

Exercises with Modelphi

Simon Wren-Lewis
University of Exeter

These exercises were written to accompany Lectures in an MSc Economics option at Exeter University. The exercises use my own non-linear model solution software, Modelphi, to illustrate certain features involved in macromodel analysis and construction. They are designed for students who have completed an MSc course in macroeconomics, but no modelling experience is required.

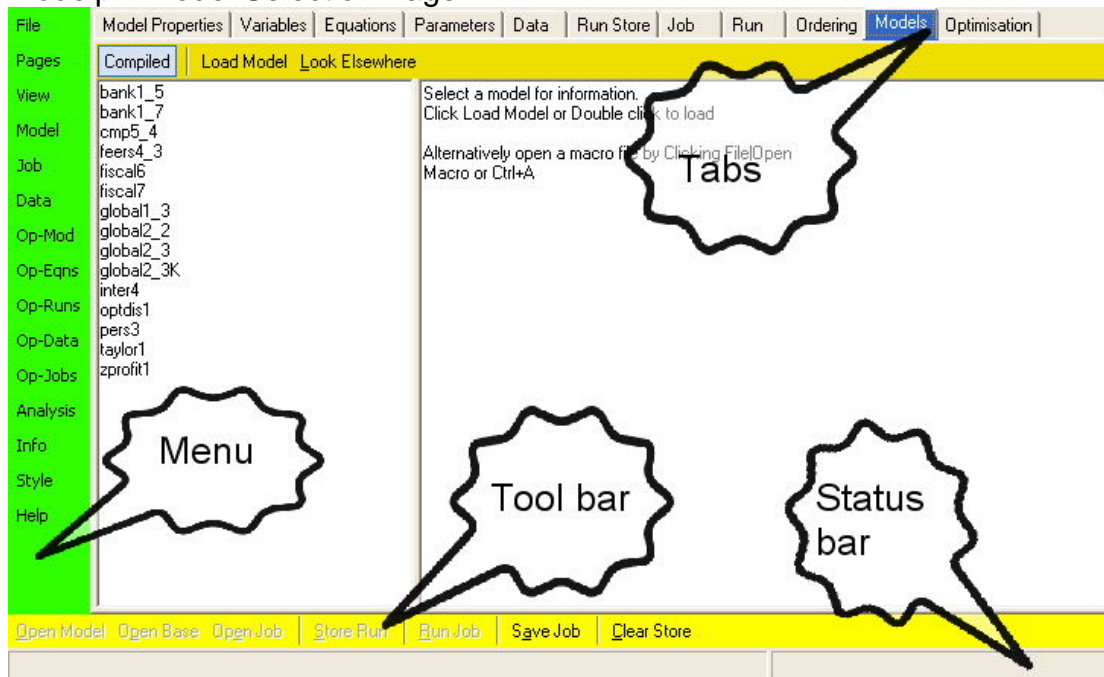
Exercise 1

Introduction to Modelphi (Modelphi's pages, Bases and Jobs). Solution methods. Using a simple growth model.

1. Starting Modelphi

Run modelphi.exe. A welcome screen will appear. Click Start, or press Enter. Modelphi's main window¹ will appear. This consists of a number of 'pages', which can be selected using the 'tabs' at the top of the window. The programme opens on the 'Models' page, which has a gold tool bar at the top. Surrounding the pages are a green menu bar² on the left, and a yellow 'tool bar'³ below with a status bar⁴ below that.

Modelphi Model Selection Page



Pass the mouse over the page to see some information displayed in the status bar.

¹ A 'window' is a moveable object with borders, with a blue title bar at the top.

² A menu bar is another standard Windows device, normally seen on the top of programmes.

³ A tool bar is a bar with buttons. Some of these buttons can be on or off, while others are just there to be clicked.

⁴ A status bar gives you information about the programme.

2. The Model Selection Page: Compiled and non-compiled models

This initial page is the model selection page, where you choose a model to work with. (You cannot do much without loading a model, but you can build a new model.) On the left hand side of the page (to the right of the green menu) is a list of models available to you. Models can be of two types: compiled and non-compiled.

All models are stored in a file with an extension of either 'mod' (when they are in text form) or 'dmd' (when they are in binary form). Compiled models also exist as a 'dynamic link library (dll)' – a file with extension 'dll'. A dll file can be directly read by another windows programme. The advantage of writing a model in dll form is that its equations can be solved very quickly. The disadvantage is that these equations, once compiled, cannot be changed.

A non-compiled model can still be solved by Modelphi, but each equation has to be 'interpreted' by the programme, which takes more time. For example, if the equation is $y=x+z$, then it first has to recognise that y, x and z are variables, that the $+$ sign means add, etc etc. However Modelphi's interpreter is quite fast, so for simple models it is often easier to work with non-compiled models.

In the gold tool bar above the list of models is a button labelled Compiled, which is currently selected. Click on this to see either compiled models or non-compiled models. If you highlight a model from the list by selecting it, then some information about the model is displayed on the right of the page. To select a model, either double click it, or click the Load Model button from the page's tool bar above the list.

For this exercise, show the list of non-compiled models, and load the model called growth1.mod. As the model is loaded, the displayed page in Modelphi's main window changes.

3. The Model Properties Page: a simple growth model

The Model Properties Page just displays a single 'property editor'. A property editor is a two-part list. On the left hand side is a list of various 'properties', and on the right hand side the values of those properties. These property editors are used a good deal in Modelphi. For example, the 'path' property of the model is simply its complete file name (including its directory path). This property cannot be changed here, but is recorded every time the model is saved. However the 'info' property is simply a piece of text, and can be changed by using the property editor. Simply select the value of that property (click on it), and retype or edit – you have changed the description of the model. For the moment, all that has changed is the value of the info property in the version of the model you have loaded into Modelphi, but if you saved this model (don't!) this property would also be saved.

A Property Editor in Modelphi

Model	Properties
StartDate	<input type="text" value=""/>
Periods	200
FrequencyPerYear	0
EndDate	199
NewIdentifier	@modelphi model file
TypeOfModelphiFile	mf_model
Info	Solow model with fixed savings rate
FileVersion	14
NumberOfStoredVariables	7
FileCreated	24/12/2004 13:57:36
Path	c:\program files\modelphi\msc\growth1\growth1.mod
Runsaved	True
NumberOfWorkerVariables	0
ModName	Growth1
ModClass	Growth
MaxLagLengthx	0
MaxLeadLength	0
MaxNumberOfPeriods	200

This model is a version of the model analysed at the beginning of the Macro course – the Solow growth model with a fixed savings rate. To see this, we need to look at the model's variables and equations.

4. The Variables Page.

The next page is the Variables Page. If you select the variables tab, a new page appears, which includes a new gold menu bar at the top, a list on the right hand side, and another property box. There is also a set of 'radio buttons'⁵ below the property box.

The list on the right hand side contains the model's variables in code form. A quick way of seeing what the code stands for is to place the mouse over each symbol in turn: you will see that a longer description appears in the right hand portion of the status bar. Now click on one of these symbols. The property box now fills up with various properties of that variable. Note that the description that appeared in the status bar is in fact the 'info' property of that variable. The code that appears in the list is its 'name' property.

The model contains a variable called 'theta'. However, theta (the rate of time preference: impatience) does not appear in the Solow model. A quick way of seeing that is to click the Trace button. If you do this for 'saverate', you will see that it appears on the right hand side of the equation for consumption (we will look at the equation below.) However, looking at Trace for theta gives nothing, so its not part of the main model. (If a trace gave $c(1)$ rather than c , then this denotes that this variable appears in an alternative equation for this variable – more on alternative equations below.)

