something like `C:\Users\…\Documents\GAMS\modellibs` (the exact path will depend on the settings on your PC) to `C:\cgemod\cgemod_lib\pract_lib`.
7. Close the Studio `Settings` window; remember to click `Apply` and `OK` at the bottom of the `Misc` tab.

[1] The default settings in WinZip may make you do this as a two-stage process. In WinZip 'classic view' you can avoid the two-step process. It is called progress!!!!

Working Directories

The practice we advocate and assume is followed by those taking cgemod courses and in all courses, involves creating a series of directories and sub directories that are used to keep distinct parts of the course separate. Basically, we create a master directory (`C:\cgemod`) that contains a series of sub directories for each course, e.g., `C:\cgemod\pract`, that contain sub directories for the various components of each course. We also create a sub directory for User libraries (`C:\cgemod\cgemod_lib`) that also has sub directories for each course and a sub directory to preserve the (zipped) library files that have been downloaded. We also have a sub directory to preserve all codes downloaded for courses `C:\cgemod\downloads`.

So, the next stage is to establish a sub directory for the closed economy models and then create a first NEW project for the exercises.

1. in Windows Explorer create the directory for this course, i.e., `C:\cgemod\pract`,
2. in Windows Explorer create the directory for these exercises, i.e., `C:\cgemod\pract\clmod\clmod1`,
3. open the Settings (`F7`)
4. on the General tab make sure that the option '`Open file in current project by default`' is selected (this should already be the case),
5. choose `File>New Project` and this opens the project options in the edit viewer,
6. the default name for the project is `newProject`,
7. rename the project name as `clmod1` (the name in Project Explorer will not change until the project is saved),
8. set the Working directory to the new project directory, e.g., `C:\cgemod\pract\clmod\clmod1`, using the BROWSE button,
9. note the Base directory will default to the same name as the Working directory,
10. choose `File>Save` – **DO NOT USE SAVE AS**[2]
11. now choose `File>New` (see below) that will open a Windows Explorer window with defaults File name, `new***.gms`. (this step together with step 3 above is necessary to ensure that Studio is directed to use the new project's directory),

[2] As of Studio version 1.22.2 using Save As produces an error.

12. click `SAVE` and `new**.gms` will appear in the `clmod1` project in the Project Explorer.

You now have a New Project that is effectively empty. This method will be used repeatedly although he instructions to create a New Project will be truncated.

Populating the Working Directory

Once a project specific working directory has been created it can be populated from the User Model Library.

1. In Studio press `F6` and in the Model Library Explorer select the `Practical CGE Library` and then select the library file `clmod1` and choose `Load` (or double click of the name), which is SeqNr: 1. (Figure 2.5.)
2. The `clmod1.gms` model will now be displayed in the editor window and be listed in the Project Explorer as being in the project `clmod1`. (Figure 2.6).
3. If you right-click on the project name and select `Open Location`, you will see that 4 files have been downloaded to the directory `C:\cgemod\pract\clmod\clmod1`. (Figure 2.7).
4. Studio can now be used to work with the closed economy model.

This is how each new project can be created from files in a User Model Library. It also demonstrates how multiple files can be provided from a single library entry. This method is used through all the courses offered by cgemod. For subsequent exercises the details for setting up a New Project will be abbreviated.

