

DIY Macroeconomic Model Simulation

Table of contents

Welcome	3
Project team	3
Contact	3
License	5
Acknowledgements	5
1 Getting Started	6
1.1 Structure of platform	6
1.2 Access and introduction to <i>R</i> and <i>Python</i>	6
1.3 Simple exercise	14
2 How to Simulate Economic Models	15
2.1 Introduction: economic models	15
2.2 Solving economic models numerically	17
2.2.1 Solving economic models numerically: examples	17
2.3 How to plot the results of a model	29
2.4 How to create a directed graph of a model	36
2.5 Appendix: How to simulate dynamic model in continuous time	39
2.6 References	41
I Static Models	42
3 A Neoclassical Macro Model	43
3.1 Overview	43
3.2 The Model	43
3.3 Simulation	45
3.3.1 Parameterisation	45
3.3.2 Simulation code	47
3.3.3 Plots	52
3.4 Directed graph	58
3.5 Analytical discussion: derivation of behavioural functions	62
3.5.1 The firm's problem: profit maximisation	62
3.5.2 The government's budget constraint	63
3.5.3 The household's problem: intertemporal utility maximisation and Ricardian Equivalence	63

References	65
4 An IS-LM Model	66
4.1 Overview	66
4.2 The Model	66
4.3 Simulation	68
4.3.1 Parameterisation	68
4.3.2 Simulation code	69
4.3.3 Plots	74
4.4 Directed graph	79
4.5 Analytical discussion	84
4.5.1 Calculate equilibrium fiscal multiplier	84
References	85
5 A Neoclassical Synthesis Model (IS-LM-AS-AD)	86
5.1 Overview	86
5.2 The Model	87
5.3 Simulation	88
5.3.1 Parameterisation	88
5.3.2 Simulation code	89
5.3.3 Plots	95
5.4 Directed graph	100
5.5 Analytical discussion	105
References	106
6 A Post-Keynesian Macro Model with Endogenous Money	107
6.1 Overview	107
6.2 The Model	108
6.3 Simulation	110
6.3.1 Parameterisation	110
6.3.2 Simulation code	110
6.3.3 Plots	116
6.4 Directed graph	123
6.5 Analytical discussion	127
References	128
7 A Kaldor-Robinson Distribution and Growth Model	129
7.1 Overview	129
7.2 The Model	129
7.3 Simulation	130
7.3.1 Parameterisation	130
7.3.2 Simulation code	131
7.3.3 Plots	135

7.4	Directed graph	137
7.5	Analytical discussion	141
7.5.1	Calculate analytical solutions numerically	141
	References	142
8	A Post-Kaleckian Distribution and Growth Model	143
8.1	Overview	143
8.2	The Model	143
8.3	Simulation	145
8.3.1	Parameterisation	145
8.3.2	Simulation code	145
8.3.3	Plots	151
8.4	Directed graph	154
8.5	Analytical discussion	158
8.5.1	Calculate analytical solutions numerically	159
	References	160
II	Dynamic Models	161
9	An Introduction to the Analysis of Dynamic Models	162
9.1	Solution of a single first-order linear difference equation	162
9.2	Solution of a linear system of difference equations	164
9.3	Complex eigenvalues and cycles	173
9.4	Nonlinear systems	181
9.5	Key takeaways	182
9.6	References	182
10	A New Keynesian 3-Equation Model	183
10.1	Overview	183
10.2	The Model	183
10.3	Simulation	184
10.3.1	Parameterisation	184
10.3.2	Simulation code	185
10.3.3	Plots	189
10.4	Directed graph	192
10.5	Analytical discussion	197
10.5.1	Derivation of core equations	197
10.5.2	Equilibrium solutions and stability analysis	198
10.6	References	200
11	A Sraffian Supermultiplier Model	201
11.1	Overview	201

11.2 The Model	202
11.3 Simulation	203
11.3.1 Parameterisation	203
11.3.2 Simulation code	203
11.3.3 Plots	208
11.4 Directed graph	212
11.5 Analytical discussion	216
11.6 References	218
12 A Malthusian Model	219
12.1 Overview	219
12.2 The Model	219
12.3 Simulation	220
12.3.1 Parameterisation	220
12.3.2 Simulation code	220
12.3.3 Plots	225
12.4 Directed graph	229
12.5 Analytical discussion	232
12.6 References	234
13 A Ricardian One-Sector Model	235
13.1 Overview	235
13.2 The Model	235
13.3 Simulation	236
13.3.1 Parameterisation	236
13.3.2 Simulation code	237
13.3.3 Plots	242
13.4 Directed graph	247
13.5 Analytical discussion	251
13.6 References	257
14 A Ricardian Two-Sector Model	258
14.1 The Model	258
14.2 Simulation	260
14.2.1 Parameterisation	260
14.2.2 Simulation code	261
14.2.3 Plots	267
14.3 Directed graph	276
14.4 Analytical discussion	281
14.5 References	287
15 A Lewis Model of Economic Development	288
15.1 Overview	288

15.2 The Model	288
15.3 Simulation	290
15.3.1 Parameterisation	290
15.3.2 Simulation code	290
15.3.3 Plots	295
15.4 Directed graph	301
15.5 Analytical discussion	305
15.6 References	306
Additional Online Resources	307
Economic Modelling in General	307
Stock-Flow Consistent Modelling	307
Agent-Based Modelling	307
Coding	307

Welcome



Warning

This website is under construction and will be regularly updated and extended.

This platform provides an open source code repository and online script for macroeconomic model simulation. It follows a “do-it-yourself” (DIY) approach, empowering users to numerically simulate key macroeconomic models on their own using the open-source programming languages *R* and *Python*. Whether you are a university teacher, student, researcher, or an economics enthusiast, our platform offers resources to deepen your understanding of both macroeconomic modelling and coding.

The platform covers an array of macroeconomic models, including canonical textbook models, models from different economic paradigms, and seminal models from the history of economic thought. It bridges a gap between intermediate and advanced level macroeconomics by providing detailed yet accessible treatments of seminal macroeconomic models. Most intermediate macroeconomics textbooks focus on graphical analysis, while advanced level materials are often more mathematical and less accessible. Our platform offers a hands-on and approachable resource for users to build both a solid foundation in modelling and macroeconomic intuition.

The platform’s DIY-approach aims to foster reproducibility and open-source principles in macroeconomic research and education by providing learning materials that are freely available and modifiable by everyone. The platform’s content will expand over time through new entries added by the project team. The Python extension is currently under construction (available for chapters 1-10).

Project team

Contact

We welcome any feedback. If you encounter any issues, find typos or mistakes, or have questions/thoughts on the content, please do get in touch.

1. If you are Github user, you can [report an issue](#) in our repository
2. You can also email us directly: franz.prante@wiwi.tu.chemnitz.de and k.kohler@leeds.ac.uk



Figure 1: Franz Prante

Figure 2: Karsten Kohler

Franz is a research associate at Chemnitz University of Technology, where he is currently working on the macroeconomic effects of monetary policy and price effects on energy demand. He is also a PhD student at Université Sorbonne Paris Nord.

Karsten is a Lecturer in Economics at Leeds University Business School, where he does research on the interaction between finance and the real economy, especially sources of cyclical dynamics, instability, and rising inequality.

License

The material on this page is licensed under [CC BY-NC 4.0](#), thus feel free to use the material for non-commercial purposes but please give credit.

Acknowledgements

We are grateful for helpful comments from Adam Aboobaker, Chandni Dwarkasing, Giuseppe Fontana, Alex Guschanski, Eckhard Hein, and Rafael Wildauer on various sections of this website. All errors are ours.

1 Getting Started

1.1 Structure of platform

The platform starts off with a general introduction the numerical simulation of economic models (Chapter 2).

After that, it jumps right into a series of macroeconomic models. These models are grouped into static and dynamic models. In static models, time plays no role and all variables adjust instantaneously. By contrast, dynamic models characterise the adjustment of variables over time.

The model entries are largely self-contained and can be read independently of each other. For each model, the chapters provide three main components:

1. **Model descriptions** that concisely explain the key ideas, assumptions, and equations of each model. This helps users grasp the underlying concepts and intuition behind the models.
2. **Annotated code** that allows users to numerically simulate the models, examine their results under different scenarios, and produce visualisations to better understand the models' structure and output. This hands-on approach enables users to gain practical coding skills while exploring different macroeconomic theories.
3. **Analytical discussions** for users who are interested in delving deeper into the mathematical properties of the models.

To further facilitate the understanding of dynamic models, Chapter 9 of the section on dynamic models begins with a general introduction into the mathematical analysis of dynamic models (this is mostly relevant for the analytical discussions of dynamic models).

1.2 Access and introduction to *R* and *Python*

All simulation codes are written in the open-source programming language *R*. A Python extension is under construction and currently available for chapters 1 - 10. To be able to manipulate the codes on this platform on your own machine, you first need to [download and install *R* and *RStudio*](#). For Python, there are different options. One of them is to download and install

Spyder via the [Anaconda Python distribution](#). *Spyder* provides an interface for *Python* (like *RStudio* for *R*). If you install it via *Anaconda*, it will install *Python* automatically.

Besides being free, a key advantage of both *R* and *Python* is their huge and growing functionality due to new user-written libraries and packages continuously being added. In addition, a large amount of learning material is freely available on the web, e.g. [here](#) and [here](#) for *R* and [here](#), and [here](#) for *Python*. However, to get started in can be best to directly delve into some of the codes on this platform and learn by doing. To this end, the following “cheatsheets” that provide a concise overview of key functions are useful:

- [R Studio Cheatsheet](#)
- [Base R Cheatsheet](#)
- [more R cheatsheets here](#)
- [Python Basics Cheatsheet](#)
- [Python for Data Science Cheatsheet](#)
- [Python Cheatsheet for NumPy library](#)

Once you have installed *R* or *Python*, you can play with the codes on this platform yourself by copy and pasting them into the script panel of your local interface (IDE) and hitting CTRL + Enter to execute them. Don’t forget to always comment your code using the “#” symbol and to save your scripts to make sure your future self can seamlessly continue working on it.

The codes below covers some basic operations.¹. If you are new to *R* or *Python* and keen to get started, do the following:

- copy the codes below into the script panel of *RStudio* or *Spyder* on your machine
- adjust the working directory to your personal folder
- then run the code
- make sure you understand what it does

```
##### R Basics #####
#In the R script you write code and comments
#any line starting with # is a comment and it is NOT executed

## First things first: set the working directory to the the folder in which your R files a
# note that you need to separate folders by slashes /

#Let's define some variables
```

¹We are grateful to Rafael Wildauer for permission to reproduce a slightly modified version of his learning materials. In addition, some of the material below is taken from [here](#).

```

s = 0.05
Y = 10

#We can also assign several variables the same value at once
C = I = R = 2

#For displaying them we simply call the name of the object and execute the relevant line
Y
C

#We can define new variables using existing ones
W = Y - R
W

#R has a vast amount of built in functions, for example
max(10,2,100,-3)
sqrt(9)
abs(-13)

#you can find out more about these by using the help function, e.g.:?max()

#How can you delete stuff? Use the remove function rm()
#for individual objects
rm(Y)
#if you want to remove everything
rm(list=ls(all=TRUE))

#You can also assign text (a string) to a variable
text1 = 'Reggaeton'
text2 = "Bad Bunny"
#note "text" is the same as 'text' and a string can contain spaces
#You use the paste function to combine strings
paste(text1, text2)

#R uses standard operators like +, -, *
#for exponents use ^
3^2

#####
##### if statements and loops #####
#####

#sometimes we need to introduce if conditions into our code

```

```

#The syntax is
#if condition { do something }
a = 10
b = 13
if (b>a) {
  print("b is bigger")
}

#we can also tell R what to do in case the condition is not fulfilled
b=10
if (b>a) {
  print("b is bigger")
} else {
  print("b not bigger")
}

#Next we will look at loops which are a key tool to repeat tasks such as solving
#a theoretical model again and again to find its equilibrium.
#The basic structure is:
for (i in 1:5){
  print(i)
}

#Let's use it to solve a simple Keynesian cross model of the form
#Y=C+I
#C=c0+c1Y
#Define exogenous parameters
c0=2
c1=0.8
I=10
#set initial values for two endogenous variable
Y=C=1
#Use a for loop to solve it
C
Y
for (i in 1:100){
  Y = C+I
  C = c0+c1*Y
}
C
Y

```

```

#solution is Y=(I+c0)/(1-c1)=60

#What is special about this loop is that it uses the values from the previous iteration
#to define the values of the next, because it starts with assigning a value to Y
#and then uses that value to assign a new value to C and in the next iteration it
#uses this new value for C to define a new value for Y etc.

#####
# Data structures #####
#####

#In most applications we produce outputs which do not consist of a single number.
#Often we have an entire stream of results, or we want to analyse data and have to store lots of numbers.
#R has a variety of data structures for this purpose.

#let's clean up first
rm(list=ls(all=TRUE))

#####Vectors
#We can create an empty vector and fill it later (with results of our model for example)
vec1 = vector(length=3)
#we can define vectors explicitly using the c() function (c for column?)
vec2 = c(1,2,3)
vec3 = c(6,7,8)
#we can also use the sequence operator
vec4 = 1:10
#and we can define the step size
vec5 = seq(1,2,0.1)
vec5
#we can call specific entries using square brackets
vec5[4]
#if we want to access more elements at once
vec5[c(4,1)]


#####Matrices
#define a matrix: 3 rows and 2 columns, all elements equal to 0
mat1 = matrix(0, nrow=3, ncol=2)
mat1
#we can also fill it with specific values
mat2 = matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2)
mat2
#access specific elements (columns,rows)
mat2[3,1]
#access entire rows or columns

```

```
mat2[,1]
mat2[1,]
#access sub matrices
mat2[c(1,2),]
## Combine two column vectors in a matrix
mat3=cbind(vec1, vec2)
mat3
## Combine two row vectors in a matrix
mat4=rbind(vec1, vec2)
mat4
```

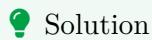
 Python code

1.3 Simple exercise

If you further want to practice your coding skills, attempt the following exercises:

1. Write a loop which calculates the running sum of $x_i = \frac{1}{i^2}$ by saving each element $\sum_{i=1}^j x_i$ in a vector (where $j = 1, 2, \dots, 10$).

If your code is correct, you should get the solution:

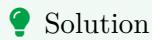


Solution

```
[1] 1.000000 1.250000 1.361111 1.423611 1.463611 1.491389 1.511797 1.527422  
[9] 1.539768 1.549768
```

2. Write a loop which calculates the running sums of $x_i = \frac{1}{i^2}$ (as in exercise 1) and in addition also $y_i = \frac{1}{i^3}$. Combine the results in a matrix.

If your code is correct, you should get the solution:



Solution

	x	y
[1,]	1.000000	1.000000
[2,]	1.250000	1.125000
[3,]	1.361111	1.162037
[4,]	1.423611	1.177662
[5,]	1.463611	1.185662
[6,]	1.491389	1.190292
[7,]	1.511797	1.193207
[8,]	1.527422	1.195160
[9,]	1.539768	1.196532
[10,]	1.549768	1.197532

2 How to Simulate Economic Models

2.1 Introduction: economic models

Why do we build formal economic models? Because they help us think carefully about the *causal mechanisms* that generate certain economic outcomes (e.g. unemployment). Models are especially useful when the variables of interest are inherently quantitative (e.g. the interest rate and unemployment) and when several of these quantitative variables interact with each other. More specifically, models provide a precise formal representation of a set of *interlinked* causal mechanisms that are often difficult to analyse informally.

