Two Sector Closed Economy CGE Model: Exercises

cgemod

Contents

1. Introduction

It is rarely sensible to build a complex model from scratch. Indeed, it is usually good practice to get a simple version, e.g., a version that has simple behavioural relationships and an aggregated database, of a model running and then make sequential extensions. With each extension, the complexity of the model increases. This follows the simple principle that making one change, or one SMALL group of related changes, at a time is a method by which you can avoid making too many errors. It also has the even bigger advantage of ensuring you understand your own model!!

This exercise is one step along such a route. A very basic closed economy model, then a closed economy model with government and savings, then a very, very simple CGE trade model, then stick the bits together to get a simple single country model. Thereafter the size of the database can slowly increase, after you have learnt to handle the mole hill of data and results from the closed economy and simple trade models, you start to learn how to handle the Peak District (UK, look it up) sized hills of data and results of the simple single country model before facing the Grampian (UK, look it up) sized hills of data and results and later the Rocky Mountain (USA, look it up) sized hills of data and results, and so on until you get to the Himalayas. After that, and somewhere along the route, you may realise that bigger is not always better and it is time to make the model more sophisticated.

When you realise this, you can call yourself a modeler and not just a user of (other people's) models.

Remember, that errors **will** be made, so you need to keep learning how to identify and correct errors.

This set of exercises will require you to do more coding; the template this time does not include the new variable definitions and initialisations or the new equations and assignments. Moreover, some (many?) of the previous equation assignments will need to be changed.

GO SLOW. Use a check list and make sure you do all the steps in order. You will need to find your way but note that being random will rarely, if ever, work.

2. Model 2: Savings, Government and Intermediate Inputs in a Closed Economy Model

This exercise starts from a solution to the basic closed economy model, to which we have added a few things to ease you into the problem. This exercise requires updating of the SAM and sets, the declaration and assignment of the equations, the declaration and initialisation of the additional variables, a few changes to the calibration of some parameters plus a few new ones and extending of the macroeconomic closure and market clearing rules.

Setup

The first step is to get the correct files into a working directory. So, do the following

1. Open GAMS Studio and select `File>New Project`. This opens the `Import Project` window that is essentially a Windows Explorer Window.
2. Add a new (sub) directory in the directory `C:\cgemod_training\clmod`; by right-clicking in the left-hand panel and choose `New>Folder` from the menu.
3. Give the New Folder the name `clmod2` and SELECT this sub-directory.
4. The Studio panels should appear automatically once the new directory has been created with the working directory's path is reported as `C:\cgemod_training\clmod\clmod2`.
5. In Studio press F6 and in the Model Library Explorer select the `Practical CGE Library` and then select the library file `clmod2` and choose `Load` (or double click of the name), which is SeqNr: 3.
6. The `clmod2.gms` model will now be displayed in the editor window and be listed in the Project Explorer as being in the project `clmod2`.
7. If you right-click on the project name and select `Open Location`, you will see that 6 files have been downloaded to the directory `C:\cgemod_training\clmod\clmod2`. (Figure 2.5).
8. Studio can now be used to work with the closed economy model.
9. The file `clmod2`.gms will now be open in GAMS Studio; save this file as `clmod2_**.gms`; where ** are wild cards, e.g., your initials.

10. Review the model and note each of the subsections, including those with nothing in them – there are pre-defined spaces for all the code you will generate.

This is the first step. Do not move on until you have successful completed this sequence.

<u>Coding the Two Sector Model (clmod2)</u>

The code you need to write for this model requires that you use the technical document for this model to identify those equations that need to be added and those that need to be changed. The technical document details the equations in algebraic form; you will need to translate the equations into the GAMS form.

READ THROUGH THE REST OF THIS SECTION BEFORE DOING ANYTHING MORE.

We recommend that you proceed in the following way.

1. In the code: update the sets.
2. In the code: update the SAM data.
3. On paper: make a list of all the EXISTING equations that need to be modified.
4. On paper: make a list of all the NEW equations that need to be added.
5. On paper: make a list of all the VARIABLES that need to be added.
6. On paper: make a list of all the PARAMETERS that need to be modified.
7. On paper: make a list of all the PARAMETERS that need to be added.
8. In the code: declare all the NEW equations.
9. In the code: work with **ONE** equation at a time, start with the first one in the Section 14.
10. In the code: type in the NEW equation or revise an EXISTING equation.
11. In the code: declare and assign any NEW parameters and check the definitions for existing parameters for this equation. (The code for the GST is already in the template so you can use it to help you work out the code for the other taxes and the savings rates. Part of the code for IO coefficients is in the template, so you just need to complete this code.)
12. In the code: declare and initialize any NEW variables and check the initialization of any existing variables for this equation.

13. On paper: tick off all the items from your list that you have completed.
14. In the code: more on to the next equation.
15. In the code: continue until all the equations are completed.
16. In the code: add or modify the model closure conditions.