Figure 2.5 Practical CGE (User) Library

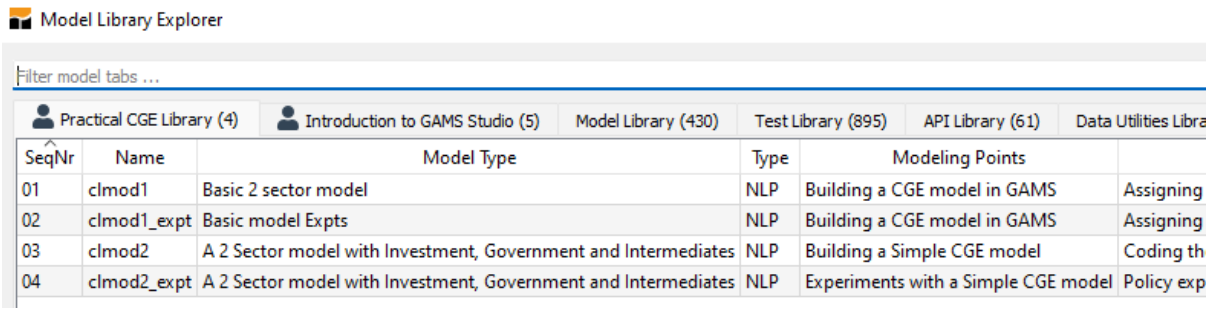


Figure 2.6 A First Model in GAMS Studio

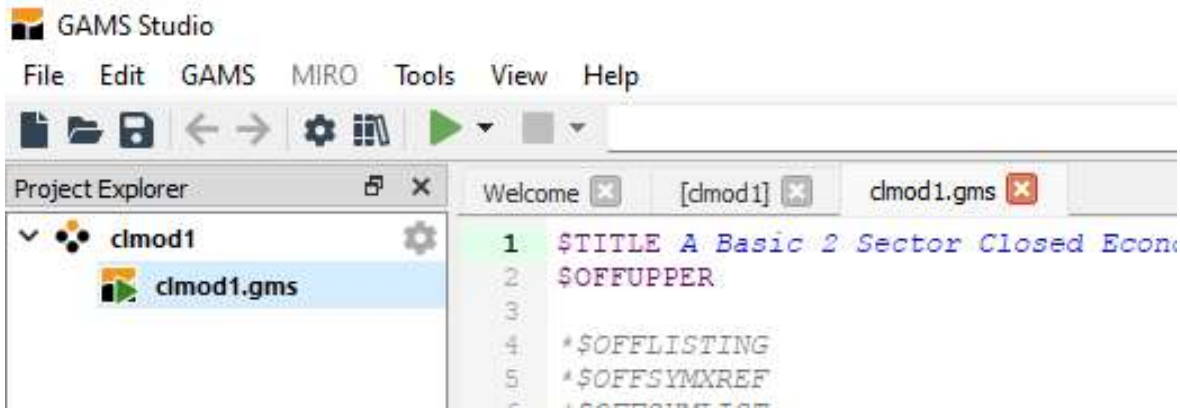
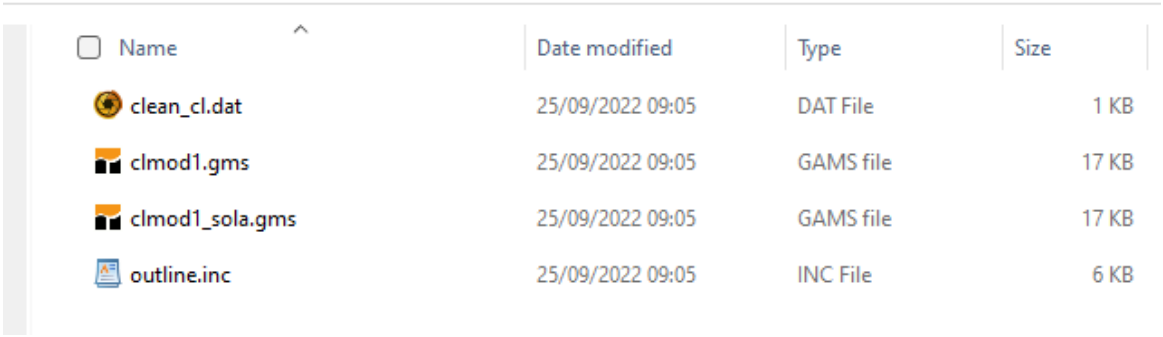


Figure 2.7 Contents of `clmod1` Directory



Having established the working directory and acquired the requisite files from the User Library it is possible to begin working on the first set of exercises with the basic closed economy model.

Exercise 1 Adding Model Equations

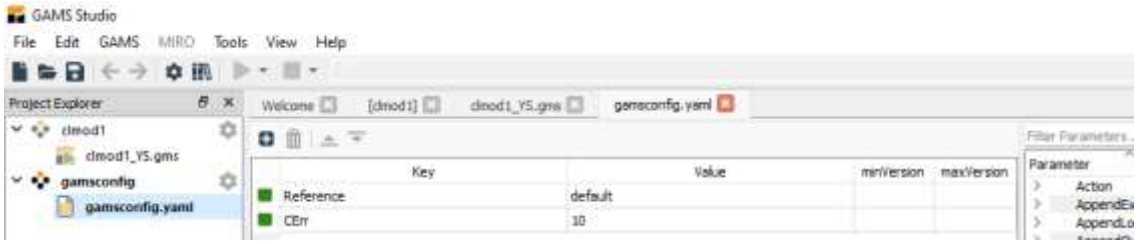
1. . Save the model code as `clmod1_**.gms`; where ** are wild cards, e.g., your initials, and close the file `clmod1.gms`. (This preserves the files in case you make errors that require you to start again from the template.
2. Print off a copy of the template `clmod1_**.gms`.
3. Read through the template and identify how and where the SETS, PARAMETERS and VARIABLES are declared and assigned, how and where the data are entered, the names used (NB: a full listing of parameter and variable names is included).

NB: The programme file '`clmod1.gms`' contains many sections that are unused at this stage. This is deliberate. The template forms the basis for other 2*2*2*2 models. The intention is that the final model from this initial exercise is the starting point for the next exercise and so on; therefore, it is useful if the template contains sections that will not be used in the basic/simple model.

The missing equations should appear in sections 14 and 15 of the programme file. The equations have already been declared in section 13.

Using the algebraic expression of the model's equations given in the technical document, "A Basic Closed 2 Sector CGE Model" available as a download from the Moodle site in Module O1, and the notation specified/implied in the template, compose the missing equations. When completed, section 14 should contain the 24 equations, which are written in 12 lines of code by using set notation. Section 15 should contain the 3 model closure conditions, which can be written as 2 lines of code.

Before first running the model clmod1_**.gms ensure that GAMS Studio is configured to automatically generate a reference file and stop after a specified number of errors (circa less than 10 and more than 4) have been encountered. This requires that the Default GAMS Configuration has been customised; details about adjusting the Default GAMS Configuration are available as section 10 in the 'Intro to GAMS Studio.pdf' that can be downloaded from https://www.cgemod.org.uk/Intro to GAMS Studio.pdf . The `gamsconfig` is shown below.

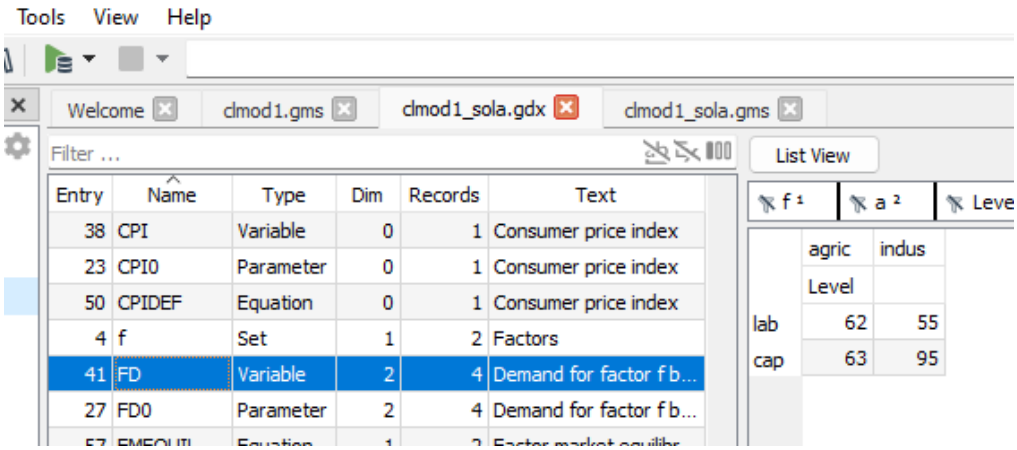


The default setting for the reference file means that the reference file produced will always be `[modelname].ref`.