⁵ Radio buttons are a set of circles, only one of which can be 'on' at a time.

5. The Equations Page: notation in Modelphi

The next page is the Equations Page. If you select the equations tab, a new page appears, but the variable list remains. Click on c (consumption). At the top below the tabs you will see the model's equation for consumption. It is straightforward, as we have a fixed savings propensity. However notice that the savings rate (code=savrate) is also a variable, which can vary through time. All variables are followed by their date in square brackets. Round brackets have their usual meaning. Here different colours are used for different bracket 'levels': this can be useful in reading more complex equations.

The Equations Page

The screenshot displays the 'Equations' page in Modelphi. The main window shows the equation $c[t] = (1-savrate[t])*y[t]$. A red callout box points to this equation with the text 'Selected equation'. Below the equation is a toolbar with options like 'Compiled version 0 Fixed saving rate equation' and 'Compiled version 1 Log utility intertemporal euler equation'. A red callout box points to this list with the text 'List of alternative equations'. To the right is a 'Model Variable List' showing variables like k, popgrowth, rr, savrate, theta, and y. A red callout box points to this list with the text 'Model Variable List'. At the bottom, a yellow toolbar contains buttons like 'Open Model', 'Open Base', 'Open Job', 'Store Run', 'Run Job', 'Save Job', and 'Clear Store'. The status bar at the bottom right shows 'consumption'.

Below the equation is another gold tool bar, and below that a short list with check boxes. This tells you that the model actually contains an alternative to this equation, which we could switch in if we wanted to.

Select the next variable, capital. (If the variable list has 'focus'⁶, you can use arrow keys to move down or up the list.) Note that this is a dynamic equation, as $k[t-1]$ appears on the right hand side. The equation for the population growth rate is very simple. The equation for the real interest rate contains a 'parameter', which is denoted as par[name or number]. The parameter value is given in a separate list below the equation. Parameters are useful for giving models flexibility, particularly for compiled models (see below).

⁶ A part of a window (a component) has focus if any keyboard instructions go to that component. You can generally give a component the focus by clicking on it.

6. The parameter page

All the model's parameters are collected on the parameter page. This does not contain all the coefficients of the model, but only those formally denoted as parameters: in this case, the coefficients from the production function. Alternatively, we could write the values of these parameters directly into the equation. The model would solve in exactly the same way. (To see parameters become parameter values when equations are displayed, choose one of the options from the Op-Eqn menu.)

Parameters become useful because they make it easy to change their value. If we wanted to change a coefficient that was part of an equation, we would have to edit that equation, which would involve saving a new equation at least, and possibly a whole new model. If we were working with a compiled model, we would have to recompile the model. However, if the coefficient is entered as a parameter, then we could change the value of the parameter, and we would not need to touch the equation code. Recompiling would not be necessary.

7. The Data Page, and Creating a Steady State Base

The Data page is where the variables values over time are displayed. It is quite complex, so for the moment just click on Blank Run at the middle left, and the All Variables button near the bottom right. All the variables in the model are then listed. Now click Show on the green tool bar below this list. A new page appears with a grid, showing the value of each variable for each time period. Of course, all the data is zero, because we have yet to load or create any data.

For a stylised model of this kind, a first step is to create a steady state that can act as a 'base' from which to do various simulations. One simple way to do this is to select from the left hand side menu: File+Create Steady State Base.⁷ A new page appears: the Run Page. If the text on the main part of this page is black, then this operation worked successfully. Now go back to the Data page, and click Show again. The grid now contains non-zero values, which are constant through time. The model has solved for steady state values.

How was this done? The first step, which can only occur using Modelphi's interpreter, was to ignore the dynamics in each equation. In effect, all variables are set to the same date t . Then, some starting values are given to each variable. (We could start with the zeros we saw earlier, but this would cause problems for some forms of equation e.g. division by zero.) The model is then solved, in a way that is discussed below. Finally, the solution is then set up as a base.

⁷ The instruction File+xxx means click File, and then the submenu xxx.

8. The Job Page, and a first simulation

Having set up a base, we can then do a simple simulation. Simulations are created using the concept of a 'Job'. A job is essentially a set of interventions to the data or model. From the toolbar at the bottom of the main window, click Open Job. This shows the set of available jobs that can be used with this model. Select *savrate.mjb*. This loads the job, and shows the job page.

At the centre of the job page is a large box, with a single entry in it. Select this entry by clicking on it. The entry is of type A, for variable *savrate*. Above this box are radio buttons, and you can see that intervention type A is selected. This stands for Additive residual. Double click the entry. A new window appears with a simple grid. This shows that 0.02 is added to the variable *savrate* from period 1 onwards.

The exogenous value of *savrate* in the steady state base was 0.2 (20%), so this increases it by 10%. At the bottom of the job page we have the job start and end dates. This simulation will begin in period 0 and end in period 199. To run the job, click Run job in the lower tool bar.

9. Examining the results of a simulation: the Data Page in detail

Once again the run page appears. If the run was successful, then we can go back to the data page, and look at the results. First, look at the left hand side of the page. At the top are dates: these have been set as the dates of the run, but they can be altered to focus on smaller data intervals. Below this, in red, is a label saying 'Newrun'. Below that is a list of various 'forms' of data that can be shown. Note that the highlighted item says Run-Base. That means that any variables you display will be in the form of the absolute difference between values in the simulation, and values in the base.

The Data Page

The screenshot shows the 'Data' page of a software application. The interface includes a menu on the left, a toolbar at the top, and a main data table. The 'Data' tab is selected in the top toolbar. The 'Variables to' section shows 'Graph' selected. The table below has the following data:

Variable	Run	Form	BaseRun
savrate	Newrun	Run - Base	1
y	Newrun	Run - Base	1

Annotations with arrows point to 'Newrun' (labeled 'Variables to view'), 'Run - Base' (labeled 'Runs to view'), and 'Run - Base' (labeled 'Variable form'). A 'View form' label points to the 'BaseRun' column.

It is always good practice, when looking at simulation results, to check that the intervention you made in the job carried through to the simulation. So let's look first at the savings rate. You can either double click savrate in the variable list, or select it and then click Add on the blue tool bar. In the main list you will now see a single variable, savrate, in the form of Run-Base, for NewRun.

Above this list you will see radio buttons, and 'graph' is selected. So if you now click Show, then a new graph page will appear. In this case a graph is not very helpful, so select 'grid' instead. By clicking Show we can see that nothing changed in period 0, but from period 1 onwards the savings rate increased by 0.02, which is what we intended with the job. Now let's see what happened to output. Either double click 'y' in the variable list, or select and click Add. Select 'graph', and click Show. You can see that output gradually increases to be higher by 0.1 (10%), but the adjustment period is very long.

You can look at other variables. As a shortcut, double clicking on the white part of the main list clears it of all variables. Try changing the dates over which the data is graphed. You can also look at other forms of the variable, such as '%Run/Base'. You can look the values in the simulation itself (run value), or the base itself. Note that every simulation will involve two sets of data: the run and the base.