The code for the aggregate SAMs in sections 12 and 17 has been modified for you. **BUT** until all the parameters have been assigned and all the variables have been initialized this code will generate errors.

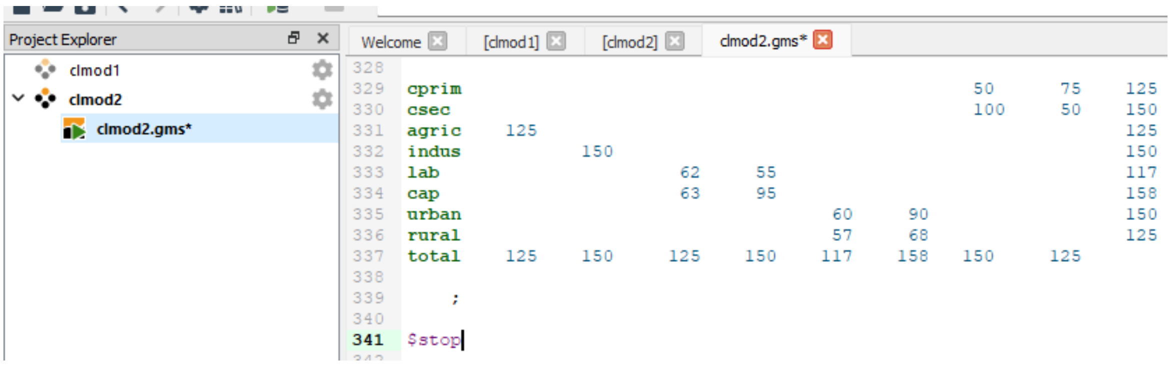
Sets and Data

Extend the sets for the model in section 6 of the template.

The SAM for model 2 is provided in the workbook '`clmod_SAM.xlsx`' in the worksheet '`SAM clmod2`'. These data need to replace the Social Accounting Matrix in Section 8 of the template. The data can be added by copying and pasting (and then ensuring that columns line up[1]).

Before going further, it is very wise to test the sets and the data are being read into GAMS correctly. This set of actions can save a very large amount of time later. Do the following:

1. Add the statement `$STOP` immediately before section 9 (Parameter Assignment); this will cause the program to stop at this point.
2. Run the model using F10 (its assumed that a reference file is generated by default).
3. Open the file `clmod2_**.gdx` and check that the sets (sac) are complete, and that the data (SAM) are complete.



```

cprim                                                50      75    125
csec                                                 100     50    150
agric    125                                                       125
indus           150                                                150
lab                   62     55                                    117
cap                   63     95                                    158
urban                               60     90                      150
rural                               57     68                      125
total    125    150    125    150    117    158    150     125

    ;

$stop
```

1 NB: entries in a column must be lined up underneath the column label while entries in a row must be in the same row as the label.

Once you are sure that the sets and data are correct remove the `$STOP` command and move on to the equations knowing that the data and sets have been loaded correctly. Later the same process can be used to make testing your coding easier.

Example for One Equation

The process of working with one equation is detailed below for the NEW equation TOTSAVEQ.

1. Declare the equation *TOTSAVEQ* in Section 13 (Equation Declaration); remember to provide a description.
2. Assign the equation *TOTSAVEQ* in Section 14 (Equations): note the 2 periods after the equation name.
3. The equation code identifies that 3 NEW variables (*TOTSAV*, *KAPGOV* and *SADJ*) and 2 NEW parameters (*ty* and *shh*) are required.
4. For the variables
    a. Declare the variables in section 10 (Variable Declaration); remember to provide a description.
    b. Declare the parameters that will record the initial values for theses variables in section 7 (Subsection Parameters Used in Calibration); remember to provide a description.
    c. Assign values to the parameters that will record the initial values for theses variables in section 9.
5. For the parameters
    a. Declare the parameters in section 7 (Subsection Parameters Used in Equations); remember to provide a description.
    b. Assign values to the parameters in section 9 (subsection Calibration of Parameters from Data).
6. Initialise the variables: Initialise the variables in section 11 (Variable Initialisation); note how many of the initialisations use the code from the equation. **NB**: Note the use of the variable attribute *.*L* (levels values) for variables in section 11; it is essential.

Stepwise Testing of the Model

You will make errors. So, it is a good idea to have a method that allows you to progressively eliminate errors before individual errors are compounded. One way to do this is to use the

`$STOP` command together with F10 (run with GDX creation) and the default settings for the reference file and CRee (set to say 5-10) – `CErr` stops the model after the specified number of errors have occurred).