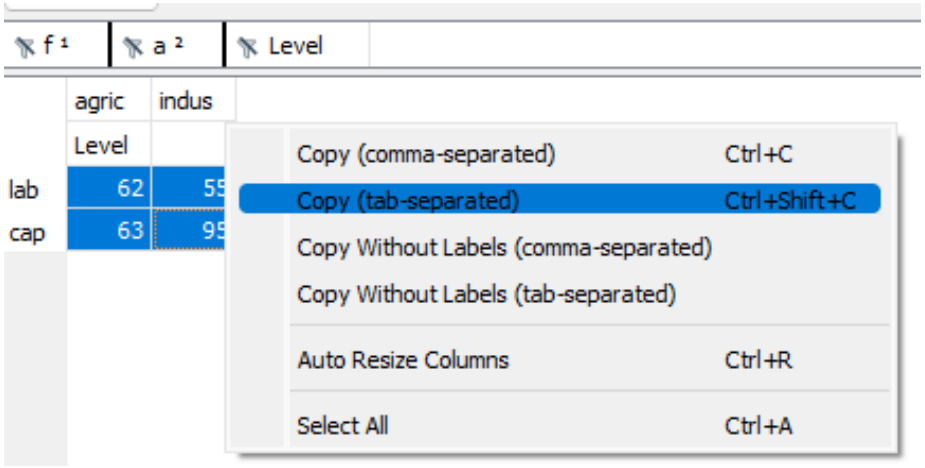
Now run the programme with GDX creation (`F10`). The GDX file created will be labelled `[modelname].gdx`. The resultant GDX file will contain all the information used by the model; sets, parameters, variables, equations, etc.: this is very useful when coding, debugging, and running a model. One of its other advantages is that it removes the need to add display statements to the code.

If you choose only to compile the programme use `Shift + F10`. If you compile rather than run the programme you may be able to resolve syntax errors before having to confront any execution errors.

Once you have compiled or run the programme for the first time open the file `clmod1_**.gdx` and review the results within GAMS Studio.



You can export results to Excel using Select All (Ctrl+A) copy using `Copy (tab-separated)` (`Ctrl+Shift+C`). There are other ways in GAMS to output GDX data to other programmes; typically, these involve using the GDX utilities in more sophisticated ways; some of those ways are explored in various courses offered by cgemod.



You will make mistakes. This is NOT a problem. In fact, it will allow you to gain experience in using the output from GAMS to identify and rectify errors in your GAMS programme. Experience solving problems at an early stage with a simple bit of programming

will help you find errors when you undertake more complex programming activities. Some simple guidance on compilation and execution errors is given below.

You may get a lot of syntax errors, i.e., red text, when first compiling/running the model; this is normal. A simple error early in the code can cause lots of subsequent errors hence the advice to ser `CErr` to a single digit number). Deal with each error in turn, and only try to solve a few errors between each time the model is compiled/run.

Compilation Errors

Compilation errors are generally syntax errors although they can also involve errors with the reading of data etc. In the process window syntax errors will appear in red with error messages attached; double clicking on the red error message will take you to the place in the *.gms file to which the error message refers – note this can be quite distant from the cause. Shift clicking on the error message takes you to the relevant place in the *.lst file: the listing file will provide useful suggestions about the likely cause of the error message. The listing file also contains markers that make it easy to find the compilation errors: search for the text string "****". **NB**: "***" marks something different so make sure you set the search up correctly.

Some general principles may be helpful

- start from the top of the programme and work down;
- solve each error as it appears - do not skip onto the next error without a good programming reason;
- do not make too many changes at a time - as you become more familiar with GAMS, and compilation errors, the number of errors corrected at each stage will increase, but when starting it is easy to compound errors (NB an error in the programme often results in subsequent 'errors' that are only defined as errors because of a previous error);
- if substantial changes are made to the code save the input file with a revised name before running the file (GAMSIDE should do this automatically but some of us are risk averse).

Common syntax errors include the following

- failing to end an operation with a ";" - GAMS often identifies this type of error as occurring on the next line of code or at the next keyword;
- assigning a parameter or variable before it has been declared;
- using a parameter or variable before it has been assigned;
- spelling/typing mistakes;
- not including a double period ".." after the equation name when assigning the equations;
- the "*" used for comments and/or to comment out lines of code is NOT the first item in the line of code (NB: a space is an item in a line of code);
- a set controlled operation trying to use sets that are already under control - this is where the aliases become very useful.

The solutions to most syntax problems are relatively simple. The more practice you get the better you will get a resolving these errors. Controlling sets is one type of syntax error that can prove somewhat less tractable.