Every economic model essentially consists of three things:

1. a set of N equations
2. a set of N endogenous variables
3. a set of exogenous or fixed coefficients ('parameters') and exogenous variables

The solution to the model, its 'equilibrium', will pin down values for the endogenous variables of the model for a given set of parameters and exogenous variables. Thus, the endogenous variables, e.g. unemployment, are determined within the system, while exogenous variables are determined outside of the system and often reflect policy variables, such as the central bank interest rate. The equations connect the variables of the system. They typically express:

- economic (accounting) identities (e.g. that in a closed economy without government, saving is income that is not consumed: $S = Y - C$)
- budget constraints (e.g. that business investment can be financed out of retained profits and new debt, $I = \Pi + \Delta D$)
- behavioural functions (e.g. that households consume a constant proportion of their income, $C = c_1 Y$), which often contain key parameters of the model (e.g. the marginal propensity to consume c_1)
- equilibrium conditions (e.g. that demand must be equal to supply)

Equations may be linear or nonlinear. If a model contains nonlinear relationships between the endogenous variables, it may admit more than one solution (often called multiple equilibria).

Economic models can be either static or dynamic. In a static model, time plays no role and all endogenous variables are determined simultaneously. In a dynamic model, time matters and the endogenous variables adjust gradually over time.

The endogenous variables are typically interrelated: e.g. x determines y , but y also determines x . These interrelationships can be:

- simultaneous: x and y determine each other simultaneously (within the same period)
- recursive: x affects y only in $t + 1$ (or vice versa)

Unlike static models, dynamic models describe what happens out of equilibrium.¹ Note that dynamic models may contain both simultaneous and recursive relationships.

Whether the relationships between the variables is simultaneous or recursive has implications for how the model can be solved. In general, simple economic models can often be solved analytically without a computer. If the model contains simultaneous relationships, it needs to be solved as a simultaneous system by solving for the endogenous variables through repeated substitution. This means going from the so-called ‘structural form’, i.e. the full set of equations, to the so-called ‘reduced form’, where the right-hand side of the equations only contains exogenous variables and parameters. If the system is linear, techniques from linear algebra such as matrix inversion or Cramer’s rule can be used (see Chiang and Wainwright (2005), chaps. 4-5). If the system contains recursive relationships, the equilibrium solution can be found by setting $x_{t+i} = x_{t-i} = x_t$ for all x and then solving the resulting simultaneous system (more on this Chapter 9).

However, often a complete analytical solution is difficult to come by. Common challenges are:

- a model has more than 3 dimensions ($N > 3$): then it’s very tedious to compute analytical solutions
- a model has nonlinearities that preclude the computation of analytical solutions
- a model is dynamic and you want to examine the dynamic adjustment of the endogenous variables (which is tedious to do analytically)

In these cases, numerical solution by means of computer simulation becomes useful. A key advantage is that it allows you to study much more complex models than the analytical approach does. A key disadvantage is that numerical solution requires the choice of a (possibly arbitrary) set of numerical values for the models’ parameters. It is thus less general than

¹That raises the question of whether an equilibrium is stable or unstable, which is discussed in Chapter 9.

analytical solution – a limitation which should be borne in mind. We think that both analytical and numerical approaches are useful. Correspondingly, we supplement the numerical simulations with analytical model solutions where possible.

2.2 Solving economic models numerically

If a (dynamic) model exclusively contains recursive relationships, it can be solved iteratively by sequentially updating the endogenous variables from (arbitrarily set) initial conditions. This is easy to do with a computer. By contrast, if interrelationships are simultaneous, solving the system for the endogenous variables as described above is less trivial for a computer (finding the solution for x requires the solution of y , but the latter requires in turn the solution for x). One approach is to use linear algebra: cast the system in matrix form ($b = Ax$) and let the computer find $x^* = A^{-1}b$ through some algorithm (e.g. [the Gauss-Seidel method](#)).

We will use an approach that is simpler and based on *iteration*:

- choose a set of numerical parameter values (e.g. $c_1 = 0.8$)
- choose (arbitrary but non-zero) initial values for the endogenous variables (e.g. $C_0 = 1$)
- then solve the system of equations many times using a *for loop*

In this way, the solution gets approximated successively.

A limitation of the method of iteration is that it will only converge to the solution of the simultaneous component of a model if the equilibrium is stable (more on stability in Chapter 9). For most static models, stability is required for the model to be economically meaningful. In that sense, if the iterative approach does not yield a solution, this is a sign that the model and/or parameterisation needs to be reconsidered. In dynamic models, stability is a key question that should be addressed in any case. Therefore, this limitation of the method of iteration may not be too restrictive in practice.

2.2.1 Solving economic models numerically: examples

2.2.1.1 A static model

Consider a two-dimensional simultaneous system represented by a simple Keynesian goods market model:

$$Y = C + I_0$$

$$C = c_0 + c_1 Y$$

In a closed economy without government, aggregate demand is composed of consumption C and investment I , with the latter assumed to be exogenous. Goods market equilibrium requires aggregate demand to be equal to aggregate income Y . Consumption is assumed to be determined by an autonomous component c_0 and a marginal propensity to consume out of income c_1 .

Suppose the parameters are given by $c_0 = 3$ and $c_1 = 0.8$. You are interested in how a change in investment from $I_0 = 5$ to $I_0 = 6$ affects the solution of the system. Through the method of substitution, we can easily derive that $Y^* = \frac{c_0 + I_0}{1 - c_1}$. The code below shows how to find this solution via simulation.

```
### Simulate Keynesian goods market model via iteration

#Clear the environment
rm(list=ls(all=TRUE))

# Set number of parameterisations that will be considered
S=2

# Set fixed parameter values
c0=3
c1=0.8

#Create vector in which equilibrium solutions from different parameterisations will be stored
Y_eq=vector(length=S)
C_eq=vector(length=S)

#Create vector with parameter that will change
I0=vector(length=S)
I0[1]=5
I0[2]=6

# Initialise endogenous variables at arbitrary positive value
Y=C=1

#Solve this system numerically through 1000 iterations based on the initialisation
for (i in 1:S){
  for (iteration in 1:1000){
    Y = C + I0[i]
    C = c0 + c1*Y
  } # close iterations loop
```

```
#Save results for different parameterisations in vector  
Y_eq[i]=Y  
C_eq[i]=C  
} # close parameterisations loop  
  
# Display solutions  
Y_eq
```

```
[1] 40 45
```

```
C_eq
```

```
[1] 35 39
```

```
# Verify solutions for Y  
(c0+I0[])/(1-c1)
```

```
[1] 40 45
```

 Python code

```

### Simulate Keynesian goods market model via iteration

# Load NumPy library
import numpy as np

# Set the number of parameterisations that will be considered
S = 2

# Set fixed parameter values
c0 = 3
c1 = 0.8

# Create numpy arrays in which equilibrium solutions from different parameterisations wi
Y_eq = np.zeros(S)
C_eq = np.zeros(S)

# Create a numpy array with the parameter that will change
I0 = np.zeros(S)
I0[0] = 5
I0[1] = 6

# Initialize endogenous variables at an arbitrary positive value
Y = C = 1

# Solve this system numerically through 1000 iterations based on the initialization
for i in range(S):
    for iteration in range(1000):
        Y = C + I0[i]
        C = c0 + c1 * Y

    # Save results for different parameterisations in the numpy arrays
    Y_eq[i] = Y
    C_eq[i] = C

# Display solutions
Y_eq
C_eq

# Verify solutions for Y
(c0+I0)/(1-c1)

```

Let's break this code down a little bit:

- set the number of scenarios S , define parameter values, and create vectors of length S in which results for endogenous variables will be stored
- define changes in exogenous variables or parameters (i.e. construct different scenarios)
- initialise the endogenous variables
- write down the equations (solved for the endogenous variables such that every endogenous variable of the system appears on the left-hand side of an equation exactly once)
- place these equations inside a for loop
 - the loop says: repeat the segment of code insights the curly brackets 1000 times
- nest the loop that solves the system in an outer loop that loops through different parameterisations (here for I_0)
- after the iterations loop is finished, save the results for the current parameterisation

What happens is the following: in the first iteration, Y and C are calculated based on the initial values and the parameter values. In the second iteration, the values are then overwritten based on the results from the first iteration. This process continues 1000 times. In this way, the correct solution is successively approximated. If you have an analytical solution, you can compare it with the numerical one to double-check your results.

When does the method of iteration fail to provide a solution? In the following code, everything is the same with the only difference that the marginal propensity to consume is now larger than unity ($c_1 = 1.2$). Now the numerical simulation fails to find the solution. This is because with $c_1 > 0$, the so-called Keynesian equilibrium condition is violated and the system happens to be unstable. We will discuss in Chapter 9 the issue of stability and how to analyse it formally. At this point, we simply note that the method of iterations requires static models to be stable to yield equilibrium solutions.

```
### Parameterisation for which method of iteration fails

#Clear the environment
rm(list=ls(all=TRUE))

# Set number of parameterisations that will be considered
S=2

# Set fixed parameter values
c0=3
c1=1.2
```

```

#Create vector in which equilibrium solutions from different parameterisations will be stored
Y_eq=vector(length=S)
C_eq=vector(length=S)

#Create vector with parameter that will change
I0=vector(length=S)
I0[1]=5
I0[2]=6

# Initialise endogenous variables at arbitrary positive value
Y=C=1

#Solve this system numerically through 1000 iterations based on the initialisation
for (i in 1:S){
  for (iteration in 1:1000){
    Y = C + I0[i]
    C = c0 + c1*Y
  } # close iterations loop

  #Save results for different parameterisations in vector
  Y_eq[i]=Y
  C_eq[i]=C
} # close parameterisations loop

# Display solutions
Y_eq

```

[1] 5.818655e+80 8.832196e+159

```

# Verify solutions for Y
(c0+I0[])/(1-c1)

```

[1] -40 -45

Python code

```
### Parameterisation for which method of iteration fails
c1 = 1.2

# Initialize endogenous variables at an arbitrary positive value
Y = C = 1

# Solve this system numerically through 1000 iterations based on the initialization
for i in range(S):
    for iteration in range(1000):
        Y = C + IO[i]
        C = c0 + c1 * Y

    # Save results for different parameterisations in the numpy arrays
    Y_eq[i] = Y
    C_eq[i] = C

# Display solutions for Y_eq
Y_eq

# Verify solutions for Y
(c0+IO)/(1-c1)
```

2.2.1.2 A dynamic model (in discrete time)

Consider now a dynamic version of the Keynesian goods market model that was proposed by Paul Samuelson (1939). In this model, investment (I) becomes endogenous and reacts to the change in consumption. Aggregate demand now also contains government spending (G), which is assumed to be exogenous. Consumption (C) responds to changes in income (Y) with a lag:

$$\begin{aligned}Y_t &= C_t + I_t + G_0 \\I_t &= \beta(C_t - C_{t-1}) \\C_t &= c_1 Y_{t-1}.\end{aligned}$$

This is a dynamic model, in which the endogenous variables adjust gradually over time. However, the model is not purely recursive as investment reacts to consumption in the same period (and output to consumption and investment). By shifting $Y_t = C_t + I_t + G_0$ one period back,

substitution into the consumption, and then investment function, the system can be reduced to two equations that are fully recursive:

$$C_t = c_1(C_{t-1} + I_{t-1} + G_0)$$

$$I_t = \beta[c_1(C_{t-1} + I_{t-1} + G_0) - C_{t-1}]$$

We can find the solution for output analytically by setting $x_t = x_{t-1}$ for all variables and then applying the method of substitution. This yields $Y^* = \frac{G_0}{1-c_1}$. The code below shows how to find this solution through simulation.

```
### Simulate Samuelson 1939

#Clear the environment
rm(list=ls(all=TRUE))

# Set number of periods for which you want to simulate
Q=100

# Set number of parameterisations that will be considered
S=2

# Set period in which shock or shift in an will occur
s=15

# Set fixed parameter values
c1=0.8
beta=0.6

# Construct (S x Q) matrices in which values for different periods will be stored; initial
C=matrix(data=1, nrow=S, ncol=Q)
I=matrix(data=1, nrow=S, ncol=Q)

#Construct matrices for exogenous variable or parameter that will change over time to capt
G0=matrix(data=5, nrow=S, ncol=Q)

# Set parameter values for different scenarios
G0[2,s:Q]=6      # scenario: permanent increase in government spending from I0=5 to I0=6 fr

#Solve this system recursively based on the initialisation
for (i in 1:S){
  for (t in 2:Q){
    C[i,t] = c1*(C[i,t-1] + I[i,t-1] + G0[i,t-1])
```

```
I[i,t] = beta*(c1*(C[i,t-1] + I[i,t-1] + G0[i,t-1]) - C[i,t-1])
} # close time loop
} # close scenarios loop

# Calculate output
Y=C+G0+I

# Display solution
Y[,Q]
```

```
[1] 25 30
```

```
# Verify solutions for Y
(G0[,Q])/ (1-c1)
```

```
[1] 25 30
```

 Python code

```

### Simulate Samuelson 1939

# Set the number of periods for which you want to simulate
Q = 100

# Set the number of parameterisations that will be considered
S = 2

# Set the period in which a shock or shift in 'an' will occur
s = 15

# Set fixed parameter values
c1 = 0.8
beta = 0.6

# Construct (S x Q) matrices in which values for different periods will be stored; initialise at 5
C = np.ones((S, Q))
I = np.ones((S, Q))

# Construct matrices for exogenous variables or parameters that will change over time to capture different scenarios, initialise at 5
G0 = np.ones((S, Q))*5

# Set parameter values for different scenarios
G0[1, s:Q] = 6 # scenario: permanent increase in government spending from I0=5 to I0=6

# Solve this system recursively based on the initialization
for i in range(S):
    for t in range(1, Q):
        C[i, t] = c1 * (C[i, t - 1] + I[i, t - 1] + G0[i, t - 1])
        I[i, t] = beta * (c1 * (C[i, t - 1] + I[i, t - 1] + G0[i, t - 1]) - C[i, t - 1])

# Calculate output
Y = C + G0 + I

# Display the solutions at time Q
Y[:, Q - 1]

# Verify solutions for Y
(G0[:, Q - 1])/(1-c1)

```

The code solves the recursive system numerically through iteration. Let's again break down what the code does:

- set the number of periods for which we want to simulate the model (here $Q = 100$), set the number of scenarios S and the period s in which a change in the scenario should occur
- set the parameter values and create $(S \times Q)$ matrices in which results for endogenous variables will be stored, such that the columns represent time and the rows represent different scenarios; initialise the endogenous variables
- define changes in exogenous variables or parameters (i.e. construct different scenarios)
- write down the equations (solved for the endogenous variables such that every endogenous variable of the system appears on the left-hand side of an equation exactly once)
- place these equations inside a for loop that runs from $t = 2$ to Q
 - the loop says: repeat the segment of code insights the curly brackets, each time with the index number t shifted by +1 until $t = Q$
- nest the loop that solves the system in an outer loop that loops through different parameterisations (here for G_0)

Note that due to the exclusive presence of recursive equations, no loop is need that iterates the equations within every period. However, many dynamic models contain both simultaneous and recursive equations. In this case, a third loop inside the time loop is needed that iterates the equations within each period so as to solve the simultaneous equations. Otherwise, the approach to solving dynamic systems is not fundamentally different from the one for static systems.

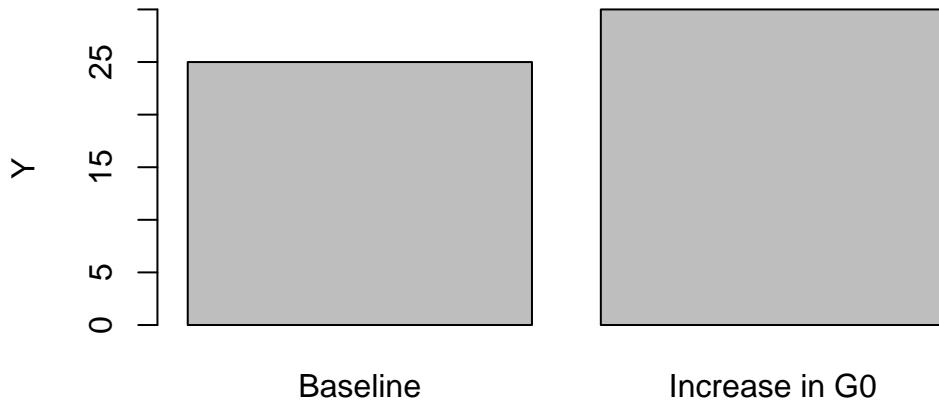
At the end of the simulation run, we can again compare the results from the numerical simulation with the analytical solution. However, what is perhaps more interesting is to trace the dynamic adjustment of the endogenous variables towards equilibrium. The best way to examine this is by plotting the results.