There are 4 points in the model code where using the `$STOP` command is recommended. (Remember to remove the `$STOP` command after it has done its job – it frustrating to look for output only to find out that you left a `$STOP` command earlier in the code.) These are

1. After data entry (end of Section 8) – already done (see above)
    a. Check the sets are as intended
    b. Check the data are as intended
2. After parameter (end of Section 9)
    a. Check the syntax up to the end of section 9 – resolve any syntax errors before doing the next checks.
    b. Check the parameter are 'correct'
        i. If shares should sum to one, do they do so
        ii. Are any parameters negative that should be positive?
        iii. Are all the prices correct? Basic prices equal to one and purchaser prices equal to one plus the GST rate.
    c. Think about how you might add some extra checks in the code.
3. After the first Aggregate SAM check (end of Section 12)
    a. Check the syntax up to the end of section 12 – resolve any syntax errors before doing the next checks.
    b. Check that the parameter and initial values for the variables are consistent with the data; the model will have aborted if this test is failed
        i. If this test is failed identify where in ASAM the row and column totals do not equate – the intersection of the unbalanced row and column is a likely error point
        ii. Identify where the cells in ASAM do not equate to the totals in the matching sub matrices of SAM (the base data).
    c. Check the levels values of the variables (*.*L*) are equal to the values of the matched parameter (*0).
4. Immediately before the Solve statement

   a. Check the syntax up to the end of section 15 – resolve any syntax errors before doing the next checks.

When you have carried out all these steps you can remove the final `$STOP` command and run the programme (F10).

If it does not run because of execution errors the mostly likely causes are

1. The equation and variable count is out; GAMS will tell you this is the case.
   a. The model statistics should read

```
MODEL STATISTICS


BLOCKS OF EQUATIONS          26     SINGLE EQUATIONS          45

BLOCKS OF VARIABLES          31     SINGLE VARIABLES          54

NON ZERO ELEMENTS           166     NON LINEAR N-Z            75

DERIVATIVE POOL              20     CONSTANT POOL             32

CODE LENGTH                 185
```

   b. Check all the closure conditions are included
   c. Check all the equations have been assigned.
2. One or more equation has been incorrectly coded, or the parameters are not correct.

If the model aborts because it fails, the second aggregate SAM check (this checks that the SAM returned by the model after the solve is complete and consistent).

1. Check that the solution SAM is complete and consistent data; the model will have aborted if this test is failed
   a. If this test is failed identify where in ASAM2 the row and column totals do not equate – the intersection of the unbalanced row and column is a likely error point
   b. Identify where the cells in ASAM2 do not equate to the totals in ASAM1 (the base data that entered the model)
   c. Check the levels values of the variables (*.*L*) are equal to the values of the matched parameter (*0).
2. Check the value of WALRAS in the solution. If it is not ZERO, or very close to zero, i.e., less that E-8, something has been omitted in the model.

THE MOST LIKELY ERRORS, AFTER THE SYNTAX ERRORS HAVE BEEN REOMOVED, WILL A MISCALIBRATION OF PARAMETERS.

Check that you got the algebra right: did you sum over the right index (row or column), did you use the right arguments, …….

Using the Worked Solution

In the directory `C:\cgemod\clomod\clmod2` there will be a file `clmod2_sola.gms`; this is our sample solution. One way to use this is in conjunction with the sequential `$STOP` commands.

AFTER SPENDING TIME TRYING, DILIGENTLY, TO SOLVE THE PROBLEMS AT EACH STAGE.

Use a programme for comparing files (see section 12 in the 'Intro to GAMS Studio.pdf' 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. Work out why you got it wrong and then type the changes into your code – **do not copy and paste the code from our sample code**.

Now move on to the next stage of the checking process. Again, spend time trying, diligently, to solve the problems, before resorting to our sample code.

Hints and Suggestions

Some hints and suggestions may prove helpful.

1. Do one equation at a time and complete all the stages before going onto the next equation.
2. Assigning values to the parameters is one of the trickier tasks – look at how values have been assigned to other parameters and adapt the method to the present needs.
3. The Scaling Factors have a base value of 1, e.g., for the Savings Rate the scaling *SADJ*0 =1, this is done near the start of the Parameter Assignment section (9).
4. Declaring and assigning the Parameters Used in Calibration is very useful. It may seem that you can skip some of these parameters; it is tempting to try, and you may get away with it, but sometimes you will not and at other times you will need to go

back and make the assignment later, e.g., for calculating percentage changes. Hence if you always follow the steps it can reduce errors and save time later.

5. Remember that GAMS can only use a parameter if it has already been declared and assigned, and that you can only assign something after it has been declared.
6. Use `$ontext` and `$offtext` to shut off part of the code to help with any debugging, e.g., you may want to shut off some parts of the code. This can be used in conjunction with the `$STOP` command.
7. Mistakes are normal. Learning to use the debugging facilities in GAMS will benefit you in the short, medium, and long run.

Testing Model 2 with a Simple First Experiment

One way to test a model is to run a simple experiment and then check the results make sense.