Execution Errors

Execution errors are trickier. They can often arise because the model has been incorrectly specified. There are three basic consistency checks that will be used in this suite of models and that can be used to eliminate common execution errors:

i) The *WALRAS* variable should return a zero value - if it does not it indicates an error or errors in the system of equations.

ii) The model solution should return a consistent and balanced SAM, which is identical to the SAM database. Identifying the rows and columns that either do not balance or return values different to those in the database will help to identify the equations containing errors.

iii) Since the model should be homogeneous of degree zero in prices it should solve for relative prices. Hence if all prices increase by some constant factor the real variables should remain unchanged. If the model is not homogeneous of degree zero in prices the real variables will be changed. If the initial model solution does not return unit prices it indicates a problem. The simplest way to run this check is to double the *numéraire*.

There are some ways to help find execution errors.

i) The `[modelname].gdx` file will report the values for all the parameters, variables and sets. The values of the parameters and initial values for variables can be used to check the values returned by the programme.

ii) The listing of values for variables from the model should be consistent with the values from the database, i.e., the initial values.

iii) If the model is consistent the left-hand and right-hand sides of the equations should equate, or at least contain no 'significant' discrepancies. This can be checked in the equation listing (controlled by the "limrow" and "limcol" options linked to the solve statement). Search first for the string "LHS" in the *.lst file, or use the index associated with the lst file to go to the 'Equation' block. This will move you to the start of the equation listing. 'Significant' discrepancies are marked with ***; therefore, now search for the string "***". A discrepancy is indicated by the statement "LHS = <value>"; where the error is only likely to be important if 'value' is greater than about 1.0E-7. This indicates that the problem is probably associated with the definitions of the parameters and variables in that equation.

As with compilation errors it is unwise to try and do too much at once. If substantial changes are made to the code, it is wise to do so in a new version of the *.gms file.

A full working example of this simple model is provided in `clmod1_sola.gms` (this will have been downloaded to your directory `clmod1`). You should try to solve the programming problem for yourself and only refer to this example as a last resort.

NB: If you open `clmod1_sola.gms` in the same project you will need to change the main file. Right click on `clmod1_sola.gms` and select `Set as Main File.`

A manual comparison between the code/equations you wrote and our worked example (clmod1_sola.gms) is relatively easy at this stage. But there are a much more efficient options although these require using programmes other than GAMS Studio; details about some alternative programmes for comparing files are available as section 12 in the 'Intro to GAMS Studio.pdf' that can be downloaded from https://www.cgemod.org.uk/Intro to GAMS Studio.pdf . Our preference is WinMerge, which is open source and provides a method for highlighting differences between files. Access to such a utility is valuable.

Before running policy experiments there are a couple of exercises that will be useful later. (You can do these exercises in conjunction with the running of policy experiments). Use the Help menu to help you understand what things mean.

Dollar ($) Control Directives

Investigate the effects of switching on and off the $OFFLISTING, $OFFSYMXREF and $OFFSYMLIST directives.

1. What does each do?
2. When might they be useful?

SOLVE Statement OPTIONS

1. What value should 'limrow' have to ensure that all the equations are detailed in the Equation Listing? Where in the *.lst file will you find details about the number of equations, variables etc., in the model?
2. Reduce 'iterlim' until the model fails to solve? Where in the *.lst file will you find the number of iterations the model needs to reach a solution?
3. Experiment with solving the model using different solvers. Does the choice of solver make any difference to the solutions?

3. Model 1: Policy Experiments

Getting a working model is only the first stage of the process; much more interesting is using the model to carry out policy experiments. Policy experiments can be conducted in several different ways. The fundamental principle in each is the same. The CGE model is set up to replicate the economy in the base year, i.e., the model is calibrated so that the solution reproduces the database/SAM for the year to which the model is benchmarked. Imposing a series of exogenous shocks (to parameter values) and solving the model for each shock carries out the policy experiments. Multiple policy experiments can be conducted during a single run of the model by using multiple solve statements. The values of variables, and of course parameters, at the end of each solution cycle are those at the start of the next cycle.

Hence carrying out policy experiments involves three stages. First, the definition of the exogenous shocks; second, a series of solve statements; and third, the specification of some method of capturing the results at the end of each cycle, i.e., after each solve statement. GAMS, as standard, provides a method, albeit crude, for achieving the third stage so it is natural to start by concentrating on the first two stages.

A feature of a GAMS programme, and of most computer languages, is that variables and parameters retain their values until they are overwritten. Thus, if the value of a parameter is changed and then in a subsequent experiment the original value is required the user must reset the parameter value. For variables it means that the solution values for variables (*.*L*) after one solve statement are the starting points for the variables for the next solve statement.[3]

Simple CGE Experiments

These experiments will focus on the impacts of changing the quantities of different factors upon the optimal solution. Technologies, preferences, and the patterns of factor ownership (endowments) will be held constant, and the experiments will consider four cases in which the total quantities of labour and capital changes:

- a 25 percent increase in total capital stocks;
- a 25 percent increase in total labour stocks;
- a 25 percent increase in both total labour and capital stocks; and

[3] In some large and complex models this can cause problems if the model needs to make very large changes to move from one solution to another. You will not have this problem on this course.

- a 10 percent increase in total capital stocks and 25 percent increase in total labour stocks.

Notice that demands for factors by activities ($FD_{f,a}$) are variables but that the system is constrained by the supply of factors (FS_f). Although *FS* are declared as variables the model closure section states that `FS.FX(f) = FS0(f)`, which means that the variable attribute for *FS* is set so that *FS* is fixed, i.e., the values for *FS* act as parameters (see the GAMS documentation for details on variable attributes). Thus, the factor supplies can be shocked since they are classified as parameters.

These experiments will be used to both develop an understanding of implementing experiments and develop GAMS coding skills (the economics is also interesting).