10. Editing jobs, and the Run Store Page.

This was a permanent change to the savings rate. Suppose we wanted to examine a temporary increase, over 10 periods for example. Return to the job page. Select the single intervention. Then click on Edit below this, on the yellow tool bar. Now click on the 0.02 underneath period 11. Type in zero. Do we need to type in zero for all subsequent periods? Luckily not, there is a shortcut. Select 0.02 under period 12. Then right click. From the pop up menu, select Extrapolate flat data. This and all subsequent periods take the value zero. Now click OK. You will see that the 'dates and values' part of the savrate entry has changed. We now have only a temporary increase.

If we ran the job now, it would wipe out the results of the previous simulation. To keep the previous results, click Store Run from the bottom yellow tool bar. You will be asked to type in a tag: try sr_perm10%. You now see a new page, the Run Store Page. This simply consists of a list, and a property editor. In the list you will see Output 0, Output 1, and the Base for both runs. Output 1 is what you just stored, our permanent savings rate change. It is a good idea to record what these runs are as you store them. You have already done this for the tag property. You can also type in a long description under the 'Info' property e.g. permanent savings rate change.

Now click Run job. Go back to the data page. You will see that, where before there was just 'New Run', there is now an additional item, called sr_perm10%. So we have two runs: the one we stored, and the run we have just done. If you double click on 'y', then it will only add this variable for the run highlighted in red. However you can select both run, by holding down the Ctrl key and clicking on the other run. Now both are in red. Double click on 'y', and two items will appear, which are identical except for the run they refer to.

Click Show, and you will see that the simulations are identical for the first 10 periods, but diverge thereafter.

You are now in a position to try other experiments. You can change other exogenous variables. Go to the job page, select the change, and click Del on the tool bar below. That intervention disappears. Now select 'popgrowth' from the variable list, and click Add. Type in some numbers, and click OK. (Remember this variable is 0.05 in the base, so the numbers to type in should be small e.g. 0.01.) You can now do a simulation where the population growth rate changes.

11. The importance of the Base, and Editing the Model

All these jobs ran off the base we created right at the beginning, which was the model's steady state. Would a savings rate simulation look the same if some of the model's parameters were different, so the base was different? To examine this, let's look at what would happen if the steady state population growth rate was 3% per period, rather than 5%. We could create a job that permanently changed 'popgrowth', but this would give us the results of this variable changing, and so the simulation would be dynamic. Instead, we want to look at the steady state implied by this change, and then do a simulation off this new base.

First, go to the Equations Page, and select popgrowth. In the gold tool bar below the equation, click Edit. A new window appears: the Equation Analyser window. (It may seem like a new page, but in fact the main window is still there, hidden behind the new one.) Under where it says 'Type in the new equation..', change 0.05 to 0.03. Click replace in the tool bar below this, and keep the old equation as an alternative. You go back to the Equation Page, but now with a new equation.

Now from the menu, click File+Create Steady State Base as you did before. Then go to the Run Store page. You will now see two bases. One for Output 1, and one for Output 0. Output 0 is always the data created when you run the model. So Modelphi, realising that you are creating a new base, has stored the old base to go with the output we previously stored.

Go to the data page, and select both runs, but also 'Base Value' from the list below. Now select variable 'rr', real interest rates, and show these in a grid. You can see that in the new base real rates are 7.5%, compared to 12.5% before. Why is this? Well, in steady state we have

$$y_s = kn \text{ or } y/k = n/s$$

i.e. savings equals investment for the new born. But with a Cobb-Douglas production function

$$rr = f'(y) = \alpha \cdot y/k$$

so real interest rates has to fall in the same proportion as population growth.

Now click Open Job from the bottom tool bar, select 'savrate.mjb', and run this job. We now have two identical simulations, that differ only in the base they use. From the Data Page, select both runs, and output, and graph this. As you will see, convergence to equilibrium is even slower from our new base.

12. Additional Exercises

1. Recreate the steps 2 to 10. Note any questions for next time.
2. Try alternative simulations changing 'savrate' or 'popgrowth'.
3. Try looking at the same simulation with different bases

Exercise 2

How does Modelphi solve models? (Model ordering, Convergence criteria). Intertemporal consumption model. Forward looking models. Anticipated changes.

1. Solution methods

How exactly does Modelphi solve a model? There are three key concepts here: simultaneity, linearity and convergence. We know how to exactly solve a set of simultaneous equations of the linear form

$$x = Ax + By$$

where x is a vector of endogenous variables, A and B are matrices of coefficients, and y is a vector of exogenous variables, providing we can invert the matrix $A-I$. However models may not be linear.

An alternative solution method involves *iteration*. We start off with some starting values for x , and compute each equation sequentially. As we solve each equation, we will get new values for x . This is the first iteration. We then repeat the process, until the solved values for x are almost equal to the values in the previous iteration. At this point we say that the solution has converged. (Take the linear system above. Suppose we guessed the true solution. If we then put this solution into the right hand side of each equation, we would find it got the right answer for each left hand side variable.)

Modelphi is based on an iterative procedure (Gauss-Siedel), but can use linear approximations to speed convergence. To see how, open Modelphi, and load growth1.mod.

2. Saved bases

Last time we created a steady state base. We could do so again, but we can also load a base directly from file. Click Open base from the tool bar at the bottom of the page, and load 'growth1.ddb'. This is a binary file containing data for all the variables in the model. You can check that this contains the same values as the base we created last time.

The file growth1.ddb is large, and it may seem extravagant to save bases in this way. However, we shall see shortly one reason why this may be necessary. A more frequent example of a base that has to contain data from a file is an estimated model where the base is historical data.

Load the savrate.mjb job, and run it. Now look at the Run Page. Note the line that says 'Sim block solved in 3 iterations'.

3. The ordering page, iterations and crashes.

Click on the Ordering tag, which presents a new page. On the right is the beginning of a 'Tree List', called Model Ordering. Click on the + signs, until the complete tree is visible. Model variables, which appear in black, are

nested in two 'loops', called `presim0` and `backward0`. They are separated into two groups, because those in `presim0` are not part of a simultaneous system. They are predetermined variables, which can be computed on their own, without the need to know the values of any other variables.

The simultaneous block involves four variables. Two of these are linear, but two are not. (Check by looking in the equations page.) If you select the `backward0` loop in the tree, you will see that the same loop is highlighted in the list on the left. To the right of it is a property box. The property 'Solution Method' has the value `ST_NR`. This means that the loop will solve by taking a linear approximation to its variables, and solving this by inverting the derived matrix.

How does it create a linear approximation? By changing each variable by some amount, and computing the effect on all the variables in the iteration bloc. In other words it computes a set of partial derivatives or multipliers. We can see the implied matrix by right clicking on the loop in the list. (This process will overwrite our previous run – do not worry about this.) Check the matrix by looking at each equation.

Why bother to do all this, when the model is not exactly linear. The answer is that it is roughly linear, so this method gets us to the solution fast. Recall the line on the run page that said 'Sim block solved in 3 iterations'. This meant that, using this linear approximation, we required three iterations before we got to a converged solution.

Could we have used a 'pure' iterative method, that did not try to use a linear approximation? We can see. Note the radio buttons above the loop list. Select `job`. Now click on `backward0`. If you click on the solution method property, you get a 'drop down list'. Select `ST_GS`. (Loops exist for both the model, and any job. Think as the model values as defaults, and the job values as things that you can change – if necessary – depending on the type of job.) Now run the job, and look at the Run Page. We now needed many more iterations.

Using linear approximations will always (?) reduce the number of iterations, but it takes time to compute the approximation (particularly for a large number of variables: matrix inversion time increases with size exponentially), so there is always a trade off. However another reason for using a linear approximation is that we are more likely to get to a solution. Iterative methods may iterate towards the solution, but they can iterate away from it!