Use an experiment from the previous model (Model 1). In the directory, there is a copy of a worked example of that experiment as an `$INCLUDE` file `clmod2_test.inc.gms`. Include this file at the end of the model using the code

```
$INCLUDE clmod2_test.inc.gms
```

You may choose to compare the results from running this experiment with Model 2 and those produced from running the experiment with Model 1.

3. Model 2: Tax Policy Experiments

The policy experiments/scenarios/simulations for Model 2 all use the same set of shocks, changes in income tax rates and changes in GST rates. The first set of simulations are designed to further develop your ability to code simulations and extend your ability to interpret results. The second and third sets of simulations are designed to develop your ability to change the economic assumptions that are embedded in the version of Model 2 you coded; the second set of simulations assumes that that determinants of investment behaviour are exogenous to the model – this involves a change in the macroeconomic closure conditions, while the third set of simulations assumes a short-run solution period - this involves a change in the market clearing conditions.

Comparing the results from the first set of simulations will help you to understand how changes indifferent tax instruments have different implications for production and consumption. The second and third sets of simulations will help you to understand that different economic assumptions about the operation of an economy mean that the same shocks can have different implications.

To ensure that you are working with model files that are consistent, and thereby to make it much easier for us to support you, these exercises will be carried out using a set of pre-prepared files. If you choose to work with the files you prepared when developing your version of Model 2, we will require you to change to our versions before trying to help solve your problems. We need to be able to focus on your problems with the experiment NOT with problems in your version of Model 2.

<u>Setup</u>

The first step is to get the correct files into a working directory. So, do the following

1. Open GAMS Studio and select `File>New Project`. This opens the Import Project window that is essentially a Windows Explorer Window.
2. Add a new (sub) directory in the directory `C:\cgemod_training\clmod`; by right-clicking in the left-hand panel and choose `New>Folder` from the menu.
3. Give the New Folder the name `clmod2b` and SELECT this sub-directory.

4. The Studio panels should appear automatically once the new directory has been created with the working directory's path is reported as `C:\cgemod_training\clmod\clmod2b`.
5. In Studio press F6 and in the Model Library Explorer select the `Practical CGE Library` and then select the library file `clmod2_expt` and choose `Load` (or double click of the name), which is SeqNr: 4.
6. The `clmod2_sola.gms` model will now be displayed in the editor window and be listed in the Project Explorer as being in the project `clmod2`.
7. If you right-click on the project name and select `Open Location`, you will see that 6 files have been downloaded to the directory `C:\cgemod_training\clmod\clmod2`. (Figure 2.5).
8. Studio can now be used to work with the closed economy model.
9. The file `clmod2_sola.gms` will now be open in GAMS Studio; save this file as `clmod2_**.gms`; where ** are wild cards, e.g., your initials but that are different to those used in the previous exercise.
10. Review the model and note each of the subsections.
11. Run the model (F10) and check that it solves.

This is the first step. Do not move on until you have successful completed this sequence.

Spend time interpreting the results for each set of simulations before moving onto the next set of simulations. It will be time well spent.

Changes in Tax Rates

Assume the government decides to increase its revenue from taxes, and it is interested in evaluating the impacts of increases of 25 percentage points in the sales and income taxes. Compile an experiment file to run the following experiments

1. Increase of 25 percentage points in the sales tax on primary commodity
2. Increase of 25 percentage points in the sales tax on secondary commodity
3. Increase of 25 percentage points in the sales tax on primary & secondary commodities
4. Increase of 25 percentage points in the income tax on urban household

5. Increase of 25 percentage points in the income tax on rural household
6. Increase of 25 percentage points in the income tax on urban & rural households

The coding of the experiments will be done by modifying an existing experiment file. There are 10 steps in this process

Open the file `clmod2_test.inc`, which will be in the directory `C:\cgemod_training\clmod\clmod2b`, and save this file as `clmod2_expt.inc`.

The starting file for these experiments (`clmod2_test.inc`) has shocks for factor supplies (`FSSIM`). This file needs to be modified to create shocks for income taxes (`ty`) and GST (`ts`).

1. There are six simulations so you will need 7 members of the set `sim`; 6 for the simulations and one for the base case. Extend the members of the set sim to seven, adding descriptions.
2. In the parameter declaration section (after the keyword), declare two parameters `ty0(h)` and `ts0(c)`. These are there to store the values of the tax rates in the base data.
3. In the parameter declaration section (after the keyword), declare two parameters `TYSIM(h,sim)` and `TSSIM(c,sim)`; remember to include a description for both.
4. In the section for defining policy experiments, delete or comment out the assignment statements for `FSSIM`.
5. In the section for defining policy experiments, assign the parameters `ty0` and `ts0` as equal to the income tax and GTS rates.
6. In the section for defining policy experiments, define the shocks for income taxes and GSTs. If you assign `TYSIM(c,sim) = ty0(c)`, then you will only need to assign `TYSIM` for those sims that need to change the value of `ty`. The same applies for `TSSIM`.
7. Inside the LOOP comment out the statement `FS.FX(f) = FSSIM(f,sim)`, and replace it with two appropriate statements for `ty` and `ts`. Note `ty` and `ts`. are parameters and not variables.
8. In the file `clmod2_**.gms` and the command to include the file `clmod2_expt.inc`.