These experiments will start from the sample solution to the model coding exercise, i.e., `clmod1_sola.gms`., that is in the directory `clmod1`. However, we wish to carry out these exercises in their own project. So, choose `File>Open in a New Project` (IF you have followed previous instructions the keyboard short cut is `Ctrl+Shift+O`) and select the file `clmod1_sola.gms`; this should open in a new project. Rename the file as `clmod1_expt1.gms`. We are starting from the sample solution, rather than your solution, which should be identical, to ensure that you all start from identical code for the model and to make it easier to provide support.

Experiment 1

It is now time to run a simple first experiment. The first experiment involves increasing the quantity of capital available to the system by 25 percent.

In section 18, the Policy Experiments section, add the following code

```
*################## 18. POLICY EXPERIMENTS ##################
* a 25 percent increase in total capital stocks;
 FS.FX("cap") = FS0("cap") * 1.25 ;
Display FS.UP ;
 Solve clmod1 Using NLP MAXIMIZING GDP ;
*################## END of EXPERIMENTS ######################
```

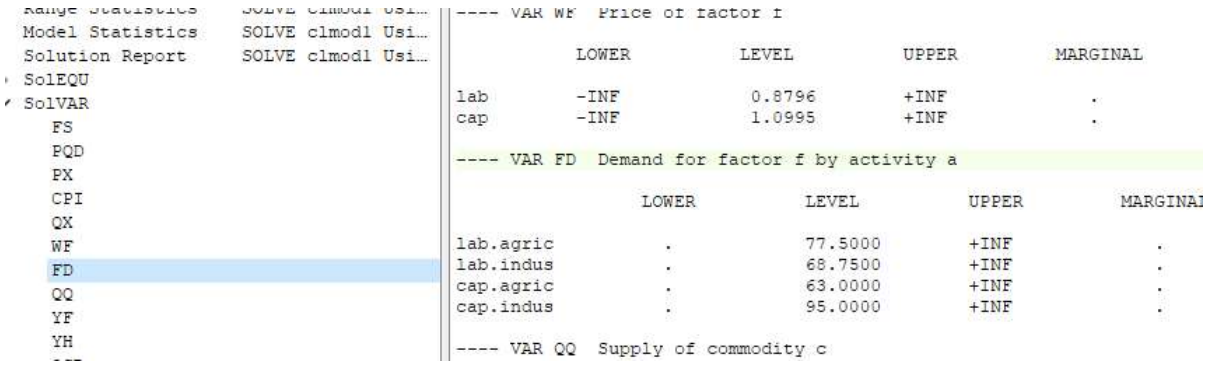
Notice several things about this block of code:

1. The inclusion of short note of explanation about the intention with the shock. The actual shock can be determined from the code, but the actual shock may not be the intended shock!!
2. The value for *FS*0 is not changed; thus, the information about the initial factor supplies is retained.
3. A Display statement is included; this allows the user to check that the intended changes to parameter values have been imposed before the shock is implemented and subsequently to check that shock has been implemented. In this case – a variable operating as a parameter - it is necessary to define the attributes of the variable, or an error will be generated. (The attribute `*.UP` has been used; could other attributes be used and achieve the same results?) This is one of those situations where a Display statement may be useful even though the same information will be reported in the GDX file.
4. A Solve statement is included; this reruns the model using all the currently binding parameter values.

Run the model with GDX creation (F10) and view the file `clmod1_expt1.gdx`.. Notice how the *.lst view is in two parts; an index on the left-hand side (the *.lxi file) and the listing on the right-hand side (the *.lst file). The solution values for the variables after each solve statement are indexed under SolVAR; clicking on the + sign to the left of SolVAR makes a list of the model variables visible. The user can then double click on the variable of interest and the list file will move to the chosen section (see the illustration).

NOTICE that there two extra sets of solution values (SolEQU and SolVAR); one for each Solve statement. The first is the replication of the base the second has the results for the experiment.

Figure 3.1 SolVar for FD in lst View



Use the information reported for the variables to report on the following:

1. The production of each commodity (*QX*).
2. The quantities of each factor used by each activity (*FD*).
3. The quantitative of each commodity consumed by each household (*QCD*).
4. The prices (*PX* and *PQD*) for each activity and commodity.
5. The factor prices (*WF*) for each factor.
6. The sources of income to each household (*hvash* and *YF*).

Compile this information into an Excel spreadsheet; this spreadsheet will also be used to compile the information from the next 3 experiments so it is important to plan how the results will be laid out to assist making comparisons between the results from each experiment. (The organisation of data and results is very important when working with CGE models; typically, each person has their own ways of doing this so providing a template is not always helpful – furthermore you will often learn best by making errors.)

Can you find these values in the file `clmod1_expt1.gdx`. Why are these not the same as the values BEFORE the experiment?

Experiment 2

The second experiment involves increasing the quantity of labour available to the system by 25 percent.

In section 18, the Policy Experiments section, add the following code below the code added for the first experiment.

```
*################## 18. POLICY EXPERIMENTS ##################

* a 25 percent increase in total labour stocks;
```

```
 FS.FX("lab") = FS0("lab") * 1.25 ;

 FS.FX("cap") = FS0("cap") ;

Display FS.UP ;

 Solve clmod1 Using NLP MAXIMIZING GDP ;

*###################### END of EXPERIMENTS ############################
```

Use the information reported for the variables to report on the following:

1. The production of each commodity (*QX*).
2. The quantities of each factor used by each activity (*FD*).
3. The quantitative of each commodity consumed by each household (*QCD*).
4. The prices (*PX* and *PQD*) for each activity and commodity.
5. The factor prices (*WF*) for each factor.
6. The sources of income to each household (*hvash* and *YF*).

Add this information to the Excel spreadsheet developed for the previous experiment

Experiment 3

The third experiment involves increasing the quantities of labour and capital available to the system by 25 percent.