In Modelphi, what happens in this case – does the programme compute forever? No, because each job contains limits on the number of possible iterations. If this limit is hit, the solution ends, and the run page will be in red, with some message indicating failure to converge. However, problems may occur before this. If a solution is diverging, variables may generate extreme values, and the solution will 'crash' – a typical example would be taking the log of a negative number. Modelphi will record the error, and abort the run.

The speed of a pure iterative solution may also depend on the order in which variables are solved. To change the ordering in the job, click Mod above the ordering tree, and then click Job. (This selects the ordering for the job, not the model.) Expand, and drag the 'k' variable to above y, but after c, so the ordering is 'c,k,y,rr' rather than 'c,y,k,rr'. Now run again. The number of

iterations should fall substantially. (Ordering within a loop has no effect if the loop is solved using ST_NR.)

4. Convergence

Return the solution type of backward0 to ST_NR. Go to the job page, and click the Properties button under the main list. A property editor will appear, which are the properties of the job. Look for the properties 'AbsConv' and 'PercentConv'. Add a couple of zeros into both. Now rerun the job. The run page tells you that the sim iterations block now took 4 iterations rather than 3.

The numbers you changed was the tolerance by which the programme judges a solution has converged. After each iteration, it checks whether the new solution for each variable is within this tolerance of the previous iteration's solution. If it is not, then another iteration is required. All variables must pass one of either an absolute or percentage convergence criteria.

Thus for a non-linear model the programme never generates an exact solution, but something tolerably close to it.

5. A forward looking model – intertemporal consumption.

Go back to the Models Page, and now load growth3.mod. Loading a new model clears all existing bases, runs and jobs. As you will see from the Info property, this is the basic neoclassical model with government that we analysed in the main macro course. Click on the Equations Page, and you will see that it contains most of the same variables as the previous model, but two new ones: 'g' and 'oil'.

Looking at the equations from the bottom of the list, the output equation is the same except for an additive term in 'oil': 'oil' is output that does not require capital to produce it. The real interest rate equation also has to be adapted for oil. In this specification, oil output is zero. The capital equation, which is of course the national income identity, now has government spending in it. The main difference is the consumption equation, which is the Keynes Ramsey rule. This equation is forward looking: consumption at t depends on consumption at $t+1$.

Now click on the ordering page, and expand the model ordering tree. You will notice that the simultaneous section is more elaborate than before. This is because the model is forward looking, which creates issues for model solution that we will discuss below.

6. An example of multiple solutions

Lets see if we can create a steady state base as before: click File+Create Steady State Base from the menu. It solves, so look at the solution in the Data Page: click All Variables. You can check the solution works, but only because consumption is zero. But zero consumption is a perfectly valid solution, because the Keynes Ramsey rule holds, but it is an economically uninteresting solution.

Is there another, more interesting solution? We can answer this by clicking Open Base, and again loading 'growth1.ddb'. Look at the values in

the Data Page. The value for 'g' is wrong: zero instead of 0.2, because the growth1 model did not have government.

Now from the menu click Op-Runs+Steady State Base from existing base. This option is now checked. Try File+Create Steady State Base again, and look at the solution. Consumption is no longer zero, but 0.6, and $g=0.2$. So we have created a new and different steady state solution. In particular, we have $rr=\theta+\text{popgrowth}$, so the Keynes Ramsey rule holds.

Why do we get the economically meaningful solution this time? The only difference is in the starting values the programme used to compute the steady state. In the first case, it used a naive starting point – each variable was equal to unity. In the second is used a more sensible starting point, the growth1 base.

We could create this base each time this way, but here it is far easier to load the base from file. In this case it is growth3.ddb. How can we check that a base from file is in fact a true steady state solution, which recreating it ourselves? From the Data Page, clear the list of variables to plot, and click the plus sign besides 'Additive Residuals' which is in the same list as 'Base Values' etc, and then select 'A Res using interpreter'. Click All Variables again. What this does is compute each equation using base values, and reports the difference between this and the base value: the *residual*. If it is zero, the equation holds. We can see all zeros, except for capital in period zero. This is because there is no value for k in period -1 , so the programme takes zero, rather than the steady state value of 4, so the residual is also 4.

7. Increasing impatience, terminal values, and more on starting values

Now load the job called 'theta.mjb'. It involves raising impatience by 0.01, from period 1 onwards. Run the job, go to the Data Page, select absolute differences, and plot θ and rr . From the Keynes Ramsey rule, we know that real interest rates will eventually increase by the same amount as the rate of time preference. However, as the capital stock changes gradually, so real interest rates will only gradually move to this new steady state.

Note that the solution at the end is not smooth, but wiggles slightly. This is a problem that often occurs in forward looking models, and reflects in part the problem of terminal dates.

The simulation starts in period 1. The equation for capital is backward looking, and requires $k[t-1]$, but this exists in the base, and because the simulation starts in period 1 it will also be the solution value. However the problem for the consumption equation is more difficult. In the last period of the simulation, period 197, the consumption equation requires a value for $c[t+1]$, but what is consumption in period 198. We cannot take the base value, because we know the solution will be different. (Taking the base value might not be a bad approximation for a temporary shock, as this would be the old and new equilibrium.) This is the terminal value problem.

There is no correct solution to this problem, apart from solving the model to infinity. So we have to use an approximation. If we click on the variables page, and select consumption, we see two properties: ExpectType and ExpectLeads. The latter tells the programme that we will need a value for $c[t+1]$. The former tells the programme how to generate that value. In this case zero (or 1) means that $c[t+1] = c[t]$.

We can see this from the Data page. Choose absolute differences again, and select consumption and capital, but set the last period to 198, and look at grid values. While the value for k is zero (it is at its base value), the value for c is not.

We know this will only be a correct assumption if the model has reached its steady state. (More precisely, it will be the correct assumption is the solution is sufficiently near the steady state given the model's convergence criteria.) For this reason it is always important to simulate forward looking models for long periods of time, so that the last period is close to its equilibrium. For example, store this run, and then change the end date on the job page to 100. The solution now looks a lot less convincing.

8. Anticipated changes and an oil shock

With forward looking models, the starting period of the solution is the period in which all information became known to agents, and not necessarily the date from which changes occur. This is because the solution is based on perfect foresight. To see the implications of this, load a new job, `oil_anticip.mjb`.

From the job page you can see that this raises oil output from zero to 5% of steady state output from period 5 onwards, but the simulation starts in period zero. Run the job, and plot absolute differences in oil and total output. Although total output does rise by the increase in oil in equilibrium, it falls *before oil is produced*. To see why, look at consumption. This anticipates the increase in income, and so rises from period zero onwards. As no additional output has yet been produced in this closed economy, extra consumption must be at the expense of lower capital, and therefore lower output.

9. Additional Exercises

1. Recreate 8 and 9 (load the base `growth3.ddb` directly). Note any questions for next time.
2. Use the job `y_multi_antic.mjb` to recreate the multiplicative output change from the handout.
3. Use `g_temp2.mjb` to recreate the government spending shock from the handout. Edit the job to look at a longer lasting shock, or a permanent change. You can also try experiments with anticipation.