2.3 How to plot the results of a model

We will consider two main ways to plot model outputs: bar charts for static models and time series charts for dynamic models. Bar charts compare the equilibrium values of Y_t for different parameterisations:

```
# Bar chart of different equilibrium solutions of Samuelson (1939) model
barplot(Y[,Q], ylab="Y", main="Figure 1: Output", names.arg=c("Baseline", "Increase in G0"))
```

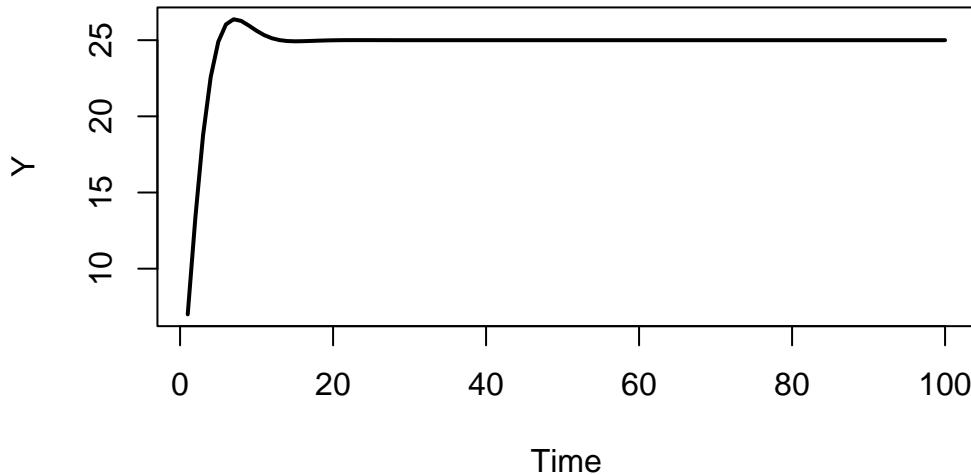
Figure 1: Output



Next, we show a basic version of a time series chart that displays the dynamics of Y_t :

```
# Time series chart of output dynamics in Samuelson (1939) model
plot(Y[1, 1:100], type="l", col=1, lwd=2, lty=1, xlab="Time", ylab="Y")
title(main="Figure 2: Output", cex=0.8)
```

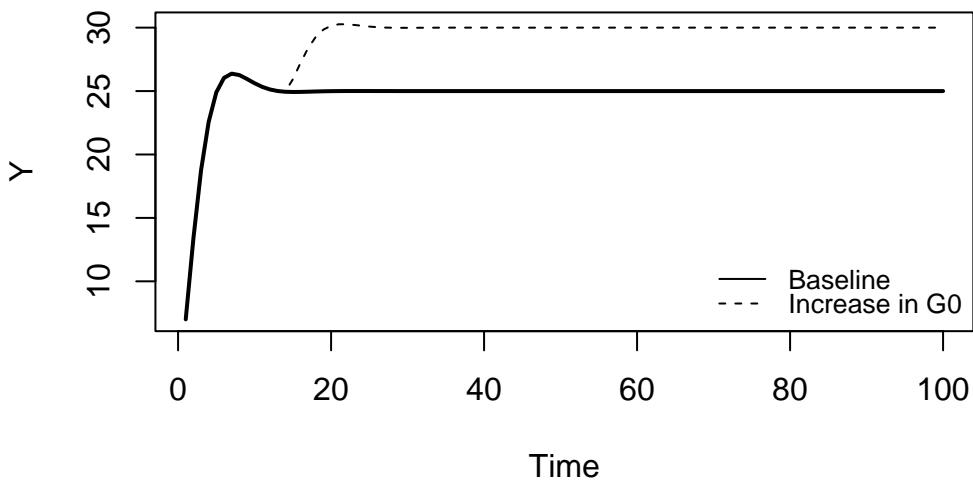
Figure 2: Output



As you can see, there are many settings you can fiddle around with to adjust the appearance of the graph to your liking (run ‘?plot’ to find information about the different options.) If we want to plot the dynamics for the two different parameterisations, we can do the following:

```
# Time series chart of output dynamics for different scenarios in Samuelson (1939) model
plot(Y[1, 1:100], type="l", col=1, lwd=2, lty=1, xlab="Time", ylab="Y", ylim=range(min(Y[1,
title(main="Figure 3: Output under different scenarios", cex=0.8)
lines(Y[2, 1:100], lty=2)
legend("bottomright", legend=c("Baseline", "Increase in G0"),
      lty=1:2, cex=0.8, bty = "n", y.intersp=0.8)
```

Figure 3: Output under different scenarios

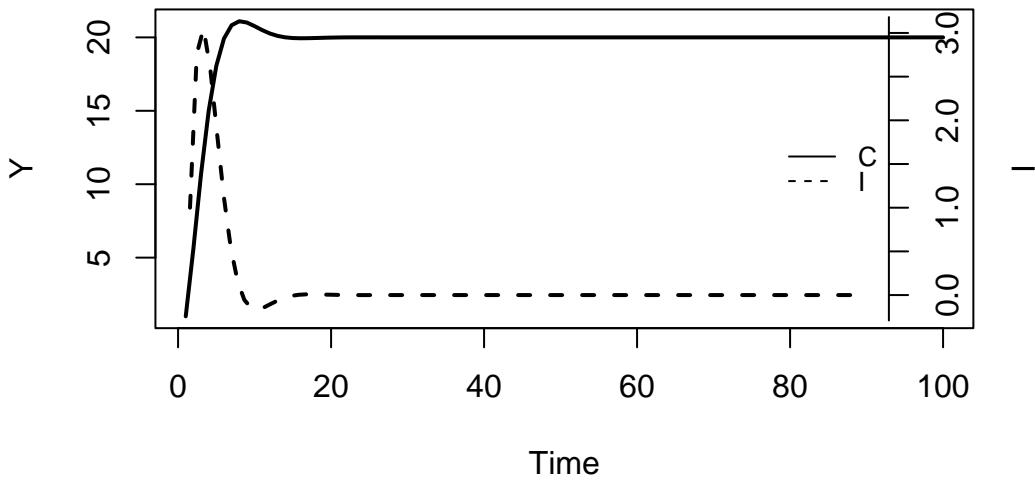


Note that we have adjusted the range of the y-axis to make sure the graph captures the minimum and maximum values from both parameterisations.

Finally, consider a plot for C_t and I_t with two separate axes:

```
# Time series chart of Samuelson (1939) model with separate axes for consumption and investment
plot(C[1, 1:100], type="l", col=1, lwd=2, lty=1, xlab="Time", ylab="Y")
title(main="Figure 3: Consumption and Investment", cex=0.8)
par(mar = c(5, 4, 4, 4) + 0.3)
par(new = TRUE)
plot(I[1, 1:100], type="l", col=1, lwd=2, lty=2, font.main=1, cex.main=1, ylab = '', axes=F)
  xlab = '', ylim = range(I[1, 1:100]), cex=0.8)
axis(side = 4, at=pretty(range(I[1, 1:100])))
mtext("I", side = 4, line = 3)
legend("right", legend=c("C", "I"),
  lty=1:2, cex=0.8, bty = "n", y.intersp=0.8)
```

Figure 3: Consumption and Investment



 Python code

```

##### Plots

# Load matplotlib library
import matplotlib.pyplot as plt

# Bar chart of different equilibrium solutions of Samuelson (1939) model
scenario_labels = ["Baseline", "Increase in G0"]
plt.bar(scenario_labels, Y[:, Q - 1])
plt.xlabel("Scenario")
plt.ylabel("Y")
plt.title("Figure 1: Output")
plt.show()

# Time series chart of output dynamics in Samuelson (1939) model
plt.plot(range(1, Q), Y[0, 0:Q - 1], color='black', linewidth=2, linestyle='--')
plt.xlabel("Time")
plt.ylabel("Y")
plt.title("Figure 2: Output", fontsize=10)
plt.show()

# Time series chart of output dynamics for different scenarios in Samuelson
#(1939) model
plt.plot(range(1, Q), Y[0, 0:Q - 1], color='black', linewidth=1, linestyle='--')
plt.plot(range(1, Q), Y[1, 0:Q - 1], color='black', linewidth=1, linestyle='---')
plt.xlabel("Time")
plt.ylabel("Y")
plt.title("Figure 3: Output under different scenarios", fontsize=10)
plt.legend(["Baseline", "Increase in G0"], loc='lower right')
plt.show()

# Time series chart of Samuelson (1939) model with separate axes for consumption
# and investment
fig, ax1 = plt.subplots()
ax1.plot(range(1, Q), C[0, 0:Q - 1], color='black', linewidth=2, linestyle='--',
          label='C')
ax1.set_xlabel("Time")
ax1.set_ylabel("C", color='black')
ax1.tick_params(axis='y', labelcolor='black')
ax2 = ax1.twinx()
ax2.plot(range(1, Q), I[0, 0:Q - 1], color='black', linewidth=2, linestyle='---',
          label='I')
ax2.set_ylabel("I", color='black')
ax2.tick_params(axis='y', labelcolor='black')
plt.title("Figure 4: Consumption and Investment", fontsize=10)
lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
39
ax2.legend(lines + lines2, labels + labels2, loc='right')
plt.show()

```

2.4 How to create a directed graph of a model

Another perspective on a model's properties is provided by its directed graph. A directed graph consists of a set of nodes that represent the variables of the model. Nodes are connected by directed edges. An edge directed from a node x_1 to node x_2 indicates a causal impact of x_1 on x_2 .

The directed graph can be derived from the model's Jacobian matrix.² Let x be the vector containing the model's endogenous variables and $f(x)$ the system of equations making up the model. The Jacobian matrix is then given by $J = \frac{\partial f()}{\partial x}$. As we often also want to display exogenous variables in the directed graph, it can be useful to expand the Jacobian matrix by adding rows and columns for those exogenous variables.

Next, construct an ‘auxiliary’ Jacobian matrix M in which all the non-zero elements of the Jacobian are replaced by ones, whereas zero elements remain unchanged, i.e.

$$M_{ij} = \begin{cases} 1 & \text{if } J_{ij} \neq 0, \\ 0 & \text{otherwise.} \end{cases}$$

Finally, taking the transpose of this ‘auxiliary’ Jacobian matrix yields the *adjacency matrix* $A = M^T$, which is a binary matrix whose elements (A_{ji}) indicate whether there is a directed edge from a node x_j to node x_i . From the adjacency matrix, the directed graph is constructed.

The code below shows this for the example of the Samuelson (1939) model.³

```
## Create directed graph
# Construct auxiliary Jacobian matrix for 4 variables:
# endogenous: (1) Y, (2) C, (3) I
# exogenous: (4) G0
# where non-zero elements in regular Jacobian are set to 1 and zero elements are unchanged

#1 2 3 4
M_mat=matrix(c(0,1,1,1, #1
              1,0,0,0, #2
              0,1,0,0, #3
              0,0,0,0), #4
              4, 4, byrow=TRUE)
```

²See Fennell et al. (2015) for a neat exposition.

³To create the directed graph, we rely on external libraries which you may have to install first. In *R*, this can be accomplished with `install.packages("igraph")` and in Python with `pip install networkx`. Once that library is installed, you only need to activate it in each session before you use it. In *R*, you can do this by executing `library(igraph)` and in *Python* through `import networkx`.

```

# Create adjacency matrix from transpose of auxiliary Jacobian and add column names
A_mat=t(M_mat)

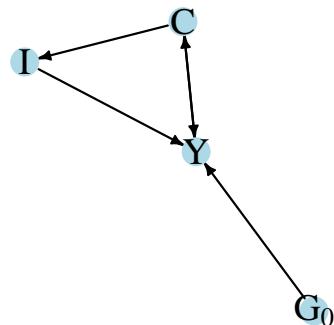
# Create directed graph from adjacency matrix
library(igraph)
dg=graph_from_adjacency_matrix(A_mat, mode="directed", weighted= NULL)

# Define node labels
V(dg)$name=c("Y", "C", "I", expression(G[0]))

# Plot directed graph matrix
plot(dg, main="Figure 4: Directed graph of Samuelson model", vertex.size=20, vertex.color=
      vertex.label.color="black", edge.arrow.size=0.3, edge.width=1.1, edge.size=1.2,
      edge.arrow.width=1.2, edge.color="black", vertex.label.cex=1.2,
      vertex.frame.color="NA", margin=-0.08)

```

Figure 4: Directed graph of Samuelson model



 Python code

```
#Load relevant libraries
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

# Construct auxiliary Jacobian matrix for 4 variables
# endogenous: (1) Y, (2) C, (3) I
# exogenous: (4) G0
# where non-zero elements in regular Jacobian are set to 1 and zero elements are
# unchanged
M_mat = np.array([[0, 1, 1, 1],
                  [1, 0, 0, 0],
                  [0, 1, 0, 0],
                  [0, 0, 0, 0]])

# Create adjacency matrix from transpose of auxiliary Jacobian and add column names
A_mat = M_mat.transpose()

# Create the graph from the adjacency matrix
G = nx.DiGraph(A_mat)

# Define node labels
nodelabs = {0: "Y", 1: "C", 2: "I", 3: "$G_0$"}

# Plot the directed graph
pos = nx.spring_layout(G, seed=42)
nx.draw(G, pos, with_labels=True, labels=nodelabs, node_size=500, node_color='lightblue')
edge_labels = {(u, v): '' for u, v in G.edges}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='black')
plt.title("Figure 5: Directed graph of Samuelson model", fontsize=12)
plt.axis('off')
plt.show()
```

Broadly speaking, three types of nodes are possible:

1. nodes from which arrows only point away represent exogenous variables (G_0 in Figure 4)
2. nodes to which arrows point but from which arrows also point away represent endogenous variables that also have a causal impact on other variables (Y , C , and I in Figure 4)

3. nodes to arrows only point towards represent endogenous variables that are residuals (not present in Figure 4)

It can thus be seen that the key exogenous variable in the Samuelson (1939) model is government spending G_0 , which determines aggregate income, which in turn feeds into consumption. Consumption then feeds into investment, which feeds back into income yielding the multiplier-accelerator effect.