9. Make sure a reference file will be generated by default and that CErr is set.
10. Run the experiment with GDX creation (F10) and resolve any bugs.

When you have successfully run the 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. The sources of income to each household (*hvash* and *YF*).
6. Government expenditures by value (*EG*) and quantity (*QGD*).
7. Government incomes in total (*YG*) and by sources (*COMTAX*, *INDTAX*, HTAX).
8. Savings in total (*TOTSAV*) and by each household and investment (*INVEST* and *QINVD*).
9. Why are the magnitudes of the shocks so different?
10. Is the specification of the government's implicit objective for these experiments realistic?

A worked example of how this exercise might be completed is provided as `clmod2_sol_tax_expt.inc`.

4. Model 2: A Keynesian Macroeconomic Closures

This, and the next, experiment will use the same tax policy experiment file generated previously, i.e., `clmod2_sol_tax_expt.inc`. This is because these two experiments are concerned with changing the perception of how the economy operates. With this experiment the emphasis is on changing the macroeconomic closure rules, in the next experiment the emphasis is on changing a market clearing condition.

These experiments will use the technique of `$INCLUDE` files so that we do not need repeatedly to change the name of the model file.

The change in the macroeconomic closure rule in this experiment will be to change the assumption that the volume of new investment is determined by the value of savings and the prices of investment commodities; what may be called a new classical or savings driven investment function. A so-called investment driven, or Keynesian, assumption might assert that the 'animal spirits of entrepreneurs' would be determined by their expectations of future profits, which are exogenous to the model since we have no intertemporal optimizing behaviour. For now, we are not concerned with which view is correct, only with the principle that not everyone agrees about the closure of macroeconomic accounts; witness the controversies in macroeconomics.

The following steps should be followed:

1. Open the programme file `clmod2_sola.gms` and rename it `clmod2_**.gms`; where ** are wild cards, e.g., your initials, that are different to those used in the previous exercise.
2. Copy section 15 (`MODEL CLOSURE`) of the programme file and paste it into a new file. Save this new file as `clmod2_bclose.inc` (the base closure for `clmod2`).
3. Delete the contents of section 15 (`MODEL CLOSURE`) of the programme file but keeping the heading (`15. MODEL CLOSURE`).
4. Now add the code `$INCLUDE clmod2_bclose.inc` in section 15 (`MODEL CLOSURE`) of the modified programme file.
5. Run the modified model with GDX creation (F10) with and without the experiment to check that it still works correctly (make sure a reference file will be generated by default and that CErr is set.)

6. Now we can start to modify the closure.
7. Open the file `clmod2_bclose.inc` and save it as `clmod2_keyncl.inc`.
8. To get the Keynesian closure we need to fix the volume of investment and flex the savings rate. We will turn one variable into a parameter and make sure another operates as a variable.
9. To fix the investment volume make `IADJ` fixed. We do this by removing the * at the start of the line `IADJ.FX = IADJ0`. This stops this being a comment and makes *IADJ* operate as a parameter.
10. But the model now has one more variable but the same number of equations. We therefore need to fix a variable to restore the count. The choice of variable to fix needs to be driven by economic theory; in a Keynesian economic view savings would adjust, i.e., savings rate would be equilibrating variable.
11. To flex saving rates we make `SADJ` flexible. We do this by adding a * at the start of the line `SADJ.FX = SADJ0`. This makes this a comment and stops *SADJ* operating as a parameter.
12. *SADJ* operates by making the savings rates of all households change by the same amount - equiproportionately.
13. Now add a line of code in section 15 (`MODEL CLOSURE`) of the modified programme file: `$INCLUDE clmod2_ keyncl.inc`.
14. Then comment out the line `$INCLUDE clmod2_bclose.inc` in section 15 by adding an * as the first element in the line. This means we can switch between closures by deciding which include file to keep in the model.
15. Change the line of code at the end of the experiment file that reads `Execute_unload 'results_clmod2.gdx'` into `Execute_unload 'results_clmod2_keyn.gdx'`. This ensures that you keep both sets of results
16. Run the modified model with and without the experiment to check that it still works correctly (make sure a reference file will be generated by default and that CErr is set).
17. Check that the volume of investment (*QINVD*) has not changed. If it has something is wrong; resolve the error before trying to analyse the results.

When you have successfully run the 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. The sources of income to each household (*hvash* and *YF*).
6. Government expenditures by value (*EG*) and quantity (*QGD*).
7. Government incomes in total (*YG*) and by sources (*COMTAX*, *INDTAX*, HTAX).
8. Savings in total (*TOTSAV*) and by each household and investment (*INVEST* and *QINVD*).
9. Why are the magnitudes of the shocks so different?
10. Is the specification of the government's implicit objective for these experiments realistic?