In section 18, the Policy Experiments section, add the following code below the code added for the first and second experiment.

```
*################## 18. POLICY EXPERIMENTS ##################
* a 25 percent increase in both total labour and capital stocks; and
 FS.FX(f) = FS0(f) * 1.25 ;
Display FS.UP ;
 Solve clmod1 Using NLP MAXIMIZING GDP ;
*###################### END of EXPERIMENTS ############################
```

Use the information reported for the variables to report on the following:

1. The production of each commodity (*QX*).
2. The quantities of each factor used by each activity (*FD*).
3. The quantitative of each commodity consumed by each household (*QCD*).
4. The prices (*PX* and *PQD*) for each activity and commodity.
5. The factor prices (*WF*) for each factor.

6. The sources of income to each household (*hvash* and *YF*).

Add this information to the Excel spreadsheet developed for the previous experiment

1. Compare the results from the three Experiments using tables **AND** graphs.
2. Using the results from the previous three experiments consider the following questions:
    - Are the activities producing on their production possibility frontiers?
    - Is welfare maximised?
    - How do changes in the RELATIVE quantities of factors impact on the equilibrium outputs and prices and consumption quantities and prices?
    - How do changes in the ABSOLUTE quantities of factors impact of the equilibrium outputs and prices and consumption quantities and prices?
    - Are the results consistent with the simple theory of general equilibrium found in intermediate microeconomic textbooks?
    - Explain why the outputs in experiment 3 increased by 25% for both activities?
    - Explain why the consumption of both commodities increased by 25% for both households?

A worked example of experiments 1, 2 and 3 is provided in the file `clmod1_sola_expt1.gms`. This file, and the other examples of experiment files, are in the User Model Library. Choose `GAMS > Model Library Explorer (or F6)` and in the `Practical CGE s`elect the second item in the library – `clmod1_expt`; this will download all the worked examples to your working directory.

RESIST TEMPTATION TO LOOK AT THE EXAMPLES BEFORE DOING THE EXERCISES.

4. Collecting Results

Since the results of interest for each experiment are the variables, for which the values are updated after each solve statement, capturing and consolidating the results requires converting the values of the variables into parameters after each solve cycle. To do this it is necessary to declare a series of parameters that will be later assigned with the values of the solution variables. However, since the values of the variables should change after each solve statement it is necessary to declare a separate set of parameters for each solution statement. One parsimonious way to do this is to exploit the fact that parameters (and variables) in GAMS can have multiple dimensions. Thus, we can declare a parameter for each variable with one more dimension than the variable, e.g., the variable for household consumption is `QCD(c,h)` and a parameter for the levels values of results could be `levQCD(c,h,sim)`, where '`sim`' is a set with one member for each policy experiment, and then assign the values of the variables after each solve statement to the appropriate element in the results parameter. Note also that it will be necessary to declare and assign the set `sim`.

These experiments will start with the sample solution to the model coding exercise, i.e., `clmod1_sola_expt1.gms`., that is in the directory `clmod1`. However, we wish to carry out these exercises in their own project. So, choose `File>Open in a New Project` (IF you have followed previous instructions the keyboard short cut is `Ctrl+Shift+O`) and select the file `clmod1_sola_expt1.gms.`; this should open in a new project. Rename the file as `clmod1_expt2.gms`. This will preserve your previous work. We are starting from the sample solution, rather than your solution, which should be identical, to ensure that you all start with identical code for the model and to make it easier for you.

Before running the simulations, it is necessary to declare and assign the set *sim* and declare the parameters for collating the results. To do this insert the following text before the first experiment – this can be done nearly anywhere in the *.gms file before the experiments, but it is sensible to include it just before the experiments. This exercise will use the same three experiments as previously setup.

```
*~~~~~~~~~~~~~~~~~~ SET UP RESULT PARAMETERS ~~~~~~~~~~~~~~~~~~~~~~~~~~~

$ontext

 In order to consolidate the results from all the simulations it is necessary

 to declare additional parameters then assign to them the results from
```

```
 the simulations
 Note these parameters will need values assigned for each experiment/simulation
 THUS we declare a set - SIM - that will have one member for each experiment,
 and declare the results parameters with one more dimension - sim.
 Then after each solution the results parameters are updated.
$offtext
Set
 sim     simulations /
 sim01     25% increase in total capital stocks
 sim02     25% increase in total labour stocks
 sim03     25% increase in both total labour and capital stocks
  /
  ;
Parameter
 levQCD(c,h,sim) Level results Household consumption by commodity c
 levFD(f,a,sim)  Level results Demand for factor f by activity a
 levFS(f,sim)    Level results Supply of factor f (check on experiments)
 levGDP(sim)   Level results GDP from expenditure
 levPQD(c,sim)   Level results Price of domestic sales by commodity c
 levPX(a,sim)    Level results Price of domestic production by activity a
 levQQ(c,sim)    Level results Supply of commodity c
 levWALRAS(sim)  Level results Slack variable for Walras's Law
 levWF(f,sim)    Level results Price of factor f
 levQX(a,sim)    Level results Domestic production by activity a
 levYF(f,sim)    Level results Income to factor f
 levYH(h,sim)    Level results Income to household h
  ;
*~~~~~~~~~~~~~~~~~~ SIMULATIONS ~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

To assign the values of the variables to the results parameters it is necessary to include assignment statements between each solve statement. Thus, after the first simulation/experiment the following code needs adding