Exercise 3

Introducing the EJ2000 model. Ricardian Equivalence with Blanchard Yaari consumers

1. The EJ2000 model

This model was published as Leith and Wren-Lewis (2000), Interactions Between Monetary and Fiscal Policy Rules, Economic Journal 110, 93-108. The model has the following differences from the previous one:

- a) there is no capital, but there is government debt and money
- b) consumption is based on finite lives (Blanchard/Yaari)
- c) the model includes nominal inertia (a Phillips curve)

The equation for consumption should be familiar. It implies that consumption is a fixed proportion of total wealth, which is made up of human capital and financial wealth. This is the Blanchard/Yaari constant probability of death model, with log utility.

The equation for government spending allows for feedback from disequilibrium in government debt. Human wealth is discounted future labour income: this equation replaces future terms in income by the lead value of human capital. The Phillips curve can be forward or backward looking. Real interest rates are determined by a monetary policy reaction function, where real rates rise if inflation is above some target level.

Tax is proportional to output, but with the possibility of feedback on debt disequilibrium. In this version of the model the debt disequilibrium feedback for tax is zero.

The model involves three dynamic processes: wealth (=government debt), inflation and consumption. The first is backward looking, the third is forward looking, and inflation can be either or both. In this version of the model, the Phillips curve is purely forward looking.

2. The base, and a consumption shock

Rather than create a steady state base for this model, we can load one directly. We can again check this is a valid steady state base by examining the additive residuals for all variables: set the initial period on the data page to zero. We can see we have zeros everywhere except in period zero for some variables. You can check that these are variables whose equations involve one lagged variable: wealth.

A key parameter in the model is μ , the 'mark-up' on real interest rates used to discount income in human wealth. Recall in the constant probability of death model it is the probability of death. Assuming that one period here is one year, then the value of 0.06 is rather high: in the paper we argue that it also captures uncertainty effects. This parameter then determines the steady state value of wealth (=government debt) in relation to income, which is normalised to one. The value of about 40% is not unrealistic.

Now load the job `ct_f_a_5.mjb`. It consists of a single additive residual: a temporary but 'autocorrelated' negative shock to consumption. When we run this job, we see that the `sim(ultaneous)` block takes 97 iterations to solve. This suggests that it uses a purely iterative solution method.

First look at absolute differences in consumption, in the grid. Its values are similar to the input shock, but not exactly the same. This is because we added the shock to an endogenous variable. (Our previous additive residual interventions have always been to variables that are exogenous.) Now look at output. Its deviations from base are very similar to consumption. This is not surprising in one sense, but it illustrates that in this model output is demand determined in the short run because of the presence of nominal inertia. Notice also one rather interesting property of the output reaction: after period 5 (when the shock ends) output returns to base, even though consumption does not.

The reason can be seen by looking at government spending, which falls by an amount equal to the increase in consumption. The reason is that both private and public consumption depend on government debt (=wealth), but in opposite directions. While extra debt raises wealth and hence consumption, it means government spending is cut back in an effort to bring debt back to base. The two processes exactly offset each other simply because two parameters, `par[6]` and `par[3]` happen to be the same.

If we plot graphically the variables 'g' and 'w', we can see that they mirror each other, although with damping on g and a lag. Wealth initially rises as a natural consequence of the consumption shock: less consumption means more savings, which in this model can only be in the form of additional government debt. Once the shock is over, lower government spending gradually reduces debt (or, equivalently, consumers steadily spend the wealth they have built up).

If we look at inflation (again, absolute difference from base: the base is zero!) with output, we can see how inflation depends on cumulated future output disequilibrium. Real interest rates follow inflation. Lower real interest rates are why consumption is higher than the negative shock (see human capital), but this reaction is too weak to dampen the negative shock very much.

3. Exogenisation: how important is money?

In this simulation, money increases following lower nominal interest rates. But how important is money in this simulation? One simple way we can investigate questions of this kind is to exogenise the relevant variable. This can be done very easily is part of the job.

First, store the simulation we just examined. In the job page, select 'm' in the variable list, and select to X type among the Intervention type radio buttons. Now click Add below the main list. You can see that we have exogenised this variable in every period: click OK. (It is possible to only exogenise a variable for selected periods.) Run this job.

In the data page, select both this run and the previous run, and double click on m in the variable list. View the two series in the grid. The exogenisation worked: money does not change. Now look at other variables. The simulation does not change very much. Wealth increases a little more:

now all savings are interest bearing, which increases income. However the general conclusion would have to be that money does not matter much for this simulation at least.

Although exogenisation can be a useful device, it has to be used carefully. It can change the structure of models in such a way that the model no longer make sense.

4. Parameter changes (and non-steady state bases)

Three of the model's equations (for g , tax and rr) are policy reaction functions. In this version of the model, tax rates are fixed, and fiscal policy operates through government spending. We noted above that the feedback from debt to government spending exactly offset the wealth effect in the consumption function.

Suppose we double the rate of fiscal feedback on debt. We could do this by editing the model, but it is more convenient to do it using a job parameter change. (This avoids having to save different versions equations or the model.) From the equation page, select the 'g' equation. The parameter we want to change is $par[3]$. Double click on this in the parameter list.

The parameter page appears, with this parameter selected. Under the main list, click Add/Edit Job Par. Type in a new value of 0.16. If you go back to the job page, you will see it listed there as a P type intervention. Now run this job. Compare 'g' in both runs: it reacts more quickly in the new run. Wealth also returns to base more rapidly. If you look at output, you will see that now output differs from base after the shock.

The most interesting comparison is inflation. Although debt is corrected more quickly, inflation disequilibrium is actually worse throughout the simulation. This is one of the main results discussed in the paper. (The paper can be accessed by clicking [Help+Manual.](#))

5. Other possible elements of a job.

We can see from the job page that there are six types of intervention that can form part of a job. We have already used additive residuals (A type), exogenisation (X type), and parameter changes (P type). Another type that is frequently used are multiplicative residuals (M type). These are useful if you want to multiply a variable by some amount, rather than add to it. To increase a variable by 10%, you will input 0.1 into the M type residual.

The two remaining types of intervention are more specialised, and less frequently used. V type interventions change the variable properties in a job. E type interventions allow estimation as part of a job.

These six types fall into two categories. Residuals (A or M type) represent shocks to the data, and are required for a simulation to do anything. The remaining types alter the structure of the model, and therefore the response to A or M type changes. They are a convenient shortcut for changing the model.

6. Ricardian Equivalence with Blanchard Yaari consumers

Let us now construct a completely new job. It is generally wise to do this by editing an existing job: that way some of the background data (such as dates) will be set correctly. So first, select the consumption shock, and press the Del key below it. Now select the tax variable, select the M type, and click Add. Type in -0.01 for four periods from period 1. Click OK, and then run the job.

Look at taxes first. They were cut by 1%, but they remain endogenous, so they will still vary with income. (If you wanted an exact 1% cut, you could exogenise the variable for the four quarters of the tax cut.) Now add consumption. You can see the standard result. Consumption smoothing spreads the increase in consumption over a much longer period, and thus the marginal propensity to consume is small.

What would it take to generate a larger increase in consumption? Increase `par[6]` to 0.12 and rerun the simulation. Consumption is higher, but not by much.

This experiment raises a question. In this case, should not the base change as well? Will the simulation still be valid?

The strict answer is that, yes, the steady state base will change. However, the simulation using the existing base still works, in the sense that we get valid simulation results. How does this work? The answer lies in how simulations are done in Modelphi.

One of the tasks Modelphi undertakes before implementing a simulation job is to *reproduce the base*. It does this by calculating the residuals needed to ensure that each equation produces the values in the base. Having made the parameter change to `par[6]`, look at the model's additive residuals again. You will see that those for consumption are now non-zero. The run then stores these residuals, and adds them to the equation when running the job. This ensures that the only reason the simulation is different from the base is due to the intervention residuals that are part of the job.