A worked example of how this closure file might be completed is provided as `clmod2_sol_keyncl.inc`.

5. Model 2: An Unemployment Market Clearing Condition

Embedded in the specification of the models developed so far has been the explicit presumption that the economy is characterised by full employment. Despite the popularity of this presumption in many theoretical and applied economic models there are many circumstances where many economists might question the validity of such an assumption and/or wish to test the impact of the full employment assumption on the results of policy experiments. Similarly, it may be considered desirable to assess the short run implications of policy changes where one or more factors are activity specific, e.g., capital may not be mobile in the short run. These are the types of issue that we start to address here.

The factor market clearing rules can be more difficult to implement than many of the other closure rules. Hence the discussion below proceeds in two stages; the first stage introduces a specification whereby unemployed factors can be accommodated while the second stage introduces a generalised treatment of factor markets that can accommodate, *inter alia,* a short run factor market where in factors can be made activity specific and/or impose factor market restrictions may arise from activity specific characteristics, rather than the factor inspired restrictions introduced by a short run treatment of factor mobility. These build on the basic specification where all factors are deemed fully employed and perfectly mobile across activities.

Surplus (or Unemployed) Labour

Consider the current configuration of the profit maximisation/factor demand equations. This can be rewritten, as it is in the model code, as

$$WF_f * WFDIST_{f,a} * FD_{f,a} = QX_a * PVA_a * \alpha_{f,a} .$$

where it is assumed that *WFDIST* is a matrix of ones and fixed. This equation states that every factor is paid the same ‘wage’ rate irrespective of the activity that employs the factor. In addition, note that the model equations do not contain a specific market clearing condition for the factor markets, i.e., no (GAMS model) equation exists that ensures that total factor demand equal total factor supplies for every factor. Rather there is an exogenous restriction placed on total factor demands through the ‘closure’ condition that total factor demand equal total factor supplies for every factor, i.e., $FS_f = \overline{FS}_f$

Assume that capital is fully employed and mobile across activities, but that labour is characterised by excess supply and that the supply of labour is perfectly elastic at the current real wage rate. This scenario can be easily achieved by modifying the model closure condition with respect to the factor market. Instead of fixing the supply of both labour and capital only the supply of capital is fixed, along with the bounds on the price of capital, and the price of labour is fixed, along with the bounds on the supply of labour, i.e.,

$$\begin{aligned} FS_{"cap"} &= \overline{FS}_{"cap"} \\ \text{Min } WF_{"cap"} &= -\inf \\ \text{Max } WF_{"cap"} &= +\inf \end{aligned}$$

$$\begin{aligned} WF_{"lab"} &= \overline{WF}_{"lab"} \\ \text{Min } FS_{"lab"} &= -\inf \\ \text{Max } FS_{"lab"} &= +\inf \end{aligned}$$

Thus, the possibility of a perfectly elastic supply of labour can be introduced without any requirement to change the equations in the model.

Note that although this change is made in the model closure section it is a change in the market clearing condition for factor markets. Typically, while such changes to the factor market operations are referred to as changes in the closure of the model, they are in fact changes in market clearing conditions.

Coding Factor Market Closures

More general factor market closure options require some modifications to the model equations, and often rely on being able to modify the conditions on the variables *WFDIST*. Specifically, the *WFDIST* variables allow the rates of return to factors to vary across activities and the quantities of factors used/demanded by activities to be fixed. Thereafter assumptions about factor immobility and/or factor unemployment can be imposed by determining which of the variables referring to factors are treated as variables and which of the variables are treated as parameters.

As general rule note that if (factor market) closure rules are changed it is important to be careful to preserve the equation and variable counts when relaxing conditions, i.e., converting parameters into variables, and imposing conditions, i.e., converting variables into parameters, ***while*** preserving the economic logic of the model.

Full Factor Mobility and Full Employment

This factor market closure requires that the total supply of and total demand for factors equate. The total supplies of each factor are determined exogenously and hence the exogenously determined factor supply condition defines the first set of factor market closure conditions. The demands for factor *f* by activity *a* and the wage rates for factors are determined endogenously. But the model specification includes the assumption that the wage rates for factors are averages, by allowing for the possibility that the payments to notionally identical factors might vary across activities through the variable that captures the 'Activity specific productivity adjustments for factor prices'. These proportions are assumed to be a consequence of the use made by activities of factors, rather than of the factors themselves, and are therefore assumed fixed. Finally, bounds are placed upon the average factor prices so that meaningful results are produced.

$$\begin{aligned} FS_f &= \overline{FS}_f \\ WFDIST_{f,a} &= \overline{WFDIST_{f,a}} \\ \text{Min } WF_f &= -\text{infinity} \\ \text{Max } WF_f &= +\text{infinity} \end{aligned}$$

In this case, the conditions on WFDIST are such that these variables operate as parameters fixed at their initial values.