```
* Assigning variable values to results parameter

 levQCD(c,h,"sim01")    = QCD.L(c,h) ;

 levFD(f,a,"sim01")     = FD.L(f,a) ;

 levFS(f,"sim01")       = FS.L(f) ;

 levGDP ("sim01")     = GDP.L ;

 levPQD(c,"sim01")      = PQD.L(c) ;

 levPX(a,"sim01")       = PX.L(a) ;

 levQQ(c,"sim01")       = QQ.L(c) ;

 levWALRAS("sim01")     = WALRAS.L ;

 levWF(f,"sim01")       = WF.L(f) ;

 levQX(a,"sim01")       = QX.L(a) ;

 levYF(f,"sim01")       = YF.L(f) ;

 levYH(h,"sim01")       = YH.L(h) ;
```

Note two important features of these assignment statements.

1. The specific elements in each parameter for each simulation are identified using a specified member of the set *sim*, which is included in quotation marks (double or single), in this case "`sim01`".
2. It is the levels value for each variable that is assigned to the results parameter, e.g., `QCD.L(c,h)`, where `.L` identifies the levels attribute of the variable.

These assignment statements need to be repeated after each solve statement – remembering to update the specified member of the set `sim`. If you use find (`Ctrl+F`) and replace and then select text to be replaced and the text used in the replacement, this can be done very quickly.

The results could then be displayed in GAMS by using the display facility, e.g.,

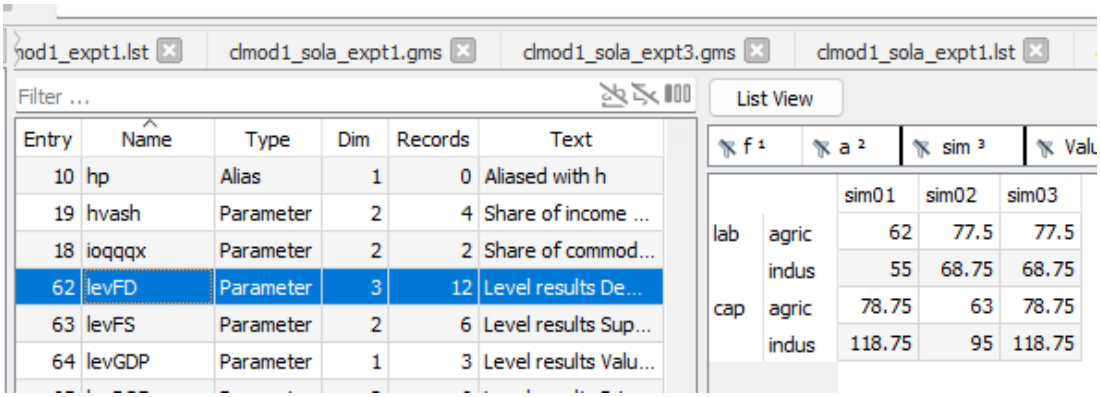
```
Display

 levQCD, levFD, levFS, levGDP, levPQD, levPX, levQQ, levWALRAS, levWF,
levQX, levYF, levYH  ;
```

This method will produce tables of results in the GAMS list file, which is certainly an easier way to access the results but it is still not an efficient method for transferring results between different environments, e.g., making GAMS results available in Excel format.

Assuming you have run the model with GDX creation all the results are stored in GDX format making the data available to other environments is relatively straightforward. We will make extensive use will be made of GDX.

The figure below illustrates the view of the parameter `levFD` in the file `[modelname].gdx`. You should experiment with the row and column assignments in the `Table View`.



It may be useful to export certain data to dedicated GDX files. Exporting data to GDX is straightforward and requires the use of instructions during the execution phase. The standard syntax is

```
Execute_unload 'filename.gdx',
id1 id2 id3… ;
```

Thus, to export all the results from this model to GDX the user adds the following command

```
Execute_unload 'results_clmod1.gdx',
 levQCD levFD levFS levGDP levPQD levPX levQQ levWALRAS levWF levQX levYF
levYH   ;
```

The resulting file can be viewed in GAMS Studio. Note that when choosing this option, it is important to make sure that all the data you want to view are exported to the designated GDX file, or you may have to have more than one GDX file open.

You should now be able to extract all the results from experiments 1 to 3 in short order. Do so and compare the results obtained this way with those obtained before.

Calculating Derived Results

It is often very useful to derive results other than the levels values of the variables, or other dimensions of the variables, e.g., the marginal values. Among the derived results that may be of use are summary macroeconomic totals, e.g., total production, income from employment, etc. These can often be computed as simple calculations based on the values of variables and parameters. The simplest, and most used, are the percentage changes in the values of the variables. Obviously once the results are in Excel, or some similar environment, such calculations are straightforward, but it is often the case that these calculations can be done more efficiently in GAMS.

Calculating and reporting percentage changes in the variables is straightforward. First, a series of parameters need to be declared to contain the computed percentage changes (see below).

```
*~~~~~~ Percentage Change Results

 percQCD(c,h,sim)   Percent change results Household consumption by commodity c

 percFD(f,a,sim)  Percent change results Demand for factor f by activity a
```

Second the percentage changes need to be assigned to the appropriate elements of the parameters (below it is done for `sim01`).