7. Damping

We can illustrate an additional point, by increasing the parameter to 0.16. If you run this, you will find the model crashes. Have a look at the data for this failed run (select absolute differences from base, and all variables.) You can see that the numbers are wild: the iterations were explosive.

What can be done in this situation? There are a variety of possibilities, but the most straightforward is damping. From the menu, click on Analysis+Damping. This tells you that two model variables have damping. What is damping. In each iteration, each variable is updated as follows

$$x(i) = \text{damp} * xc(i) + (1 - \text{damp}) * x(i-1)$$

where $xc(i)$ is the calculated value of the equation. No damping, which corresponds to $\text{damp} = 1$, does what we expect. Any value $0 < \text{damp} < 1$ dampens the computed value of the equation.

Damping obviously stops the variable getting to the true solution quickly, so why do it? The problem occurs if the variable is iterating away from its solution. If the iterative solution for one variable is explosive, then damping may avoid crashes. We can see here that one particular variable, inflation is heavily damped. The reason is its dynamic structure: if the path of output is slightly away from its solution, inflation will cumulate these deviations.

To change a variable's damping, click on the variable page. Select inflation from the variable list. You could change the model value of its damping property, but as with parameters, it is better to change this property only in the job. To do this, click on Add/Del Job. This creates a duplicate of the variable's properties that apply only in this job. Change its job property from 0.1 to 0.05. Now try running the simulation again.

8. Additional Exercises

1. Repeat the exercises and note any problems.
2. With the tax cut simulation, can you account for the precise path of consumption. (Look at wealth and monetary policy). Experiment with changing the strength of monetary policy.
3. Try adding a shock to capital. Your run will probably crash or look odd, because you did not take into account the form of the equation. What type of shock should you have added?

Exercise 4

Demand Stabilisation and Optimal control

1. Changing equations in the model: fiscal stabilisation.

A final way of changing the model in a job is to add an alternative version of an equation. As a simple example, suppose we wanted to investigate whether countercyclical stabilisation by the fiscal authorities would help reduce the impact of our consumption shock.

Check that the original simulation is stored: if it is not, recreate and store it. Select the government spending equation, and click on Edit below the equation. In the Equation Analyser window, add the following to the equation:

$$-0.2*(y[t]-ys[t])$$

Now click View below the equation. This tells you whether the equation contains any syntax errors. If it is OK, click Add to Model, and add 'with output stabilisation' as a comment.

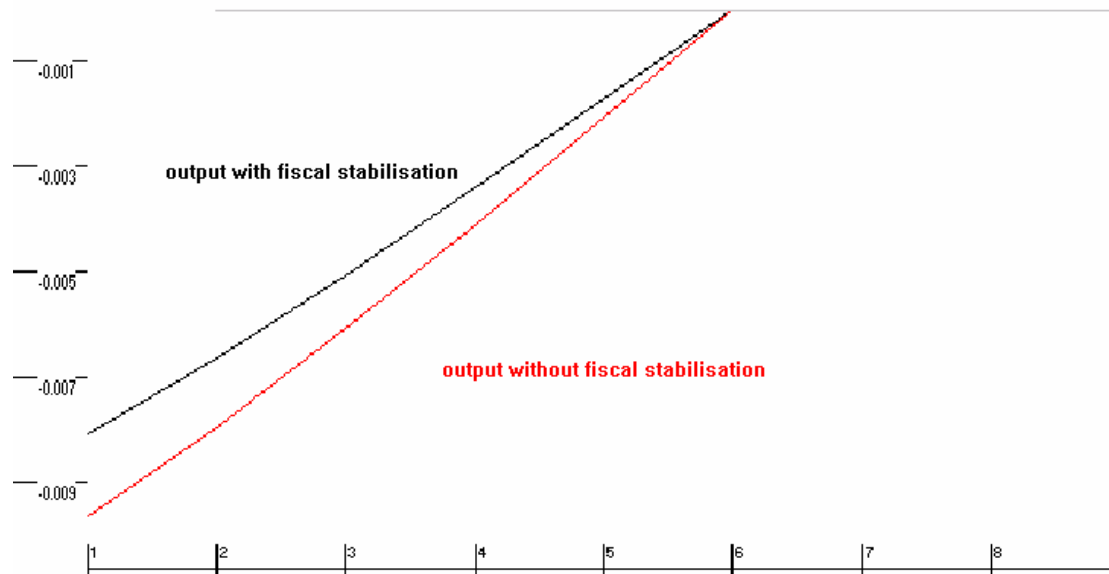
You will see on the Equations Page that there are now two versions of this equation. The selected equation is displayed, but the checked equation is used in the job. Check the new equation, and run the job.

If you look at government spending in the two runs, you can see the impact of the fiscal stabilisation you have introduced. (Would increasing spending on this scale be politically realistic?) If you look at output, you can see that it has had some effect in dampening the shock.

2. Saving output, jobs, models and runs in Modelphi

(i) Saving output

As you complete runs in Modelphi, you may want to document your results. The simplest way to do this is graphically. From the graph page, if you right click, you will see a menu of options. Copy to clipboard simply copies the graph as a bitmap to the clipboard, so you can paste it directly into a document e.g.



Notice that I have modified the graph in a number of ways. First, I have set the end date to something nearer the start of the simulation. Second, I have deleted the exist button. Third, I have relabelled the legend (right click near to them and select title) and repositioned each of them (drag and drop). An alternative is to select Data to Clip (excel) from the right click menu on the graph. This saves the data in grid form: if you open excel, you can paste the results.

You can also save the output as text. Exit from the graph, and select Text from the radio buttons rather than Graph. If you now click Show, you get a table of selected results. You can edit the form of this table by clicking Format next to Show. Details of how this works are explained in the model manual. You can also save the data to file: again see the model manual for details.

(ii) Documenting and saving jobs

Jobs are saved as text files e.g. ct_f_a_5.mjb. If you create a new job, and use the results from this job, or may want to use the job again, then you should save it. Simply click Save job on the yellow tool bar.

As the job is a text file, then you can also use this as documentation in any description of what you have done (e.g. as part of an appendix.) This is good practice, because it shows others exactly what you have done, and allows others to recreate your work.

As equation variants are stored as part of a job, then you can even save changes to model equations this way, without having to save a new model. For example, save the current job, as temp.mjb for example. Now shut down Modelphi, and start it again. Load the fiscal8 model, and its base. Check that there is only one equation for g. Now open the temp job. You will see that the new equation has been added to the model, and selected as part of this job.

This will only work if your new equation uses existing model variables, and does not create any new parameters. If your changes to the model are more elaborate than this, then you should save your new model instead. This

is done from the menu: File+Save mod/dmd file. Do not save the model for compiling – you cannot and should not need to compile a model. Make sure you save any new models or jobs using new names, rather than overwrite existing models.

You can save all the data from simulations, but this should not be necessary, and takes up much more file space than saving the job and the model used to create it. You may, however, need to save a base if you create a new one. Go to the Run Store Page, select the base, and click save. (Do not save as a Modelphi datafile.)

3. Monetary policy activism

In very simple models such as this one, the effects of demand shocks can often be effectively eliminated by a sufficiently active monetary policy. To see this, compare the basic consumption shock with a value for par[5] of 10 (rather than 0.5).