2.5 Appendix: How to simulate dynamic model in continuous time

Sometimes, dynamic economic models are written in continuous time where the time increment is assumed to be infinitesimally small. Consider, for example, a continuous-time version of the Keynesian goods market model:

$$\dot{Y} = k(C + I_0 - Y)$$

$$C = c_0 + c_1 Y,$$

where $\dot{Y} = \frac{dY}{dt}$. To simulate this model, we have to approximate the differential equation for \dot{Y} by a difference equation for Y_t and let the time increment Δt become very small. More specifically, we write:

$$Y(t + \Delta t) = Y_t + \dot{Y}\Delta t = Y_t + [k(C + I_0 - Y)]\Delta t.$$

This approach is also called the *Euler forward method*. The code below shows to implement that method. We use $\Delta t = 0.01$. Note that we need to raise the time horizon Q and set the adjustment speed k relatively high to make sure that the system has effectively converged to the equilibrium by Q .

```
### Simulate continuous time version of Keynesian goods market model

#Clear the environment
rm(list=ls(all=TRUE))

# Set number of periods for which you want to simulate
Q=800

# Set number of parameterisations that will be considered
S=1
```

```

# Set fixed parameter values
c0=3
c1=0.8
k=8
delta=0.01

# Construct matrices in which values for different periods will be stored; initialise at 1
Y=matrix(data=1, nrow=S, ncol=Q)
C=matrix(data=1, nrow=S, ncol=Q)

#Construct matrices for exogenous variable
I0=matrix(data=5, nrow=S, ncol=Q)

#Solve this system recursively based on the initialisation
for (t in 2:Q){
  for (iterations in 1:1000){
    Y[1,t] = Y[1,t-1] + delta*(k*(C[1,t-1] + I0[1,t-1] - Y[1,t-1]))
    C[1,t] = c0 + c1*Y[1,t]
  } # close within-period loop
} # close time loop

# Verify solutions for Y
(c0+I0[1,Q])/(1-c1)

```

[1] 40

Y[1,Q]

[1] 39.9999

Python code

```
### Simulate continuous time version of Keynesian goods market model

# Load NumPy
import numpy as np

# Set number of periods for which you want to simulate
Q = 800

# Set number of parameterizations that will be considered
S = 1

# Set fixed parameter values
c0 = 3
c1 = 0.8
k = 8
delta = 0.01

# Initialize matrices to store values for different periods
Y = np.ones((S, Q))
C = np.ones((S, Q))

# Initialize the matrix for the exogenous variable
I0 = np.full((S, Q), 5)

# Solve this system recursively based on the initialization
for t in range(1, Q):
    for iterations in range(1000):
        Y[0, t] = Y[0, t - 1] + delta * (k * (C[0, t - 1] + I0[0, t - 1]) - Y[0, t - 1])
        C[0, t] = c0 + c1 * Y[0, t]

# Verify the solution for Y at time Q
(c0 + I0[0, Q - 1]) / (1 - c1)

Y[0, Q - 1]
```

2.6 References

Part I

Static Models

3 A Neoclassical Macro Model

3.1 Overview

This model captures some key features of neoclassical macroeconomics. The model is based on the optimising behaviour of firms and households that interact in perfect markets. Households choose between labour and leisure as well as current and future consumption. They form (rational) expectations about their future income that impact their consumption decisions today. Firms are owned by households and maximise profits using a constant returns to scale technology (with diminishing marginal returns to factors of production). Markets are competitive and clear instantaneously through flexible prices. The most important market is the labour market, in which the equilibrium between firms' demand and households' supply of labour is established by a flexible real wage. The level of employment established on the labour market then determines aggregate supply via the production function. Aggregate demand always accommodates to aggregate supply via a flexible real interest rate that establishes an equilibrium between investment and saving (often interpreted as the market for loanable funds). The money supply is exogenous in this model and only impacts the price level but not the real economy – the so-called *Classical Dichotomy* (or neutrality of money). Government expenditures do influence the real economy but they crowd out private expenditures (through a mechanism called *Ricardian Equivalence*).

In this short- to medium-run version of the model, prices are flexible but the capital stock is fixed. The focus is thus on goods market equilibrium rather than economic growth. As all endogenous variables adjust instantaneously, the model is thus static. However, expectations about future income and government spending will impact current consumption of households that intertemporally maximise utility. The model is adapted from Garín, Lester, and Sims (2021).

3.2 The Model

$$Y = AK^a N^{1-a}, \quad a \in (0, 1) \quad (3.1)$$

$$w = (1 - a)AK^a N^{-a} \quad (3.2)$$

$$N = 1 - \frac{b_1}{w}, \quad b_1 > 0 \quad (3.3)$$

$$C = c_1 \left[Y - G + \frac{Y^f - G^f}{1+r} - c_2 \ln \left(\frac{b_1}{w} \right) \right] \quad (3.4)$$

$$I = \left(\frac{aAN^{1-a}}{r} \right)^{\frac{1}{1-a}} \quad (3.5)$$

$$G = G_0 \quad (3.6)$$

$$Y = C(r) + I(r) + G \quad (3.7)$$

$$r_n = r + \pi^f \quad (3.8)$$

$$M_s = M_0 \quad (3.9)$$

$$M_d = \frac{b_3(1+r_n)PC}{r_n} \quad (3.10)$$

$$M = M_d(P) = M_s \quad (3.11)$$

where Y , K , N , w , C , G , r , I , r_n , π , M_s , M_d , and P_t are real output, the capital stock, employment, the real wage, consumption, government expenditures, the real interest rate, investment, the nominal interest rate, inflation, the money supply, money demand, and the price level, respectively. The f -superscript denotes (expected) future values. For simplicity, expected future variables will be treated as exogenous.

By Equation 3.1, output is determined by a Cobb-Douglas production function with constant returns to scale and diminishing marginal returns to each factor (capital and labour). Equation 3.2 is the labour demand of firms solved for the real wage. Profit maximising firms will hire workers until the real wage is equal to the marginal product of labour ($\frac{\partial Y}{\partial N} = (1-a)AK^a N^{-a}$).¹ Equation 3.3 specifies households' labour supply, which is positively related to the real wage. By Equation 3.4, consumption is positively related to the

¹See the analytical discussion below for a derivation of equations Equation 3.2 -Equation 3.5 and Equation 3.10 from optimising microfoundations.

real wage, current and (expected) future income,² and negatively related to the real interest rate. The composite parameters $c_1 = \left(\frac{1}{1+b_2+b_3}\right)$ and $c_2 = b_1(b_2 + b_3)$ stem from the weights in the household's utility function (see analytical discussion below). The consumption function implies that for a higher real interest rate, households increase their saving (the supply of loanable funds). Furthermore, consumption is negatively related to current and future government expenditures. This is an implication of consumption smoothing: as households know that governments will have to repay its debts in the future through higher taxes, an increase in government expenditures today is perceived as an increase in taxes (regardless of whether the government finances its current expenditures through taxes or debt, a result that is also called 'Ricardian Equivalence').³ Equation 3.5 specifies investment as a negative function of the real interest rate and a positive function of productivity. By Equation 3.6, government expenditures are exogenous. Equation 3.7 is the goods market equilibrium condition, which pins down the equilibrium real interest rate through the market for loanable funds. Equation 3.8 specifies the nominal interest rate using the well-known Fisher equation. Equation 3.9 says that the money supply is exogenous. By Equation 3.10, households' money demand is negatively related to the nominal interest rate and positively related to consumption (capturing the transaction demand for money). Finally, Equation 3.11 is the equilibrium condition for the money market, which pins down the price level.

3.3 Simulation

3.3.1 Parameterisation

Table 1 reports the parameterisation used in the simulation. Besides a baseline (labelled as scenario 1), five further scenarios will be considered. Scenarios 2 and 3 are a monetary expansion (increase in the exogenous money supply M_0) and fiscal expansion (increase in G_0), respectively. Scenario 4 is an improvement in total factor productivity (A) and scenario 5 a fall in expected future income (Y^f). Finally, scenario 6 is a shift in household preferences towards more leisure (b_1).

Table 1: Parameterisation

Scenario A	a	b_1	b_2	b_3	G_0	Y^f	M_0	K	π_f	
1: base- line	2	0.3	0.4	0.9	0.6	1	1	5	5	0.02

²Households are assumed to form rational expectations. In a deterministic setting, this implies perfect foresight so that expected and actual future income coincide.

³See the analytical discussion below for more details on Ricardian Equivalence.

Scenario A	a	b_1	b_2	b_3	G_0	Y^f	M_0	K	π_f	
2: monetary expansion (M_0)	2	0.3	0.4	0.9	0.6	1	1	6	5	0.02
3: fiscal expansion (G_0)	2	0.3	0.4	0.9	0.6	2	1	6	5	0.02
4: productivity boost (A)	2.5	0.3	0.4	0.9	0.6	1	1	5	5	0.02
5: lower expected future income (Y^f)	2	0.3	0.4	0.9	0.6	1	0.2	5	5	0.02
6: increased preference for leisure (b_1)	2	0.3	0.8	0.9	0.6	1	1	5	5	0.02

3.3.2 Simulation code

```
# Clear the environment
rm(list=ls(all=TRUE))

# Set number of scenarios (including baseline)
S=6

#Create vector in which equilibrium solutions from different parameterisations will be stored
Y_star=vector(length=S) # Income/output
w_star=vector(length=S) # Real wage
C_star=vector(length=S) # Consumption
I_star=vector(length=S) # Investment
r_star=vector(length=S) # Real interest rate
rn_star=vector(length=S) # Nominal interest rate
N_star=vector(length=S) # Employment
P_star=vector(length=S) # Price level

# Create and parameterise exogenous variables/parameters that will be shifted
M0=vector(length=S) # money supply
G0=vector(length=S) # government expenditures
A=vector(length=S) # productivity
Yf=vector(length=S) # expected future income
b1=vector(length=S) # household preference for leisure
M0 []=5
G0 []=1
A []=2
Yf []=1
b1 []=0.4

# Set parameter values for different scenarios
M0[2]=6 # scenario 2: monetary expansion
G0[3]=2 # scenario 3: fiscal expansion
A[4]=2.5 # scenario 4: productivity boost
Yf[5]=0.2 # scenario 5: lower expected future income
b1[6]=0.8 # scenario 6: increased preference for leisure

#Set constant parameter values
a=0.3 # Capital elasticity of output
b2=0.9 # discount rate
b3=0.6 # household preference for money
```

```

K=5      # Exogenous capital stock
pe=0.02 # Expected rate of inflation
Gf=1    # Future government spending

# Initialise endogenous variables at arbitrary positive value
w = C = I = Y = r = N = P = 1

#Solve this system numerically through 1000 iterations based on the initialisation
for (i in 1:S){

  for (iterations in 1:1000){

    #Model equations

    #(1) Cobb-Douglas production function
    Y = A[i]*(K^a)*N^(1-a)

    #(2) Labour demand
    w = A[i]*(1-a)*(K^a)*N^(-a)

    #(3) Labour supply
    N = 1 - (b1[i])/w

    #(4) Consumption demand
    C = (1/(1+b2+b3))*(Y - GO[i] + (Yf[i]-Gf)/(1+r) - b1[i]*(b2+b3)*log(b1[i]/w))

    #(5) Investment demand, solved for r
    r=(I^(a-1))*a*A[i]*N^(1-a)

    #(6) Goods market equilibrium condition, solved for I
    I = Y - C - GO[i]

    #(7) Nominal interest rate
    rn = r + pe

    #(8) Price level
    P = (MO[i]*rn)/((1+rn)*b3*C)

  }

  #Save results for different parameterisations in vector
}

```

```
Y_star[i]=Y  
w_star[i]=w  
C_star[i]=C  
I_star[i]=I  
r_star[i]=r  
N_star[i]=N  
P_star[i]=P  
rn_star[i]=rn  
}
```

 Python code

```

import numpy as np

# Set the number of scenarios (including baseline)
S = 6

# Create arrays to store equilibrium solutions from different parameterizations
Y_star = np.empty(S) # Income/output
w_star = np.empty(S) # Real wage
C_star = np.empty(S) # Consumption
I_star = np.empty(S) # Investment
r_star = np.empty(S) # Real interest rate
rn_star = np.empty(S) # Nominal interest rate
N_star = np.empty(S) # Employment
P_star = np.empty(S) # Price level

# Create and parameterize exogenous variables/parameters that will be shifted
M0 = np.zeros(S) # Money supply
G0 = np.zeros(S) # Government expenditures
A = np.zeros(S) # Productivity
Yf = np.zeros(S) # Expected future income
b1 = np.zeros(S) # Household preference for leisure

# Baseline parameterisation
M0[:] = 5
G0[:] = 1
A[:] = 2
Yf[:] = 1
b1[:] = 0.4

# Set parameter values for different scenarios
M0[1] = 6 # Scenario 2: monetary expansion
G0[2] = 2 # Scenario 3: fiscal expansion
A[3] = 2.5 # Scenario 4: productivity boost
Yf[4] = 0.2 # Scenario 5: lower expected future income
b1[5] = 0.8 # Scenario 6: increased preference for leisure

# Set constant parameter values
a = 0.3 # Capital elasticity of output
b2 = 0.9 # Discount rate
b3 = 0.6 # Household preference for money
K = 5 # Exogenous capital stock
pe = 0.02 # Expected rate of inflation
Gf = 1 # Future government spending
55

# Initialize endogenous variables at arbitrary positive values
w = C = I = Y = r = N = P = 1

# Solve this system numerically through 1000 iterations based on the initialization
for i in range(S):
    for iterations in range(1000):
        # Model equations

```

3.3.3 Plots

Figures Figure 3.1 - Figure 3.3 depicts the model economy's real activity under different scenarios. Monetary expansions (scenario 2) have no effects on output and employment, capturing the idea of money neutrality (Classical Dichotomy). However, the monetary expansion does raise the price level (see Figure 3.2), capturing the key idea of the Quantity Theory of Money that inflation is driven by a growing money supply.

```
barplot(Y_star, ylab="Y", names.arg=c("1: Baseline", "2: Increase in M0", "3: Increase in
```

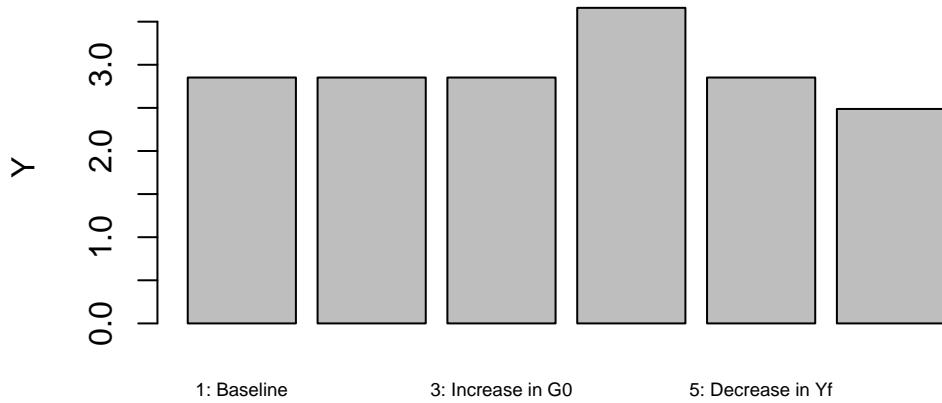


Figure 3.1: Output

```
barplot(P_star, ylab="P", names.arg=c("1: Baseline", "2: Increase in M0", "3: Increase in
```

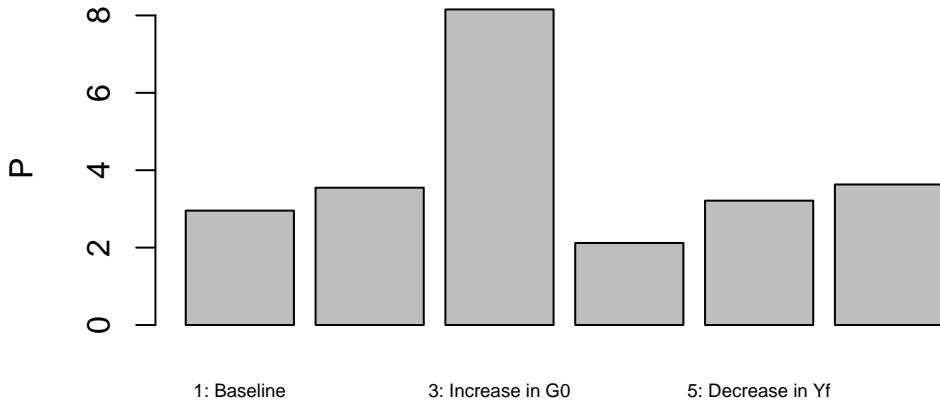


Figure 3.2: Price level

An increase in government spending (by one unit) in scenario 3 does *not* raise employment and real output as these are fully determined by the supply side rather than aggregate demand.⁴ Looking at Figures Figure 3.4 and Figure 3.5, it can be seen that government spending crowds out private spending (consumption and investment). The decrease in consumption is a result of consumption smoothing via the Euler equation: the household anticipates higher taxes in the future and reduces current consumption somewhat to smooth out the impact. The increase in government expenditures also raises demand for loanable funds, which pushes up the real interest rate (see Figure 3.6) and reduces investment (Figure 3.5) and consumption. There is also an inflationary effect linked to the higher real interest rate, which raises the nominal interest rate and reduces the demand for money.

```
barplot(N_star, ylab="N", names.arg=c("1: Baseline", "2: Increase in M0", "3: Increase in G0"))
```

⁴This result is partly driven by the use of a non-separable utility function, see analytical discussion below. With a separable utility function, the increase in government spending would increase labour supply and thereby have effects on employment and output.