This is the default setting in the closures sections of the closed economy models, and has been the basis of the assumptions until now.

Generalised Factor Market Clearing Equations

Variations in factor market clearing conditions require a more elaborate arrangement. One way to proceed is to define a block of conditions for each factor. For this model this amounts to defining nine possible equations (below), where *fact* indicates the specific factor and *activ* a specific activity. The block of equations in (below) includes all the variables that were declared for the model with reference to factors, including *WFDIST*, whose role will be defined below. The choice of which equations are binding and which are not imposed will determine the factor market closure conditions.

$$\begin{aligned}
FS_{fact} &= \overline{FS_{fact}} \\
WFDIST_{fact,a} &= \overline{WFDIST_{fact,a}} \\
\text{Min } WF_{fact} &= -\text{infinity} \\
\text{Max } WF_{fact} &= +\text{infinity} \\
FD_{fact,a} &= \overline{FD_{fact,a}} \\
WF_{fact} &= \overline{WF_{fact}} \\
WFDIST_{fact,activ} &= \overline{WFDIST_{fact,activ}} \\
\text{Min } FS_{fact} &= -\text{infinity} \\
\text{Max } FS_{fact} &= +\text{infinity}
\end{aligned}$$

This block of nine equations has been added to the factor market closure sub section for each of the factors – labour and capital. The `$ontext` and `$offtext` statements above and below the full employment closure conditions should be switched on, and the `$ontext` and `$offtext` statements above and below the blocks of text like those above, should be switched on. There is one block of code for each factor – *labour* and *capital*.

The first four equations in each block are the same as those in the 'Full Factor Mobility and Employment Closure' except they are written longhand, i.e., one block for each factor suing the factor labels. If the first four equations are operating for each of the factors, this is a longhand method for imposing the 'Full Factor Mobility and Employment Closure'. Assume that this set of conditions represents the starting point, i.e., the first four equations are binding, and hence the last five equations are not imposed, i.e., preceded by asterisks.

If the model is run in this configuration policy experiments will produce the same results as the 'shorthand' method for defining the factor market clearing conditions. However, by varying which conditions are binding – switched on – and which are not binding – switched off – a wide range of different factor market clearing conditions can be imposed. These are detailed below.

Unemployed Factor Closures

Starting from the closure conditions for full factor mobility and employments and then assume that there is unemployment of one or more factors in the economy; typically, this would be one type or another of labour. If the supply of the unemployed factor is perfectly elastic, then activities can employ any amount of that factor at a fixed price. This requires imposing the condition below and relaxing the assumption that the total supply of the factor is fixed at the base level, i.e., relaxing fixed supply of the factor.

$$WF_{fact} = \overline{WF}_{fact}$$

$$FS_{fact} = \overline{FS}_{fact}$$

$$\text{Min } FS_{fact} = -\text{infinity}$$

$$\text{Max } FS_{fact} = +\text{infinity}$$

It is useful however to impose some restrictions on the total supply of the factor that is unemployed. Hence the conditions bounding FS_{fact} at plus and minus inf(inity).[2]

Running an unemployment experiment

The change in the market clearing condition in this experiment will be to assume that there is an infinitely elastic supply of labour at the current (real) exchange rate (note that the *CPI* is fixed so *WF* is real). This market clearing condition is reminiscent of the (Arthur) Lewis model but is arguably a reasonable approximation unless the supply of labour increases appreciably.[3] For now, we are not concerned with whether this approach is correct, only with the principle that it may, in some circumstances, be appropriate to allow for unemployment of one or more factors.

The following steps should be followed:

1. Continue using the the programme file `clmod2_**.gms` from the previous exercise.

[2] If the total demand for the unemployed factor increases unrealistically in the policy simulations, then it is possible to place an upper bound of the supply of the factor and then allow the wage rate from that factor to vary.

[3] Some models argue for an upward sloping labour supply curve, citing Blanchflower and Oswald as justification, despite Blanchflower and Oswald observing that this is an inappropriate used of the Wage Curve arguments. We take the view that the standard use in CGE models of an upward sloping labour supply curve is theoretically and practically unjustified.