However, an even more efficient way of eliminating the impact of a demand shock is by using optimal control.

4. Optimal control: the general control problem and solution

Suppose some variables of the model can be chosen as objectives (vector y), and others as policy instruments (vector x). Assume policy makers potentially have complete control over x . Each vector can include the same variable at different dates.

A linearised version of a macroeconomic model can be represented as

$$y = Rx + s$$

Here both y and x are in terms of deviations from base, and s is some shock. R is a matrix of instrument multipliers.

Suppose the government wants to minimise quadratic deviations of y from base, following the shock. i.e. it wants to minimise

$$y'Qy/2$$

Here Q is a diagonal matrix of 'weights', and y' is the transpose of y .

The solution to this problem is to choose the values of instruments such that

$$x = -(R'QR)^{-1} R'Qs$$

In other words we can find the instrument settings that will minimise the disruptive effect of the shock on our objectives, assuming that the shock is known. (The control problem can also be cast in a 'forecast' context, where we attempt to minimise the deviation of objectives from a set of desired or 'bliss' values rather than a base.)

Most optimal control algorithms for macro models (including Modelphi) exploit the model's 'near linearity'. Modelphi first computes the matrix R , by

carrying out a set of conventional simulations. It then uses the formula above to compute x , simulates the result and computes the new objective function. It then repeats this process from the simulation solution, seeing if further improvement is possible. (If the model was linear, only one iteration would be necessary.) These iterations stop when no significant improvement occurs.

5. Optimal control with Modelphi

Remove all runs from the store. Make sure the consumption shock job is loaded. Go to the Optimisation Page, and select periods and variables for objectives and instruments (drag variables across from the list). As a first try, select periods 1 to 5 for both objectives and instruments, choose rr as the instrument and inf as the objective. Press Go.

You can see the value of the objective function in each iteration. As inflation is the only objective, you are minimising the sum of squared inflation over the first five periods of the run. You can see that the value of this objective function is getting pretty close to zero.

If you click View Multipliers, you can see what effect a unit change in real interest rates have on the objective. As we have already noted, because wealth effects cancel out, the model is purely forward looking in terms of demand, so changes in real interest rates in the future influences inflation today, but not vice versa.

If you go to the data page, you will find that the basic run with the shock has been stored, and the optimisation run is in slot zero. You will see that in the optimisation run, inflation and output have both been returned to base. Note that the movement in real interest rates needed to do this is much larger than we get with the simple rule (and $par[5]=0.5$), but also its profile is different.

If you are doing further optimisation runs, remember to first delete the basic shock run from the store, to avoid repeating this each time. Crashes may be caused by having too many instruments in relation to objectives, causing collinearity.

6. Additional Exercises

See the assignment!!

Exercise 5

Introducing Compact. Complexity and Theoretical Deconstruction

1. *COMPACT – a Structural Econometric Model (SEM)*

Compact is a model that has to be compiled to run, because it contains one feature, a vintage model, that is outside the scope of Modelphi's interpreter. Load `cmp5_4`. You will immediately notice one major difference compared to our earlier models – its size. If you click on some of the equations, you will also notice that they can be more complex.

Compact uses actual data as part of its base, so here the base has to be stored as a file and loaded. As it uses real data, it is located in real time: the database runs from the beginning of 1902 for a few variables (those involved in the vintage model) and ends towards the end of the 21st century. The data period is quarterly, and !99902 stands for the second quarter of 1999.

2. *Residuals, forecasts and simulations*

As Compact uses actual data as part of its base, then residuals are a vital part of the simulation mechanism. By going to the data page and viewing residuals over the past, you can see the errors implied by the model's equations. These errors have to be added to the equations predictions as part of any simulation exercise.

The Compact base also includes a forecast, and the natural simulation period is this forecast period. Because Compact is a forward looking model, and to avoid terminal date problems, Compact's forecast has to cover a long period.

In an ideal world, a model's forecast would not involve any residuals: it would simply be the predicted values of each equation. However this is not realistic for two reasons in particular. First, historical errors are unlikely to be purely random, but may contain some autocorrelation. As a result, some short term residuals may be required to avoid a sharp discontinuity when the forecast begins. Second, past errors may indicate a more serious misspecification, such as a constant shift, which may imply the projection of flat, non-zero residuals into the future. (A key distinction is the order of integration of these errors.)

Compact was never used as a forecasting tool, and so the 'forecast base' used here comes from a slightly earlier version of the model, and there are many non-zero residuals. The base is also pretty old. We will use Compact as a simulation tool, in exactly the same way as the earlier models. Experiments suggest that Compact's simulation properties are fairly independent of the base used.

3. *The structure of Compact and complexity*

This is discussed in detail in Compact's model manual (click [Help+Manual](#)), as well as academic articles referenced there. This source is

important, because the model is large and elaborate, and because equations can be complex. However, there are some tips that can help

- i) Parts of equations may be multiplied by a zero parameter. These allow extensions to the model without the need to recompile, but can be ignored when analysing the published model.
- ii) Use the trace facility on the variables page to see where a variable is used.
- iii) Some equations have been transformed to get the level of the dependent variable on the left hand side e.g. difference equations, or logs (or both)
- iv) look for other simple transformations, like real/nominal variables, or share/levels. (In Compact, a number of variables are expressed as a share of GDP, and need to be multiplied by GDP to be used elsewhere.)

Modelphi also includes a device to simplify model code: dummy variables. These are essentially just labels for extra bits of code: they are used only while each equation is solved each period, and are undated.

4. Reasons for Complexity

(i) Good reasons

- (a) SEMs need to fit the data, so it is more difficult to abstract
- (b) Policy models often need to be a 'horse for all courses'

(ii) Dangers

Complexity makes the model more difficult to understand and use. Disaggregation may allow inconsistency (or more serious mistakes!)

(iii) Why SEMs must be understandable

- (a) Complexity may be the result of mistakes
- (b) Policy makers need to be able to tell a story. 'Trust the model' lacks credibility
- (c) There are alternatives to SEMs

5. Theoretical deconstruction: and an example

The idea that the theoretical properties of a model must be explicable is formalised in Wren-Lewis et al (1996), but had been used in an informal way by many others, including the Warwick Modelling Bureau team under Ken Wallis. The aim is to relate simulation properties of SEMs to basic theoretical models. The method is to gradually strip out complexity, until results fit with chosen theory model. The model is the 'rebuilt', noting how extra complexity causes changes in properties. It provides a complete account of the SEM's properties with basic theory model as the starting point.

Theoretical construction can be achieved by the methods we have deployed earlier to alter a model: exogenisation, parameter changes (e.g. setting parameters to zero), and editing the model itself. However, the process of deconstruction requires a good deal of knowledge: what is an

appropriate theoretical model or result to aim for, what order should we deconstruct, how do we remove theoretical elements from the model without unintended effects? In the process you may discover mistakes in the model (generally inconsistencies) - so theoretical deconstruction is an invaluable tool in model building.

As an example, consider a permanent shift in world prices. In a model where neutrality holds, and there is a freely floating exchange rate, the response should be an equal, permanent change in the nominal exchange rate. The reason is straightforward: with neutrality the model generates a real equilibrium, including a real exchange rate. A given real exchange rate is compatible with any nominal exchange rate and world price combination. If the nominal exchange rate only influences the model *through* the real exchange rate, then a x% change in world prices will lead to an x% change in the nominal exchange rate.