```
*~~~~~~ Percentage Change Results

 percQCD(c,h,"sim01")$QCD0(c,h) = (QCD.L(c,h)/QCD0(c,h)-1)*100 ;

 percFD(f,a,"sim01")$FD0(f,a)   = (FD.L(f,a)/FD0(f,a)-1)*100 ;
```

Third, the results need to be exported, which simply requires extending the `Execute_unload` statements.

HINT: note that $\left(\frac{QX_1 - QX_0}{QX_0}\right)*100 = \left(\frac{QX_1}{QX_0} - 1\right)*100$ and if $QX_0 = 0$ there will be a programming problem. i.e., a division by zero error.

The right-hand sides (RHS) of the assignment equations are straightforward percentage change calculations using the initial values of the variables as the reference points. But note how the RHSs contain a division term that risks the possibility of a division by zero, which will cause fatal errors. To guard against this a $ control is imposed on the LHS. In this

instance the $ control is easy to understand; it instructs GAMS to only implement the RHS equation ***if*** the term after the $ sign is not equal to zero, i.e., `$QCD0(c,h)` instructs GAMS not to implement the RHS if `QCD0(c,h)` is equal to zero.

The code from the previous stage should be extended to calculate percentage changes for all the model variables.

A worked example for `sim01`, `sim02`, and `sim03` is provided a `clmod1_sola_expt2.gms`.

5. Experiments as INCLUDE Files

Once a model has been developed it is common for the model to be used to conduct a range of different policy experiments. Sometimes all the experiments can be conducted in one run of the model, but there are occasions when groups of experiments in isolation from other experiments are useful. When doing this, it is very useful to be able to retain the code for each group of experiments, and since the core model code would be common in all cases, it would be inefficient to include that code with every set of experiments; inefficient not only in that it would use up more computer storage – scarcely an issue now – but more importantly in that changes to the core model may not be added to every version of the core model.

One way to be more efficient is to use `INCLUDE` files with each `INCLUDE` file containing a unique group of experiments. This is very simple to achieve. Open the file `clmod1_sola_expt2.gms` in a new project `(Ctrl+Shift+O)` and save as `clmod1_expt3.gms` and close `clmod1_sola_expt2.gms`. This should make `clmod1_expt3.gms` the Main File. Then copy all the code from the policy experiments section and paste it into a new GAMS document. Save the new document as `clmod1_expt3.inc` – ***NB*** the file type is `*.inc`. In the file `clmod1_expt3.gms`, delete all the policy experiment code from the core model file and add the following code to the Policy Experiment section.

```
*################## 18. POLICY EXPERIMENTS ##################
* Factor Supply Experiments
$INCLUDE clmod1_expt3.inc
*###################### END of EXPERIMENTS ################################
```

Make sure `clmod1_expt3.gms` is the Main File and run (F10)/

It is possible to add many such groups of experiments and to select which will operate by commenting in or out the relevant include statement (***NB*** a * as the first element in a line converts that line of code into documentation). While it is possible to run many include files in sequence it is important to note that GAMS *remembers* the last value assigned to parameters and variables so care must be taken to avoid unintentionally carrying over values from one experiment to another.

A worked example is provided in `clmod1_sola_expt3.gms`.

6. Basic Two Sector Closed Economy - Policy Analysis

This exercise is designed to help you develop your ability to analyse policy experiments. The objective is to make you think about formulating and analysing policy experiments.

Analysis

The President of Utopia is distraught about immigration because he is concerned about wages. At the same time, he encourages foreign direct investment, FDI, to expand the country's capital stock, promote labor productivity, and increase wages.

Using the results from the exercises for the Basic Two Sector Closed Economy Model, comment on the impact immigration and FDI will have on Utopia. Be sure to discuss the impact on output of each sector, GDP, factor returns and household incomes. Your report should not exceed 500 words.

Changes in factor endowments

Shocks:

1. 25% increase in the capital stock
2. 25% increase in the labour stock
3. 25% increase in both labour and capital

Miscellaneous Fixed Variables:

1. Fix the consumer price index (CPI) as numeraire.

Factor market clearing:

1. Labour and capital are fully employed and mobile between sectors.

Results

You should present your results as tables and/or graphs. It may be useful to include background information about the economy such as the share parameters for the Cobb-Douglas production functions, the share parameters for the Cobb-Douglas utility functions, the distribution of factor income to households, and capital labour ratios in production.

WARNING: GDX DOES NOT REPORT RESULTS THAT ARE ZERO. BE CAREFUL WITH THE PERCENTAGE CHANGE RESULTS.

Appendices

Additional experiments

Factor Endowments: Experiment 4

A fourth factor endowment experiment involves increasing the quantities of labour and capital available to the system by different percentages, e.g., a 10% increase in capital and a 25% increase in labour. Such an experiment will help you understand how changing capital labour ratios impact on an economy

In section 18, the Policy Experiments section, add the following code below the code added for the previous experiments.

```
*################## 18. POLICY EXPERIMENTS ##################
* a 10 percent increase in total capital stocks
* AND 25 percent increase in total labour stocks
 FS.FX("cap")  = FS0("cap") * 1.10 ;
 FS.FX("lab")  = FS0("lab") * 1.25 ;
Display FS.UP ;
 Solve clmod1 Using NLP MAXIMIZING GDP;
*###################### END of EXPERIMENTS ############################
```

Use the information reported for the variables to report on the following:

1. The production of each commodity (*QX*).
2. The quantities of each factor used by each activity (*FD*).
3. The quantitative of each commodity consumed by each household (*QCD*).
4. The prices (*PX* and *PQD*) for each activity and commodity.
5. The factor prices (*WF*) for each factor.
6. The sources of income to each household (*hvash* and *YF*).

No specific worked example of this experiment is provided.

Income distribution experiment

The issue of the implications of changes in the distribution of endowments is central to welfare economics. This raises the question of what would happen to production,

consumption, factor prices, commodity prices, GDP and welfare if the pattern of ownership of endowments was changed.

The pattern of ownership of endowments is recoded by the **parameter** matrix $hvash_{h,f}$. Thus, if experiments are to explore the implications of changes in the ownership of endowments are to be explored the experiments will involve changing the values in this matrix. This is easily done and is left for you to explore later.

HINT: Note that the shares of each factor owned by the two households must sum to one. If they do not sum exactly to one, the model will not produce an equilibrium solution. The easiest way to make sure this is achieved is to specify the share for one household of each factor and then specify the shares for the other household as the residual, i.e., one minus the specified shares.