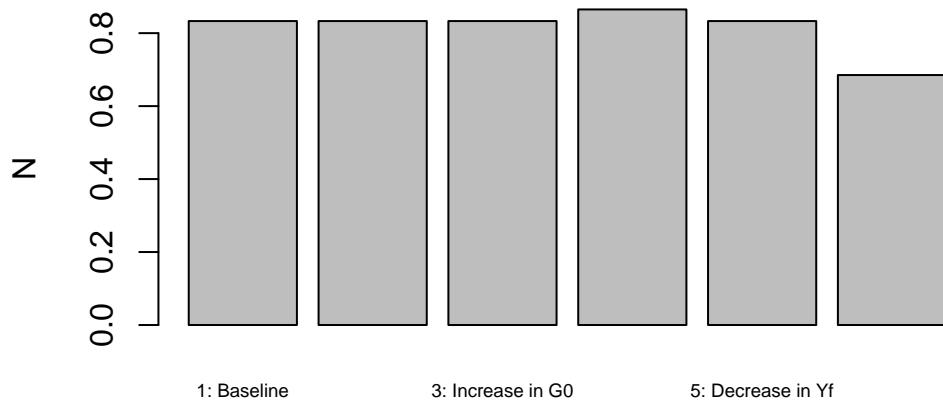


Figure 3.3: Employment

Improvements to the supply-side are captured by an increase in productivity (scenario 4), which raises real output, employment, and aggregate demand. In contrast to expansionary government policy, the price level falls.

```
barplot(C_star, ylab="C", names.arg=c("1: Baseline", "2: Increase in M0", "3: Increase in G0"))
```

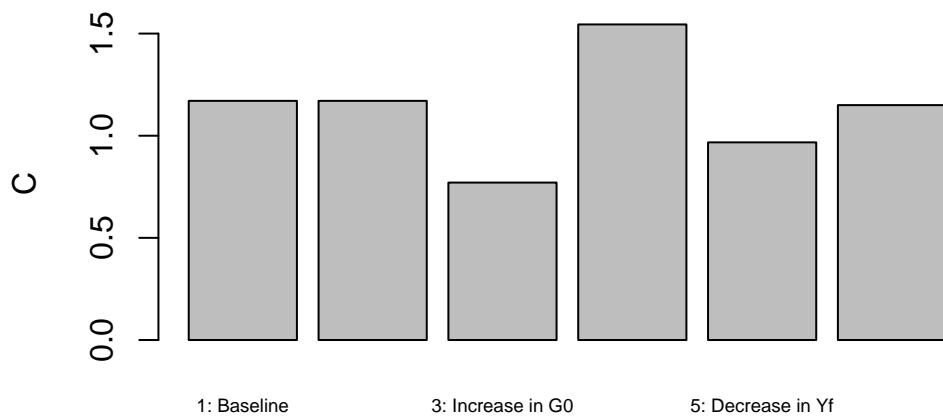


Figure 3.4: Consumption

```
barplot(I_star, ylab="I", names.arg=c("1: Baseline", "2: Increase in M0", "3: Increase in G0", "4: Decrease in Yf", "5: Decrease in Yf"))
```

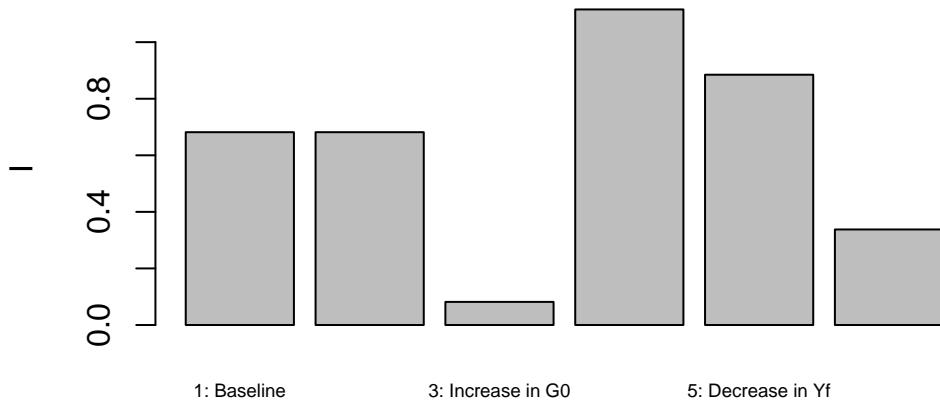


Figure 3.5: Investment

A decrease in expected future income (scenario 5) has a small expansionary effect on aggregate output. This is because households will reduce some of their current consumption (see Figure 3.4) and supply more labour to smooth their consumption over time. The increase in saving reduces the real interest rate (see Figure 3.6) and is compensated by an increase in investment (see Figure 3.5).

```
barplot(r_star, ylab="r", names.arg=c("1: Baseline", "2: Increase in M0", "3: Increase in G0", "4: Decrease in Yf", "5: Decrease in Yf"))
```

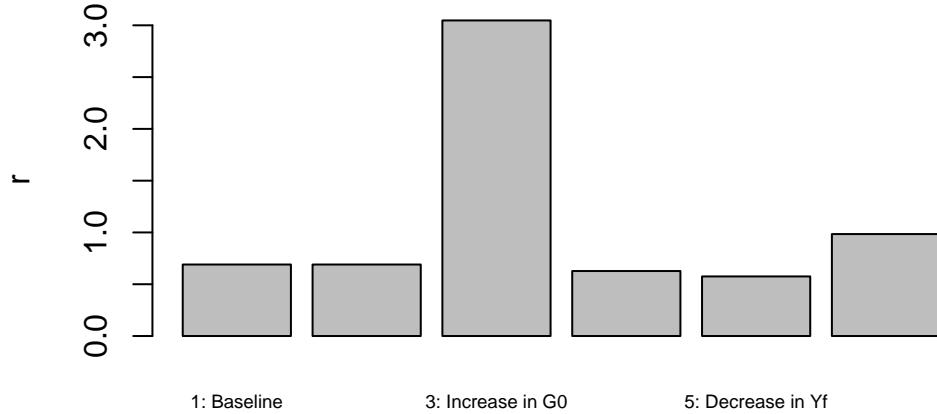


Figure 3.6: Interest rate

Finally, an increase in the preference for leisure (scenario 6) reduces labour supply and thereby output.

Python code

```
# Plot results (here for output only)
import matplotlib.pyplot as plt

scenario_names = ["1: Baseline", "2: Increase in  $M_0$ ", "3: Increase in  $G_0$ ",
                  "4: Increase in  $A$ ", "5: Decrease in  $Y_f$ ", "6: Increase in  $b_1$ "]

# Output
plt.bar(scenario_names , Y_star)
plt.ylabel('Y')
plt.xticks(rotation=45, ha="right") # Rotate x-axis labels for better readability
plt.tight_layout() # Ensure the labels fit within the plot area
plt.show()
```

3.4 Directed graph

Another perspective on the model's properties is provided by its directed graph. A directed graph consists of a set of nodes that represent the variables of the model. Nodes are connected by directed edges. An edge directed from a node x_1 to node x_2 indicates a causal impact of x_1 on x_2 .

```
## Create directed graph
# Construct auxiliary Jacobian matrix for 13 variables:
# Y w N C I r P rn M0 G0 A Yf Md
# where non-zero elements in regular Jacobian are set to 1 and zero elements are unchanged

M_mat=matrix(c(0,0,1,0,0,0,0,0,0,0,1,0,0,
              0,0,1,0,0,0,0,0,0,0,1,0,0,
              0,1,0,0,0,0,0,0,0,0,0,0,0,
              1,1,0,0,0,1,0,0,0,1,0,1,0,
              0,0,1,0,0,1,0,0,0,0,0,0,0,
              1,0,0,1,1,0,0,0,0,1,0,0,0,
              0,0,0,0,0,0,0,0,1,0,0,0,1,
              0,0,0,0,0,1,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,1,0,0,1,1,0,0,0,0,0), 13, 13, byrow=TRUE)

# Create adjacency matrix from transpose of auxiliary Jacobian
A_mat=t(M_mat)

# Create directed graph from adjacency matrix
library(igraph)
dg= graph_from_adjacency_matrix(A_mat, mode="directed", weighted= NULL)

# Define node labels
V(dg)$name=c("Y","w","N","C","I","r","P", expression(r[n]), expression(M[0]),expression(G[

# Plot directed graph
plot(dg, main="", vertex.size=20, vertex.color="lightblue",
      vertex.label.color="black", edge.arrow.size=0.3, edge.width=1.1, edge.size=1.2,
      edge.arrow.width=1.2, edge.color="black", vertex.label.cex=1.2,
      vertex.frame.color="NA", margin=-0.08)
```

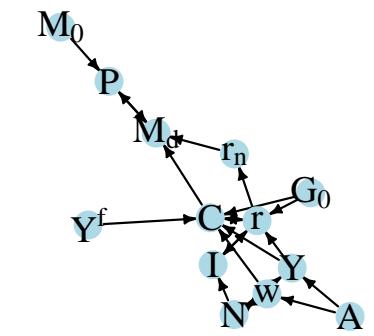


Figure 3.7: Directed graph

 Python code

```

# Load relevant libraries
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

# Construct the auxiliary Jacobian matrix
M_mat = np.array([
    [0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
    [0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
    [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0],
    [0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1],
    [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0]
])

# Create adjacency matrix from transpose of auxiliary Jacobian and add column names
A_mat = M_mat.transpose()

# Create the graph from the adjacency matrix
G = nx.DiGraph(A_mat)

# Define node labels
nodelabs = {
    0: "Y",
    1: "w",
    2: "N",
    3: "C",
    4: "I",
    5: "r",
    6: "P",
    7: r"$r_n$",
    8: r"$M_0$",
    9: r"$G_0$",
    10: "A",
    11: r"$Y^f$",
    12: r"$M_d$"
}

# Plot the directed graph
pos = nx.spring_layout(G, seed=42)
nx.draw(G, pos, with_labels=True, labels=nodelabs, node_size=300, node_color='lightblue',
        font_size=10)
edge_labels = {(u, v): '' for u, v in G.edges}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='black')
plt.axis('off')

```

In Figure 3.7 , it can be seen that productivity (A), future income (Y^f), the money supply (M_0), and government spending (G_0) are the key exogenous variables of the model. All other variables are endogenous, and many of them form a closed loop (or cycle) within the system. The left part of the graph captures the supply side: the labour market simultaneously pins down the real wage and employment, which together with the exogenously level of productivity determine output. The part in the middle embodies the demand side: aggregate demand (consumption, investment, and government spending) together with a given level of output determine the real interest rate. The right part captures the nominal side of the model given by the money market, which determines the price level. Notably, while the real side of the model feeds into the money market via the nominal interest rate (r_n), there is not causal effect from the money market to the real side reflecting the Classical Dichotomy.

3.5 Analytical discussion: derivation of behavioural functions

3.5.1 The firm's problem: profit maximisation

The firm's profit equation is given by:

$$\Pi = Y - wN - rk \quad (3.12)$$

$$= AK^a N^{1-a} - wN - rk. \quad (3.13)$$

The firm's optimisation problem is to maximise profits using employment and capital as choice variables:⁵

$$\max_{N, K} \Pi = AK^a N^{1-a} - wN - rk. \quad (3.14)$$

The first-order conditions are given by:

$$(1 - a)AK^a N^{-a} - w = 0 \quad (3.15)$$

$$aAK^{a-1} N^{1-a} - r = 0. \quad (3.16)$$

From Equation 3.15, Equation 3.2 for labour demand can be derived. While the capital stock is pre-determined in every period, the firm can adjust the capital stock through investment.

⁵See Garín, Lester, and Sims (2021, chap. 12) for a more elaborate version where the firm maximises its lifetime value. The resulting investment function is very similar.

We thus use the first-order condition for capital Equation 3.16 to derive Equation 3.5 for investment demand.

3.5.2 The government's budget constraint

The government's current and future budget constraints are given by:

$$G = T + B \quad (3.17)$$

$$G^f = T^f + (1 + r)B \quad (3.18)$$

where T is tax revenues and B is government debt.

A crucial assumption here is that the government must repay its debts in the future (it cannot permanently roll over its debts). The underlying assumption is that the government will eventually 'die' and will do so without savings nor debts.

Combining the two budget constraints to an intertemporal budget constraint yields:

$$G + \frac{G^f}{1+r} = T + \frac{T^f}{1+r} \quad (3.19)$$

Thus, while the government's budget need not be balanced in every period, it will be balanced over time (in a present value sense).⁶

3.5.3 The household's problem: intertemporal utility maximisation and Ricardian Equivalence

The household derives utility from current consumption, leisure ($1 - N$), future consumption C^f , and from holding real money balances ($\frac{M}{P}$). We use the following log utility function that is non-separable in consumption and leisure:⁷

$$U = \ln[C + b_1 \ln(1 - N)] + b_2 \ln(C^f) + b_3 \ln\left(\frac{M}{P}\right), \quad b_2 \in (0, 1); b_1, b_3 > 0 \quad (3.20)$$

⁶See Garín, Lester, and Sims (2021, chap. 13) for a more detailed discussion of the government's budget constraints.

⁷See Garín, Lester, and Sims (2021), pp.280-289), on the differences between a separable and a non-separable preference specification. Note also that for simplicity, the utility function omits the utility from future leisure.

The household's current budget constraint is given by:⁸

$$C = Y - T - S - \frac{M}{P} \quad (3.21)$$

The income that is not consumed can either be saved (S) at nominal interest rate (r_n) or held as money on which no return is earned.

The future budget constraint (assuming that households do not save in the future) is given by:

$$C^f = Y^f - T^f + (1 + r_n) \frac{PS}{P^f} + \frac{M}{P^f}. \quad (3.22)$$

Using the Fisher equation $(1 + r) = (1 + r_n) \frac{P}{P^f}$, the future budget constraint can be rewritten as:

$$C^f = Y^f - T^f + (1 + r)S + \frac{(1 + r)M}{(1 + r_n)P} \quad (3.23)$$

Finally, the two budget constraints can be combined to yield an intertemporal budget constraint:

$$C^f = (Y - C - T)(1 + r) + Y^f - T^f - (1 + r) \frac{r_n M}{(1 + r_n)P}. \quad (3.24)$$

Substituting the government's intertemporal budget constraint, the household's intertemporal budget constraint can also be written as:

$$C^f = (Y - C - G)(1 + r) + Y^f - G^f - (1 + r) \frac{r_n M}{(1 + r_n)P} \quad (3.25)$$

The fact that current and future tax payments are now replaced by current and future government spending means that it does not matter for the real economy how government spending is financed: the private sector will react to expenditures that are financed out of debt in the same way it reacts to expenditures financed out of taxation. This result is also known as *Ricardian Equivalence*. Put differently, Ricardian Equivalence means the household behaves as if the government balances its budget in every period.

With these ingredients, the household's optimisation problem can be written as:

⁸See Garín, Lester, and Sims (2021, chap. 14) for a more detailed discussion of the household's budget constraints.

$$\begin{aligned} \max_{C,N,M} \quad & U = \ln(C + b_1 \ln(1 - N)) + b_2 \ln(C^f) + b_3 \ln\left(\frac{M}{P}\right), \\ \text{s.t.} \quad & C^f = (Y - C - G)(1 + r) + Y^f - G^f - (1 + r) \frac{r_n M}{(1+r_n)P}. \end{aligned}$$

Substituting the constraint Equation 3.25 for C^f in the utility function, and making use of the identity $Y = \Pi + rK + Nw$, allows to obtain the following first-order conditions:

$$C = \frac{C^f}{b_2(1+r)} - b_1 \ln(1 - N) \quad (3.26)$$

$$\left(\frac{b_1}{1-N}\right) \left(\frac{1}{C + b_1 \ln(1 - N)}\right) = \frac{b_2(1+r)w}{C^f} \quad (3.27)$$

$$M = \frac{b_3 P C^f (1 + r_n)}{b_2(1+r)r_n}. \quad (3.28)$$

Substituting Equation 3.26, which is often also called the *Euler equation*, into Equation 3.27 and Equation 3.28 yields Equation 3.3 for labour supply and Equation 3.10 for money demand, respectively. Finally, using Equation 3.25 and Equation 3.28 in the Euler Equation 3.26 yields the consumption function Equation 3.4.