2. Open the file `clmod2_bclose.inc` (the base closure for `clmod2`) and save it as `clmod2_unempcl.inc` (the unemployment option for `clmod2`).
3. Now we can start to modify the closure.
4. To set the unemployment market clearing condition for labour we need to fix the wage rate for labour (WF_{lab}) and flex the supply of labour rate. We will turn one variable into a parameter and make sure another operates as a variable.
5. To fix the wage rate make the supply of labour (FS_{lab}) fixed; i.e., comment out the statement `FS.FX("lab")= FS0("lab")` ; - add * at the start of the line. This stops FS being fixed and makes it operate as a variable.
6. But the model now has one more variable but the same number of equations. We therefore need to fix a variable to restore the count.
7. To fix the wage rate we remove the asterisk at the start of the line `WF.FX("lab")=WF0("lab")` ;. This makes the wage rate for labour operate as a parameter.
8. It is also wise to remove the bounds on FS, by setting the bounds as +*inf* and -*inf*, i.e., `FS.LO("lab") = -inf`; and `FS.UP("lab") = +inf`;
9. Now add a line of code in section 15 (`MODEL CLOSURE`) of the modified programme file: `$INCLUDE clmod2_ unempcl.inc`.
10. Change the line of code at the end of the experiment file that reads `Execute_unload 'results_clmod2_keyn.gdx'` into `Execute_unload 'results_clmod2_unemp.gdx'`. This ensures that you keep all three sets of results
11. Run the modified model with and without the experiment to check that it still works correctly (make sure you enter `rf=ref` and `gdx=data` in command prompt.)
12. Check that the factor supply for labour (*FS*) has not changed and that the wage rate for labour has not changed (*WF*). If it has something is wrong; resolve the error before trying to analyse the results.

When you have successfully run the 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. The sources of income to each household (*hvash* and *YF*).
6. Government expenditures by value (*EG*) and quantity (*QGD*).
7. Government incomes in total (*YG*) and by sources (*COMTAX, INDTAX*, HTAX).
8. Savings in total (*TOTSAV*) and by each household and investment (*INVEST* and *QINVD*).
9. Why are the magnitudes of the shocks so different?
10. Is the specification of the government's implicit objective for these experiments realistic?

A worked example of how this closure file might be completed is provided as `clmod2_sol_unempcl.inc`.

Appendices

Other Factor Market Clearing Options

Short Run and Fixed Factors Closures

Assume now that it is planned to impose a short run closure on the model, whereby a factor is assumed to be activity specific, and hence there is no inter sectoral factor mobility. Typically, this would involve making capital activity specific and immobile, although it can be applied to any factor. This requires imposing the condition that factor demands are activity specific, i.e., the condition (C5e) must be imposed. But the returns to this factor in different uses (activities) must now be allowed to vary, i.e., the condition (C5f) must now be relaxed.

The number of imposed conditions is equal to the number of relaxed conditions, which suggests that the model will still be consistent. But the condition fixing the total supply of the factor is redundant since if factor demands by all activities are fixed the total factor supply cannot vary. Hence the condition (C5g) is redundant and must be relaxed. Hence at least one other condition must be imposed to restore balance between the numbers of equations and variables. This can be achieved by fixing one of the sectoral proportions for factor prices for a specific activity, i.e., (C5h), which means that the activity specific returns to the factor will be defined relative to the return to the factor in *activ*.[4]

$$FD_{fact,a} = \overline{FD}_{fact,a} \qquad \text{(C5e)}$$

$$WFDIST_{fact,a} = \overline{WFDIST}_{fact,a} \qquad \text{(C5f)}$$

$$FS_{fact} = \overline{FS}_{fact} \qquad \text{(C5g)}$$

$$WFDIST_{fact,activ} = \overline{WFDIST}_{fact,activ} \qquad \text{(C5h)}$$

---

[4] It can be important to ensure a sensible choice of reference activity. In particular this is important if a factor is not used, or little used, by the chosen activity.

Activity Inspired Restrictions on Factor Market Closures

There are circumstances where factor use by an activity might be restricted because of activity specific characteristics. For instance, it might be assumed that the volume of production by an activity might be predetermined, e.g., known mineral resources might be fixed and/or there might be an exogenously fixed restriction upon the rate of extraction of a mineral commodity. In such cases the objective might be to fix the quantities of ***all*** factors used by an activity, rather than to fix the amounts of ***a*** factor used by all activities. This is clearly a variation on the factor market closure conditions for making a factor activity specific.

$$FD_{f,activ} = \overline{FD}_{f,activ} \tag{C5l}$$

$$WFDIST_{f,activ} = \overline{WFDIST}_{f,activ} \tag{C5m}$$

If ***all*** factors used by an activity are fixed, this requires imposing the conditions (C5l), where *activ* refers to the activity of concern. But the returns to these factors in this activity must now be allowed to vary, i.e., the conditions (C5m) must now be relaxed. In this case the condition fixing the total supply of the factor is not redundant since only the factor demands by *activ* are fixed and the factor supplies to be allocated across other activities are the total supplies unaccounted for by *activ*.

Such conditions can be imposed by extending the blocks of equations for each factor in the factor market closure section. However, it is often easier to manage the model by gathering together factor market conditions that are inspired by activity characteristics after the factor inspired equations. In this context it is useful to note that when working in GAMS that the last condition imposed, in terms of the order of the code, is binding and supersedes previous conditions.