The job wp16 exogenises world prices, and increases them by 1%. Run the job, and plot the percentage change in world prices (wp) and the nominal effective exchange rate (ex). The nominal exchange rate increases, but not quite by 1%. To understand why, we have to play detective with the model, and deconstruct its response.

A first place to look is for real variables that change substantially. The initial change in consumption (cp) is very large. Comparing consumption with real disposable income (rydnp) suggests that income is not the driving force here. However, if we look at personal wealth (nfapc), this falls substantially at the start of the simulation. (This is a share of GDP, so look at absolute changes from base.) After a bit of work, we find that nfapc depends in part on net overseas assets (nar), and that the fall comes from here. Net overseas assets are determined by a budget constraint that contains two rates of return as dummy variables, and in one we find the change in the nominal dollar exchange rate (exdol).

Is this the source of the non-neutrality? To see, exogenise nar for the first period alone, and rerun the simulation. We get neutrality. So this was the additional complexity that accounted for the deviation from the standard theoretical result. Does it make sense?

6. Some useful details about Compact

- (i) Inertia in the wage equation is due to annual wage contracts (Par29)
- (ii) The degree of nominal inertia in prices is controlled by parameter Par51
- (iii) In COMPACT, income tax allowances adjust to return the debt to income ratio to base (TYP equation). Alternative forms of fiscal feedback are discussed in one of the model manual's topics
The speed of fiscal feedback is controlled by Par4.
- (iv) Monetary policy is controlled by the equation for interest rates. Real rates respond to inflation less target (Par13), but also past real rates (Par14). You can change either parameter, but try small changes first - many parameter combinations will lead to non-convergence or crashes
- (v) Consumption in COMPACT involves Blanchard/Yaari with credit constrained consumers. Additional discounting of income (the 'mark-

up' on real rates) depends on the degree of credit liberalisation (CCC). CCC is a transformation of FLIB, an exogenous variable denoting financial liberalisation.

(vi) There are four sectors in COMPACT, and each has a financial asset to GDP ratio variable:

na net overseas assets

nfacr companies, the discounted sum of expected future company cash flows

nfapr consumers

nfagr government

The following identity holds:

$na = nfacr + nfapr - nfagr$

7. Additional exercises

The model manual contains a simulation section with some suggestions for deconstruction. One that is certainly worth exploring is Ricardian Equivalence. However, first look at a permanent increase in income taxes: see the job `trinc16`. This produces a very surprising result! To see why, read the model manual, and the change `Par47`. You can also alter credit constraints by setting `par25=0.1`.

Exercise 6

Investment, and Tobin's q . Vintage models. Aggregate trade equations

1. Investment, and costs of adjusting capital

In the main macro course, the optimal quantity of capital was related to the real interest rate (as the cost of capital). There were no costs of adjusting capital, so firms simply invested enough to give them their desired capital stock each period.

A separate note looks at the optimisation problem the firm faces if there are costs in adjusting investment. It introduces Tobin's q .

In Compact, and in most empirical studies, q is proxied by the ratio of the market value of the firm to its 'book value' (the value of the capital stock). In equilibrium this variable will be unity. However the market value of the firm is a forward looking variable, and depends on future profits. Thus, when a profitable investment opportunity arises, expectations of future profits increase, and so the value of q increases. From the theory, we can see that it is optimal for the firm to increase investment, thereby raising the capital stock (and reducing q back to one).

2. Additional complexity in Compact: bankruptcy risk

Load Compact, and its base. If you look at the variable AQ, you can see the market value of the company sector ($nfacr$) and the capital stock (kap). The main investment variable is 'ino' (non-oil, non-housing). The ratio of ino to the capital stock depends on AQ. However, there is an additional term in 'liqr'. This is rate of company liquidations, and proxies bankruptcy risk.

The idea behind this modification is that managers of the firm will not just maximise shareholder value, but will give additional weight to the chance of bankruptcy. Greenwald and Stiglitz, 1993, discuss why this might occur: problems of information mean that owners of firms cannot tell whether bankruptcy is due to bad luck or incompetent managers.

In Compact, liqr depends on the level of debt relative to the market value of the firm i.e. its gearing. Debt builds up when a negative shock hits the company sector. This introduces an important lagged dynamic into the model. However a note of caution is appropriate here: this element of the model is relatively novel and is not based on clear microfoundations.

3. A world demand shock in Compact

The job $wtp16$ decreases world output and trade by 1% for a year. Run it. First, plot wta (world trade) and y (output). (To see the plots better, change the end date by holding down the Ctrl key and left clicking on a new end date.) This is much as we would expect – the interest is in the detail.

First, note that the end of the simulation is wobbly. To see why, go to the job page, and click the Properties button to see the job's properties. Add an extra couple of zeros into the 'absconv' property. Store the original job, and rerun. Now the results make more sense. We clearly had not got to a true

solution in the first run. But notice also that the solution for the first few decades is largely unaffected.

The simulation illustrates consumption smoothing. However, we want to investigate liquidity effects on investment. We saw from the investment equation that these entered via `par54`. So set this to zero, and look at `ino` in the two runs.

4. The vintage model

A unique feature of Compact is its vintage production technology. The details are discussed in the model manual. The central consequence of the vintage technology is that both factor mix (the capital/labour ratio) and the level of technical progress (in this case labour embodied) are embodied in machines one they are built. Therefore, to change the factor mix, or improve the technology, firms have to invest in new machines – they cannot adapt old ones.

This ‘putty/clay’ assumption is extreme – it is possible to ex post change factor mix or technology on many capital goods (e.g. buildings). However the standard assumption of ‘putty/putty’ is also unrealistic. The interesting question here is what difference a vintage technology makes.

Some implications are obvious. First, a technology shock will take time before it is fully embodied in the capital stock, and therefore output. As a result, a shock to technology will be much longer lasting. Compare two jobs, both of which represent the same improvement in technology. One, `tp15`, represents just that, but `tp&g&w_putty` makes some additional changes to deconstruct the model. The model contains a parameter, `Par30`, that replaces the vintage model with a simple putty/putty alternative. However the job also contains some other changes. One of these involves increasing world output and trade. Without it, the UK’s real exchange rate has to depreciate to sell the additional output made possible by the technical change. In Compact, the real exchange rate influences the cost of capital, which has a permanent supply side effect. By looking at mid-way stages between these two jobs, you can see what matters where. We explore this further in the next simulation.

5. Labour supply shift

In Compact labour supply is independent of the real wage, which means that Compact is not a real business cycle model. This is clearly demonstrated if we increase the labour force by 1% - job `pop16`.

The most notable feature are the long lags before output rises by (almost) 1%. Unemployment rises in the short term – not the `rbc` result at all! The reasons for this come from a property of the vintage model, and are fairly complex (and non-standard) – see the model manual for details. Once again output does not rise by the full 1%, and the real exchange rate depreciates.

Actually this result would be a little stronger if it were not for another unusual feature of Compact – its trade equations. In the macro lectures, we noted that one criticism of the result based on imperfect competition that an increase in home supply would require a depreciation was that it assumed that the type of goods produced remained unchanged. Compact tries to meet that criticism by including cumulated investment in both the aggregate export

and import equations, as a proxy for the variety and quality of goods produced in the UK.

We can deconstruct this effect by exogenising the variable INOC. However, we also need to change the model's damping to ensure convergence: change damping on PYNOC from 0.05 to 0.02. Now run the same simulation, and observe the real exchange rate, output and investment. (This 'trick' may work for other simulations that fail to converge.)

6. Additional exercises

Explore the supply side of Compact further.