References

4 An IS-LM Model

4.1 Overview

The IS-LM model was developed by John R. Hicks (1937) to formalise some key ideas of John Maynard Keynes' 1936 book [The General Theory of Employment, Interest and Money](#). The model contains two equilibrium relationships: a goods market equilibrium between investment and saving (IS) and a money market equilibrium between money demand and money supply (LM). In the goods market, aggregate supply adjusts to the level of aggregate demand given by the expenditure decisions of households, firms, and the government. Households form their consumption demand based on a constant marginal propensity to consume out of income. Firms take investment decisions based on the rate of interest. Money demand is determined by aggregate income (transactions demand) and the interest rate on bonds (speculative demand). The money supply is assumed to be exogenous and under the control of the central bank. The two markets pin down equilibrium output and the interest rate. The goods market equilibrium may well coincide with involuntary unemployment. Adverse shocks to autonomous investment ('animal spirits') or autonomous money demand ('liquidity preference') reduce output and raise unemployment. The government can use monetary policy, fiscal spending, and income taxes to stimulate economic activity and achieve full employment.

In this short-run model, prices and the capital stock are fixed. The focus is thus on goods market equilibrium rather than economic growth. As all endogenous variables adjust instantaneously, the model is static. We consider a version with linear functions, adapted from Blanchard and Johnson (2013, chap. 5).

4.2 The Model

$$Y = C + I + G \quad (4.1)$$

$$C = c_0 + c_1(Y - T), \quad c_1 \in (0, 1) \quad (4.2)$$

$$I = i_0 - i_1 r, \quad i_1 > 0 \quad (4.3)$$

$$G = G_0 \quad (4.4)$$

$$T = T_0 \quad (4.5)$$

$$M_s = M_0 \quad (4.6)$$

$$M_d = m_0 + m_1 Y - m_2 r, \quad m_1 > 0 \quad (4.7)$$

$$M = M_d = M_s \quad (4.8)$$

$$N = aY, \quad a > 0 \quad (4.9)$$

$$U = 1 - \frac{N}{N^f} \quad (4.10)$$

where Y , C , I , G , T , r , M_d , M_s , N , U and N^f are output, consumption, investment, government spending, taxes, the interest rate on bonds, money demand, money supply, employment, the unemployment rate, and the labour force, respectively. The constant price level has been normalised to unity.

Equation 4.1 is the goods market equilibrium condition. Aggregate supply (Y) accommodates to the level of aggregate demand which is the sum of consumption, investment, and government spending. Equation 4.2 is the consumption function consisting of autonomous consumption demand (c_0) and a marginal propensity to consume (c_1) out of disposable income ($Y - T$). Investment demand in Equation 4.3 has an autonomous component (i_0) capturing Keynesian animal spirits and a component that is negatively related to the rate of interest on bonds. By equations Equation 4.4 and Equation 4.5, government spending and taxation are exogenous. Similarly, the money supply in Equation 4.6 is assumed to be exogenous. By Equation 4.7, households' money demand is positively related to income (capturing the transaction demand for money) and negatively related to the interest rate on bonds (capturing speculative demand). There is also an autonomous term (m_0) capturing Keynesian liquidity preference. Equation 4.9 is a fixed-coefficient production function through which employment is determined. In conjunction with an exogenously given labour force (N^f), the level of employment can be used to obtain an unemployment rate in Equation 4.10.

4.3 Simulation

4.3.1 Parameterisation

Table 1 reports the parameterisation used in the simulation. Besides a baseline (labelled as scenario 1), five further scenarios will be considered. Scenarios 2 and 3 model a switch towards pessimistic sentiments: a fall in animal spirits (i_0) and an increase in liquidity preference (m_0). Scenarios 4 to 6 consider three different government policies to stimulate the economy: a monetary expansion (M_0), a tax cut (T_0), and a fiscal expansion (G_0).

Table 1: Parameterisation

Scenario	α_0	c_1	i_0	i_1	m_0	m_1	m_2	M_0	T_0	G_0	a	N^f
1: base- line	2	0.6	2	0.1	6	0.2	0.4	5	1	1	1.5	18
2: fall in ani- mal spir- its (i_0)	2	0.6	1	0.1	6	0.2	0.4	5	1	1	1.5	18
3: in- creased liq- uid- ity pref- er- ence (m_0)	2	0.6	2	0.1	7	0.2	0.4	5	1	1	1.5	18
4: mon- e- tary ex- pan- sion (M_0)	2	0.6	2	0.1	6	0.2	0.4	6	1	1	1.5	18

Scenario	i_0	c_1	i_1	m_0	m_1	m_2	M_0	T_0	G_0	a	N^f	
5: tax cut (T_0)	2	0.6	2	0.1	6	0.2	0.4	5	0	1	1.5	18
6: fiscal expansion (G_0)	2	0.6	2	0.1	6	0.2	0.4	5	1	2	1.5	18

4.3.2 Simulation code

```
#Clear the environment
rm(list=ls(all=TRUE))

# Set number of scenarios (including baseline)
S=6

#Create vector in which equilibrium solutions from different parameterisations will be stored
Y_star=vector(length=S) # Income/output
C_star=vector(length=S) # Consumption
I_star=vector(length=S) # Investment
r_star=vector(length=S) # Real interest rate
N_star=vector(length=S) # Employment
U_star=vector(length=S) # Unemployment rate

# Set exogenous variables that will be shifted
i0=vector(length=S) # autonomous investment
m0=vector(length=S) # Autonomous demand for money
M0=vector(length=S) # money supply
G0=vector(length=S) # government spending
T0=vector(length=S) # taxes

i0[] = 2
m0[] = 6
M0[] = 5
G0[] = 1
T0[] = 1
```

```

## Construct scenarios
# scenario 2: fall in animal spirits
i0[2]=1

#scenario 3: increase in liquidity preference
m0[3]=7

# scenario 4: monetary expansion
M0[4]=6

# scenario 5: reduction in tax rate
T0[5]=0

# scenario 6: fiscal expansion
G0[6]=2

#Set constant parameter values
c0=2 # Autonomous consumption
c1=0.6 # Sensitivity of consumption with respect to the income (marginal propensity to consume)
i1=0.1 # Sensitivity of investment with respect to the interest rate
m1=0.2 # Sensitivity of money demand with respect to income
m2=0.4 # Sensitivity of money demand with respect to interest rate
a=1.5 # labour coefficient
Nf=18 # Full employment/labour force

# Initialise endogenous variables at some arbitrary positive value
Y = C = I = r = N = U = 1

#Solve this system numerically through 1000 iterations based on the initialisation
for (i in 1:S){

  for (iterations in 1:1000){

    #Model equations

    # Goods market equilibrium
    Y = C + I + G0[i]

    # Consumption demand
    C = c0 + c1*(Y-T0[i])
  }
}

```

```

# Investment demand
I = i0[i] - i1*r

# Money market, solved for interest rate
r = (m0[i] - M0[i])/m2 + m1*Y/m2

# Employment
N = a*Y

#Unemployment rate
U = (1 - N/Nf)

}

#Save results for different parameterisations in vector
Y_star[i]=Y
C_star[i]=C
I_star[i]=I
r_star[i]=r
N_star[i]=N
U_star[i]=U
}

```

 Python code

```

import numpy as np

# Set the number of scenarios (including baseline)
S = 6

# Create arrays to store equilibrium solutions from different parameterizations
Y_star = np.empty(S) # Income/output
C_star = np.empty(S) # Consumption
I_star = np.empty(S) # Investment
r_star = np.empty(S) # Real interest rate
N_star = np.empty(S) # Employment
U_star = np.empty(S) # Unemployment rate

# Set exogenous variables that will be shifted
i0 = np.zeros(S) # Autonomous investment
m0 = np.zeros(S) # Autonomous demand for money
M0 = np.zeros(S) # Money supply
G0 = np.zeros(S) # Government spending
T0 = np.zeros(S) # Taxes

# Baseline parameterisation
i0[:] = 2
m0[:] = 6
M0[:] = 5
G0[:] = 1
T0[:] = 1

# Construct scenarios
# scenario 2: fall in animal spirits
i0[1] = 1

# scenario 3: increase in liquidity preference
m0[2] = 7

# scenario 4: monetary expansion
M0[3] = 6

# scenario 5: reduction in tax rate
T0[4] = 0

# scenario 6: fiscal expansion
G0[5] = 2

```

```

# Set constant parameter values
c0 = 2 # Autonomous consumption
c1 = 0.6 # Sensitivity of consumption with respect to income (marginal propensity to consume)
i1 = 0.1 # Sensitivity of investment with respect to the interest rate
m1 = 0.2 # Sensitivity of money demand with respect to income
m2 = 0.4 # Sensitivity of money demand with respect to the interest rate
a = 1.5 # labor coefficient

```

4.3.3 Plots

Figure 15.1 and Figure 4.2 depict the response of the model's key endogenous variables, output and the interest rate, to various shifts. A fall in animal spirits (scenario 2) and an increase in liquidity preference (scenario 3) both have contractionary effects. While the fall in animal spirits directly reduces aggregate demand and thereby output (despite a fall in the interest rate), the rise in liquidity preference depresses output through its positive effect on the interest rate. Both scenarios raise the unemployment rate (Figure 4.3). Scenarios 4 to 6 assess three different macroeconomic policy tools to stimulate output. It can be seen in Figure 15.1 that fiscal policy is more effective than monetary policy for the parameterisation in Table 1.¹ Direct fiscal stimulus is more effective than tax cuts due to the constant marginal propensity to consume. The effect on output is a multiple of the one-unit stimulus thanks to the multiplier effect. However, it can also be seen that fiscal policy raises the interest rate, which crowds out some of the expansionary effect.

```
barplot(Y_star, ylab="Y", names.arg=c("1:Baseline", "2:Fall animal spirits", "3:Rise liq.
                                         "4:Monetary exp.", "5:Tax cut", "6:Fiscal exp."), ce
```

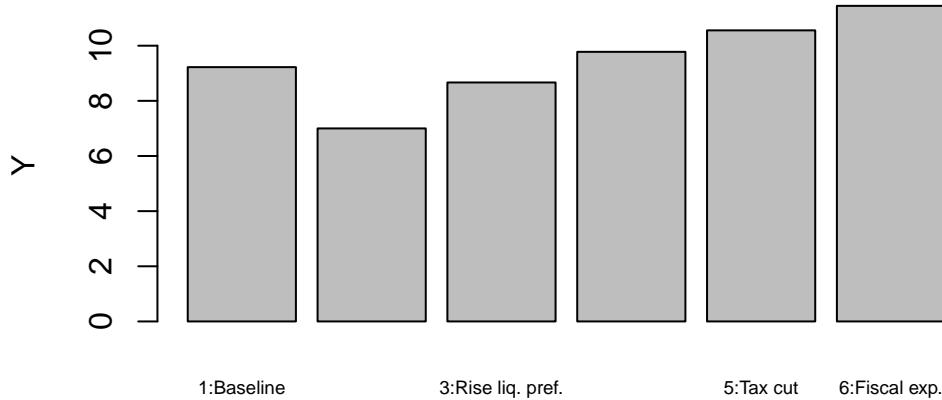


Figure 4.1: Output

¹The analytical discussion below shows formally that fiscal policy is more effective than monetary policy if $m_2 > i_2$.

```
barplot(r_star, ylab="r", names.arg=c("1:Baseline", "2:Fall animal spirits", "3:Rise liq.  
"4:Monetary exp.", "5:Tax cut", "6:Fiscal exp."), ce
```

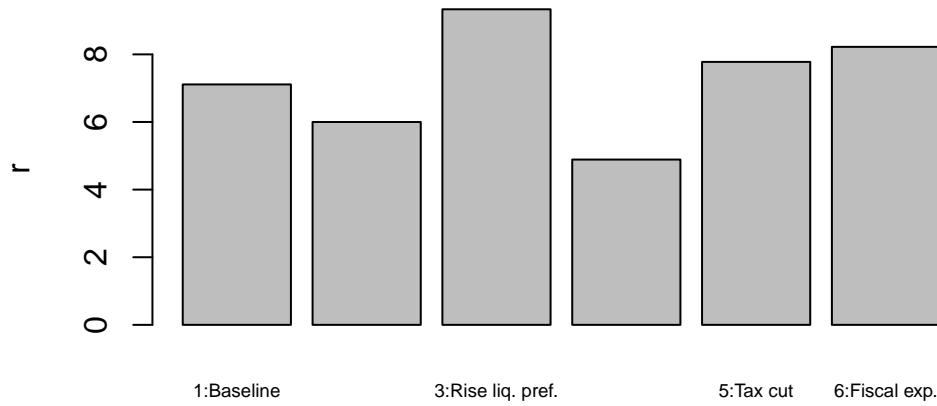


Figure 4.2: Interest rate

```
barplot(U_star*100, ylab="U (%)", names.arg=c("1:Baseline", "2:Fall animal spirits", "3:Ri  
"4:Monetary exp.", "5:Tax cut", "6:Fiscal ex
```

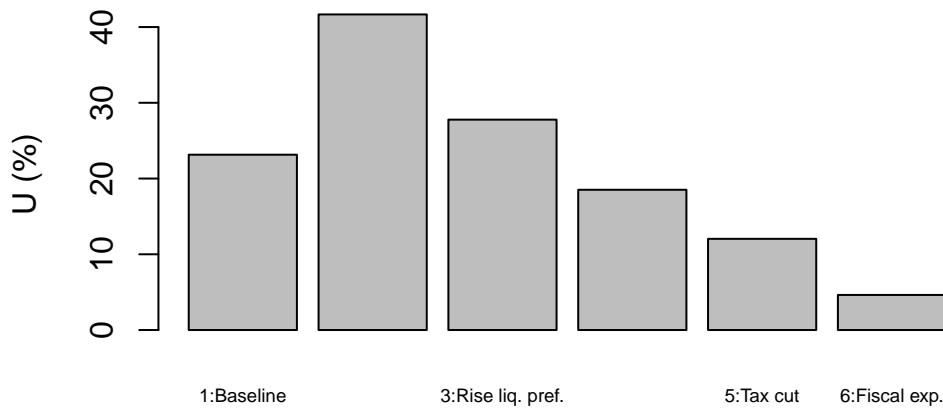


Figure 4.3: Unemployment

Figure 4.4 and Figure 4.5 further show that monetary policy mostly stimulates investment, whereas fiscal policy boost consumption.

```
barplot(I_star, ylab="I", names.arg=c("1:Baseline", "2:Fall animal spirits", "3:Rise liq.  
"4:Monetary exp.", "5:Tax cut", "6:Fiscal exp."), ce
```

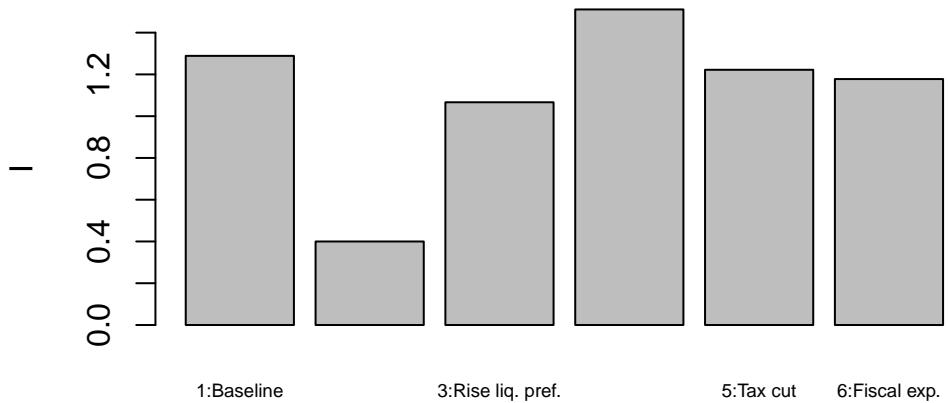


Figure 4.4: Investment

```
barplot(C_star, ylab="C", names.arg=c("1:Baseline", "2:Fall animal spirits", "3:Rise liq.  
"4:Monetary exp.", "5:Tax cut", "6:Fiscal exp."), ce
```

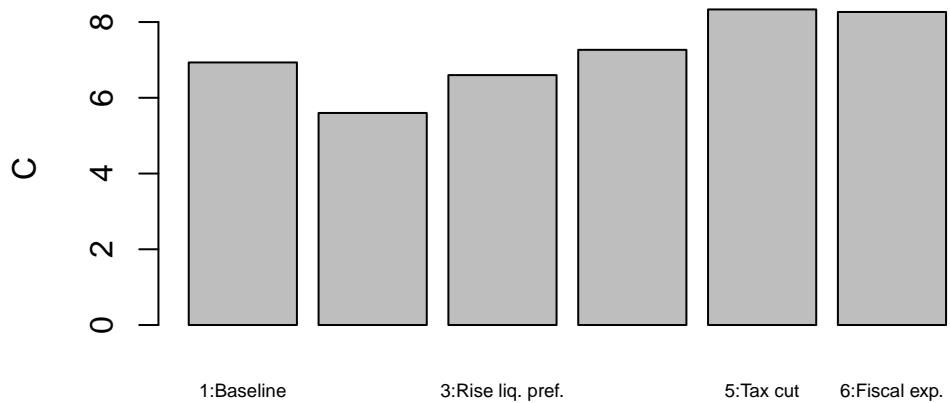


Figure 4.5: Consumption

```
barplot(N_star, ylab="N", names.arg=c("1:Baseline", "2:Fall animal spirits", "3:Rise liq.
```

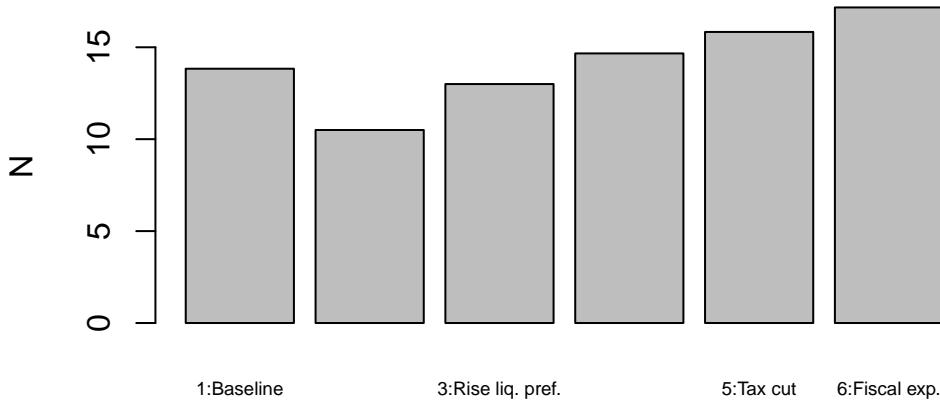


Figure 4.6: Employment

i Python code

```
# Plot results (here only for output)
import matplotlib.pyplot as plt

scenario_names = ["1:Baseline", "2:Fall animal spirits", "3:Rise liq. pref.",
                   "4:Monetary exp.", "5:Tax cut", "6:Fiscal exp."]

plt.bar(scenario_names, Y_star)
plt.ylabel('Y')
plt.xticks( scenario_names, rotation=45, fontsize=6)
plt.show()
```

4.4 Directed graph

Another perspective on the model's properties is provided by its directed graph. A directed graph consists of a set of nodes that represent the variables of the model. Nodes are connected

by directed edges. An edge directed from a node x_1 to node x_2 indicates a causal impact of x_1 on x_2 .

```
# Construct auxiliary Jacobian matrix for 11 variables: Y, C, I, G, T, r, M0, N, i0, m0, M
# where non-zero elements in regular Jacobian are set to 1 and zero elements are unchanged

M_mat=matrix(c(0,1,1,1,0,0,0,0,0,0,0,
              1,0,0,0,1,0,0,0,0,0,0,
              0,0,0,0,0,1,0,0,1,0,0,
              0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,1,0,0,0,1,
              0,0,0,0,0,0,0,0,0,0,0,
              1,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,
              1,0,0,0,0,1,0,0,0,1,0), 11, 11, byrow=TRUE)

# Create adjacency matrix from transpose of auxiliary Jacobian
A_mat=t(M_mat)

# Create directed graph from adjacency matrix
library(igraph)
dg= graph_from_adjacency_matrix(A_mat, mode="directed", weighted= NULL)

# Define node labels
V(dg)$name=c("Y", "C", "I", expression(G[0]), expression(T[0]), "r", expression(M[0]), "N",
            expression(i[0]), expression(m[0]), expression(M[0]), "M")

# Plot directed graph
plot(dg, main="", vertex.size=20, vertex.color="lightblue",
      vertex.label.color="black", edge.arrow.size=0.3, edge.width=1.1, edge.size=1.2,
      edge.arrow.width=1.2, edge.color="black", vertex.label.cex=1.2,
      vertex.frame.color="NA", margin=-0.08)
```

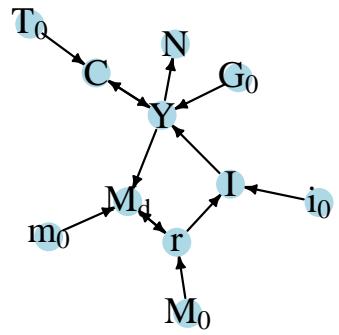


Figure 4.7: Directed graph of IS-LM model

 Python code

```

# Load relevant libraries
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

# Define the auxiliary Jacobian matrix
M_mat = np.array([
    [0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0]
])
# Create adjacency matrix from transpose of auxiliary Jacobian and add column names
A_mat = M_mat.transpose()

# Create the graph from the adjacency matrix
G = nx.DiGraph(A_mat)

# Define node labels
nodelabs = {
    0: "Y",
    1: "C",
    2: "I",
    3: r"$G_0$",
    4: r"$T_0$",
    5: "r",
    6: r"$M_0$",
    7: "N",
    8: r"$i_0$",
    9: r"$m_0$",
    10: r"$M_d$"
}

# Plot the directed graph
pos = nx.spring_layout(G, seed=43)
nx.draw(G, pos, with_labels=True, labels=nodelabs, node_size=300, node_color='lightblue',
        font_size=10)
edge_labels = {(u, v): '' for u, v in G.edges}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='black')
plt.axis('off')
plt.show()

```

4.5 Analytical discussion

To obtain the IS-curve, substitute Equation 4.2 - Equation 4.5 into Equation 4.1 and solve for Y :

$$Y = \left(\frac{1}{1 - c_1} \right) (c_0 + i_0 + G_0 - i_1 r - c_1 T_0). \quad (4.11)$$

To obtain the LM-curve, substitute Equation 4.6 - Equation 4.7 into Equation 4.8 and solve for r :

$$r = \left(\frac{1}{m_2} \right) (m_0 - M_0 + m_1 Y). \quad (4.12)$$

Finally, to obtain equilibrium solutions for Y and r , substitute Equation 4.12 into Equation 4.11 and vice versa:

$$Y^* = \left[\frac{m_2}{(1 - c_1)m_2 + i_1 m_1} \right] (c_0 + i_0 + G_0 - c_1 T_0) + \left[\frac{i_1}{(1 - c_1)m_2 + i_1 m_1} \right] (M_0 - m_0)$$

$$r^* = \left[\frac{1 - c_1}{(1 - c_1)m_2 + i_1 m_1} \right] (m_0 - M_0) + \left[\frac{m_1}{(1 - c_1)m_2 + i_1 m_1} \right] (c_0 + i_0 + G_0 - c_1 T_0).$$

From this, the following results can be obtained:

- The equilibrium effects of a change in taxes are smaller than those from a change in government spending (since c_1 is smaller than one).
- Government spending is more effective than monetary expansion if $m_2 > i_1$ (which is the case for the parameterisation in Table 1).
- The equilibrium multiplier $\left[\frac{m_2}{(1 - c_1)m_2 + i_1 m_1} \right]$ is smaller than the aggregate demand multiplier $\left(\frac{1}{1 - c_1} \right)$ due to the positive effect on the interest rate and the corresponding negative effect investment ($i_1 m_1$). This is the crowding out mechanism.

4.5.1 Calculate equilibrium fiscal multiplier

```
Y_star[6]-Y_star[1] # numerical approach
```

```
[1] 2.222222
```

```
m2/((1-c1)*m2+i1*m1) # analytical approach
```

```
[1] 2.222222
```

 Python code

```
Y_star[5]-Y_star[0] # numerical approach
```

```
m2/((1-c1)*m2+i1*m1) # analytical approach
```

References

5 A Neoclassical Synthesis Model (IS-LM-AS-AD)

5.1 Overview

The Neoclassical Synthesis was developed in the 1940s and 1950s by Franco Modigliani, Paul Samuelson and others. It introduced neoclassical components into the Keynesian IS-LM model that had been proposed by John R. Hicks (1937) to formalise some key ideas of John Maynard Keynes' 1936 book [The General Theory of Employment, Interest and Money](#). The IS-LM model, which is analysed in detail in Chapter 4, contains two equilibrium relationships: a goods market equilibrium between investment and saving (IS) and a money market equilibrium between money demand and money supply (LM). In the goods market, aggregate supply adjusts to the level of aggregate demand given by the expenditure decisions of households, firms, and the government. Money demand is determined by aggregate income and the interest rate on bonds. The money supply is assumed to be exogenous. The two markets pin down equilibrium output and the interest rate.

The Neoclassical Synthesis adds a neoclassical labour market with Keynesian frictions to the IS-LM model. Following the discussion in Froyen (2005), chap. 9, we consider a labour market in which firms have perfect information about the real wage, whereas workers need to form expectations about the price level. Price expectations are assumed to be exogenous in the short run. Workers thus suffer from ‘money illusion’: an increase in the actual price levels reduces the real wage but leaves their labour supply unchanged. This gives rise to an upward-sloping aggregate supply (AS) (or Phillips) curve. By contrast, the aggregate demand (AD) curve is downward-sloping as a higher price level increases the demand for real money balances, which pushes up the interest rate.

In this short-run model, prices are flexible but the capital stock is fixed. The focus is thus on goods market equilibrium rather than economic growth. As all endogenous variables adjust instantaneously, the model is static. We consider a version with a Cobb-Douglas production function and otherwise linear behavioural functions, based on the graphical analysis in Froyen (2005), chap. 9.

5.2 The Model

$$Y = C + I + G_0 \quad (5.1)$$

$$C = c_0 + c_1(Y - T_0), \quad c_1 \in (0, 1) \quad (5.2)$$

$$I = i_0 - i_1 r, \quad i_1 > 0 \quad (5.3)$$

$$M_s = M_0 \quad (5.4)$$

$$\frac{M_d}{P} = m_0 + m_1 Y - m_2 r, \quad m_1, m_2 > 0 \quad (5.5)$$

$$M_d(r) = M_s \quad (5.6)$$

$$w = (1 - a) A K^a N^{-a}, \quad a \in (0, 1) \quad (5.7)$$

$$W = \frac{P^e b C}{1 - \frac{N^f}{N^f}}, \quad b \in (0, 1) \quad (5.8)$$

$$P = \frac{W}{w} \quad (5.9)$$

$$N = \left(\frac{Y}{A K^a} \right)^{\frac{1}{1-a}} \quad (5.10)$$

$$U = 1 - \frac{N}{N^f} \quad (5.11)$$

where Y , C , I , G_0 , T_0 , r , M_s , M_d , w , A , K , N , W , P^e , N^f , P , and U are output, consumption, investment, (exogenous) government spending, (exogenous) taxes, the real interest rate on bonds, nominal money supply, nominal money demand, the real wage, productivity, the capital stock, employment, the nominal wage, the price level expected by workers, the labour force (or total available time for work), the actual price level, and the unemployment rate, respectively.

Equation 6.1 is the goods market equilibrium condition. Aggregate supply (Y) accommodates to the level of aggregate demand which is the sum of consumption, investment, and government spending. Equation 5.2 is the consumption function consisting of autonomous consumption demand (c_0) and a marginal propensity to consume (c_1) out of disposable income ($Y - T_0$). Investment demand in Equation 5.3 has an autonomous component (i_0) capturing Keynesian animal spirits and a component that is negatively related to the rate of interest on bonds. Government spending and taxation are exogenous. Similarly, the nominal money supply (M_0) in Equation 5.4 is assumed to be exogenous. By Equation 5.5, households' real money demand is positively related to income (capturing the transaction demand for money) and negatively related to the interest rate on bonds (capturing speculative demand). There is also an autonomous term (m_0) capturing Keynesian liquidity preference. Equilibrium in the money market Equation 5.6 yields an equation for the interest rate.

In Equation 6.11, the real wage is determined by the marginal product of labour implied by a Cobb-Douglas production function ($Y = AK^a N^{1-a}$). This means the real wage is always consistent with firms' demand for labour based on profit-maximisation.¹ Equation 6.10 specifies the nominal wage as implied by households' labour supply curve. Optimising households supply labour based on their work-leisure trade-off (with the parameter b capturing their preference for leisure, $1 - \frac{N}{N^f}$). Since they don't have knowledge of the current real wage, they base their decisions on the expected price level P^e , which is exogenous in the short run. The actual price level is then given by the ratio of the nominal wage to the real wage Equation 6.9. In other words, firms set prices such that the nominal wage they pay to workers are consistent with their own desired real wage. Equation 6.12 pins down employment as implied by the Cobb-Douglas production function. In conjunction with an exogenously given labour force N^f (or total available labour time), the level of employment can be used to obtain an unemployment rate in Equation 6.13.

5.3 Simulation

5.3.1 Parameterisation

Table 1 reports the parameterisation used in the simulation. Besides a baseline (labelled as scenario 1), five further scenarios will be considered. Scenario 2 is a switch towards pessimistic sentiments in the form of a fall in animal spirits (i_0). In scenario 3, productivity (A) increases. Scenario 4 considers a rise in the price level expected by workers (P^e). Scenarios 5 and 6 consider two different government policies to stimulate the economy: a monetary expansion (M_0) and a fiscal expansion (G_0).

Table 1: Parameterisation

¹See the notes on the Classical Model ([here](#)) for a formal derivation of the labour demand and supply curves from optimisation. A minor modification is that here we work with a normalisation of the term for leisure in the household's log-utility function, $\ln(1 - \frac{N}{N^f})$, to allow N to be larger than unity.

Scenario	c_0	c_1	i_0	i_1	A	P^e	m_0	m_1	m_2	M_0	G_0	T_0	N^f	a	b
1: baseline	2	0.6	2	0.1	2	1	6	0.2	0.4	5	1	1	7	0.3	0.4
2: fall in animal spirits (i_0)	2	0.6	1.5	0.1	2	1	6	0.2	0.4	5	1	1	7	0.3	0.4
3: rise in productivity (A)	2	0.6	2	0.1	3	1	6	0.2	0.4	5	1	1	7	0.3	0.4
4: rise in expected price level (P^e)	2	0.6	2	0.1	2	1.5	6	0.2	0.4	5	1	1	7	0.3	0.4
5: monetary expansion (M_0)	2	0.6	2	0.1	2	1	6	0.2	0.4	6	1	1	7	0.3	0.4
6: fiscal expansion (G_0)	2	0.6	2	0.1	2	1	6	0.2	0.4	5	2	1	7	0.3	0.4

5.3.2 Simulation code

```
#Clear the environment
rm(list=ls(all=TRUE))

# Set number of scenarios (including baseline)
S=6

#Create vector in which equilibrium solutions from different parameterisations will be stored
Y_star=vector(length=S) # Income/output
C_star=vector(length=S) # Consumption
I_star=vector(length=S) # Investment
r_star=vector(length=S) # Real interest rate
N_star=vector(length=S) # Employment
U_star=vector(length=S) # Unemployment rate
P_star=vector(length=S) # Price level
w_star=vector(length=S) # Real wage
W_star=vector(length=S) # Nominal wage

# Set exogenous variables that will be shifted
i0=vector(length=S) # autonomous investment (animal spirits)
M0=vector(length=S) # money supply
G0=vector(length=S) # government spending
P0=vector(length=S) # expected price level
```

```

A=vector(length=S) # Exogenous productivity

#### Construct different scenarios
# baseline
A[]=2
i0[]=2
M0[]=5
G0[]=1
P0[]=1

# scenario 2: fall in animal spirits
i0[2]=1.5

# scenario 3: increase in productivity
A[3]=3

# scenario 4: increase in expected price level
P0[4]=1.5

# scenario 5: monetary expansion
M0[5]=6

# scenario 6: fiscal expansion
G0[6]=2

#Set constant parameter values
c0=2 # Autonomous consumption
c1=0.6 # Sensitivity of consumption with respect to the income (marginal propensity to consume)
i1=0.1 # Sensitivity of investment with respect to the interest rate
m1=0.2 # Sensitivity of money demand with respect to income
m2=0.4 # Sensitivity of money demand with respect to interest rate
Nf=5 # Full employment/labour force
K=4 # Exogenous capital stock
a=0.3 # Capital elasticity of output
b=0.4 # household preference for leisure
T0=1 # tax revenues
m0=6 # liquidity preference

# Initialise endogenous variables at some arbitrary positive value
Y = C = I = r = P = w = N = W = 1

```

```

#Solve this system numerically through 1000 iterations based on the initialisation

for (i in 1:S){

  for (iterations in 1:1000){

    #Model equations

    # Goods market equilibrium
    Y = C + I + G0[i]

    # Consumption demand
    C = c0 + c1*(Y-T0)

    # Investment demand
    I = i0[i] - i1*r

    # Money market, solved for interest rate
    r = (m0 - (M0[i]/P))/m2 + m1*Y/m2

    #Unemployment rate
    U = (1 - N/Nf)

    #Real wage
    w = A[i]*(1-a)*(K^a)*N^(-a)

    #Nominal wage
    W= (P0[i]*b*C)/(1- (N/Nf))

    #Price level
    P = W/w

    #Employment
    N = (Y/(A[i]*(K^a)))^(1/(1-a))

  }

  #Save results for different parameterisations in vector
  Y_star[i]=Y
  C_star[i]=C
  I_star[i]=I
}

```

```
r_star[i]=r  
N_star[i]=N  
U_star[i]=U  
P_star[i]=P  
w_star[i]=w  
W_star[i]=W  
}  
}
```

 Python code

```

import numpy as np

# Set the number of scenarios (including baseline)
S = 6

# Create arrays to store equilibrium solutions from different parameterizations
Y_star = np.empty(S) # Income/output
C_star = np.empty(S) # Consumption
I_star = np.empty(S) # Investment
r_star = np.empty(S) # Real interest rate
N_star = np.empty(S) # Employment
U_star = np.empty(S) # Unemployment rate
P_star = np.empty(S) # Price level
w_star = np.empty(S) # Real wage
W_star = np.empty(S) # Nominal wage

# Set exogenous variables that will be shifted
i0 = np.zeros(S) # Autonomous investment (animal spirits)
M0 = np.zeros(S) # Money supply
G0 = np.zeros(S) # Government spending
P0 = np.zeros(S) # Expected price level
A = np.empty(S) # Exogenous productivity

# Construct different scenarios
# baseline
A[:] = 2
i0[:] = 2
M0[:] = 5
G0[:] = 1
P0[:] = 1

# scenario 2: fall in animal spirits
i0[1] = 1.5

# scenario 3: increase in productivity
A[2] = 3

# scenario 4: increase in expected price level

```

5.3.3 Plots

Figures Figure 15.1 - Figure 5.5 depict the response of the model's key endogenous variables to various shifts. A fall in animal spirits (scenario 2) reduces aggregate demand and thereby output and employment (despite a fall in the interest rate). This reduces workers' nominal wage demands and thus the price level. An increase in productivity (scenario 3) has expansionary effects on output but adverse effects on employment. Higher productivity means that fewer workers need to be hired to produce the same level of output. However, the corresponding reduction in employment also reduces the price level, which lowers the (real) demand for money and thus lowers the interest rate. This has expansionary effects on output.

```
barplot(Y_star, ylab="Y", names.arg=c("1:Baseline", "2:Fall animal spirits", "3:Rise produ  
"4:Rise exp. price", "5:Monetary expan.", "6:Fiscal
```

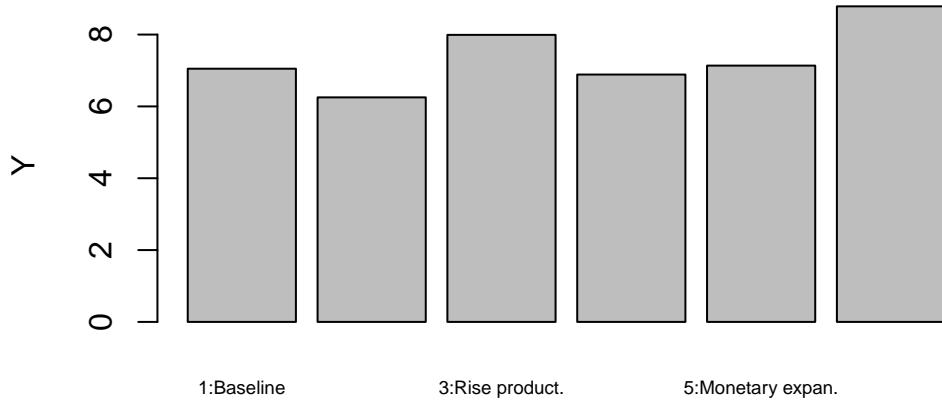


Figure 5.1: Output

```
barplot(P_star, ylab="P", names.arg=c("1:Baseline", "2:Fall animal spirits", "3:Rise produ  
"4:Rise exp. price", "5:Monetary expan.", "6:Fiscal
```

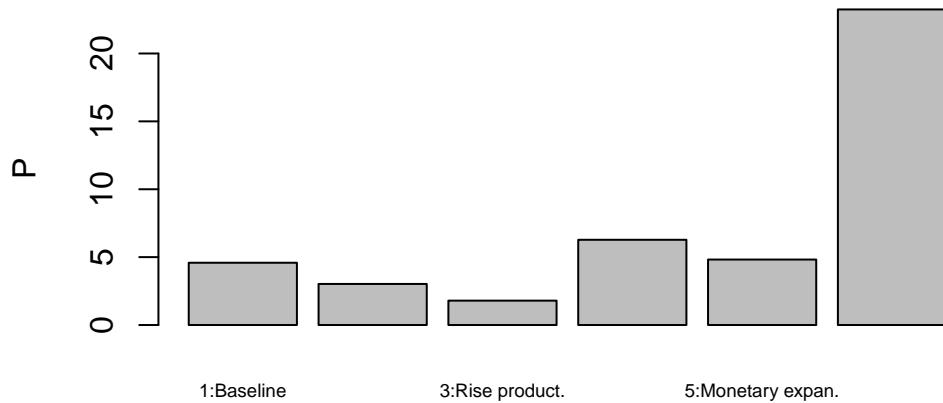


Figure 5.2: Price level

```
barplot(r_star, ylab="r", names.arg=c("1:Baseline", "2:Fall animal spirits", "3:Rise produc  
"4:Rise exp. price", "5:Monetary expan.", "6:Fiscal
```

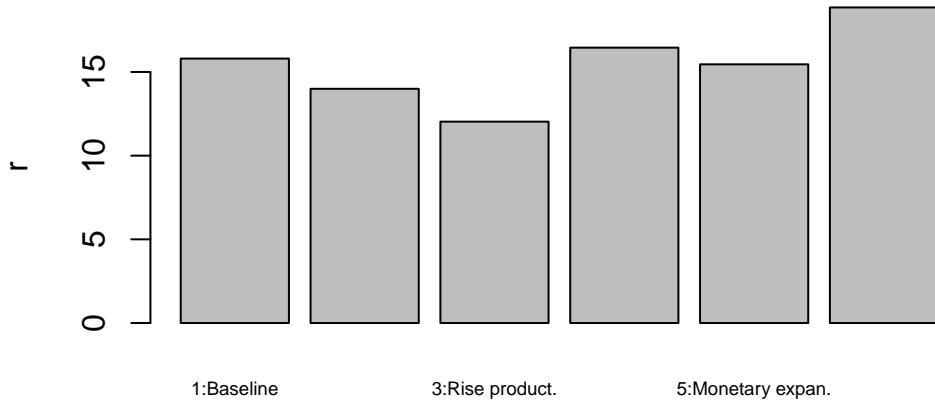


Figure 5.3: Interest rate

A rise in the expected price level (scenario 4) raises nominal wages and thereby the actual price level. This raises the interest rate, which exerts a (small) contractionary effect on output and employment. Scenarios 5 and 6 assess two different macroeconomic policy tools to stimulate output. A monetary expansion lowers the interest rate and increases output but also the price level. Similar results arise for a fiscal expansion. The main difference is that the monetary expansion lowers the interest rate, whereas the fiscal expansion increases it. \

```
barplot(U_star*100, ylab="U (%)", names.arg=c("1:Baseline", "2:Fall animal spirits", "3:Ri
                                         "4:Rise exp. price", "5:Monetary expan.", "6:Fiscal expan.")
```

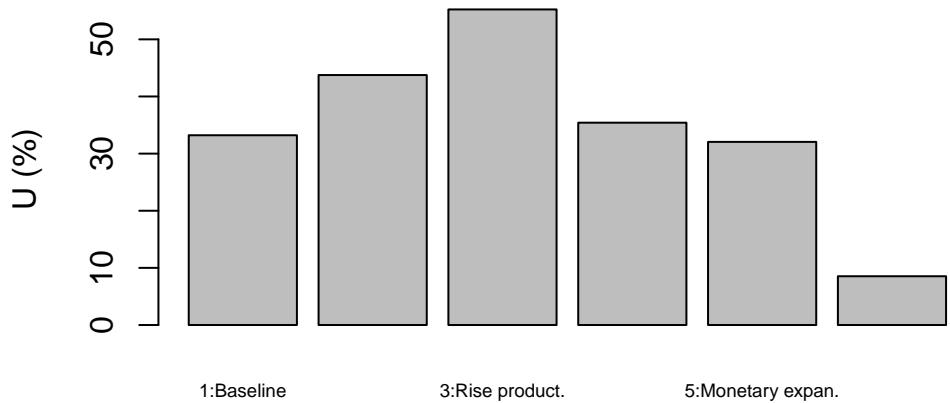


Figure 5.4: Unemployment rate

```
barplot(W_star, ylab="W", names.arg=c("1:Baseline", "2:Fall animal spirits", "3:Rise produc  
"4:Rise exp. price", "5:Monetary expan.", "6:Fiscal"))
```

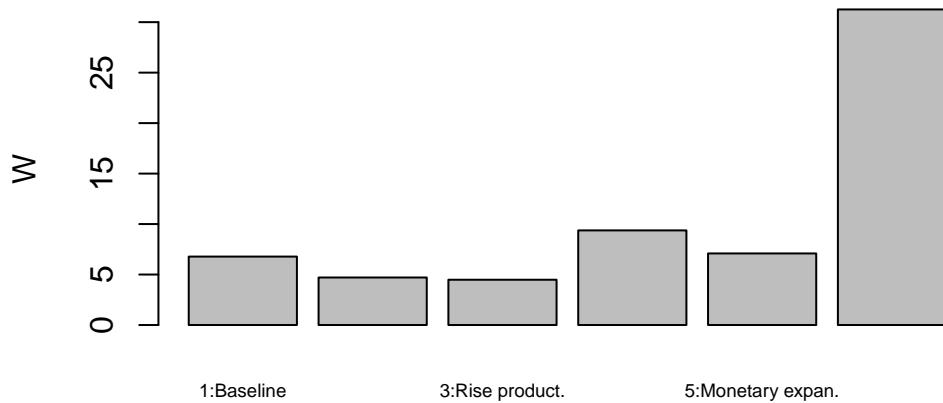


Figure 5.5: Nominal wage

```
barplot(w_star, ylab="W", names.arg=c("1:Baseline", "2:Fall animal spirits", "3:Rise produc  
"4:Rise exp. price", "5:Monetary expan.", "6:Fiscal"))
```

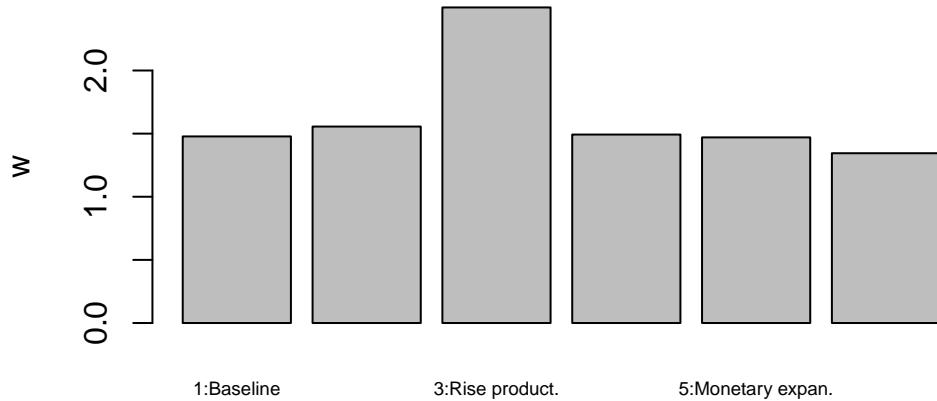


Figure 5.6: Real wage

i Python code

```
# Plot results (here only for output)
import matplotlib.pyplot as plt

scenario_names = ["1:Baseline", "2:Fall animal spirits", "3:Rise product.",
                   "4:Rise exp. price", "5:Monetary expan.", "6:Fiscal expan."]

plt.bar(scenario_names, Y_star)
plt.ylabel('Y')
plt.xticks(scenario_names, rotation=45, fontsize=6)
plt.show()
```

5.4 Directed graph

Another perspective on the model's properties is provided by its directed graph. A directed graph consists of a set of nodes that represent the variables of the model. Nodes are connected

by directed edges. An edge directed from a node x_1 to node x_2 indicates a causal impact of x_1 on x_2 .

```
## Create directed graph
# Construct auxiliary Jacobian matrix for 15 variables: Y, C, I, G, T, r, w, W, P, MO, N,
# Mmat=matrix(c(0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
#                 1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
#                 0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,
#                 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
#                 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
#                 1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,
#                 0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,
#                 0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,
#                 0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,
#                 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
#                 1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
#                 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
#                 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
#                 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
#                 1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0), 15, 15, byrow=TRUE)

# Create adjacency matrix from transpose of auxiliary Jacobian
Amat=t(Mmat)

# Create directed graph from adjacency matrix
library(igraph)
dg= graph_from_adjacency_matrix(Amat, mode="directed", weighted= NULL)

# Define node labels
V(dg)$name=c("Y", "C", "I", expression(G[0]), expression(T[0]), "r", "w", "W", "P", express

# Plot directed graph
plot(dg, main="", vertex.size=20, vertex.color="lightblue",
      vertex.label.color="black", edge.arrow.size=0.3, edge.width=1.1, edge.size=1.2,
      edge.arrow.width=1.2, edge.color="black", vertex.label.cex=1.2,
      vertex.frame.color="NA", margin=-0.08)
```

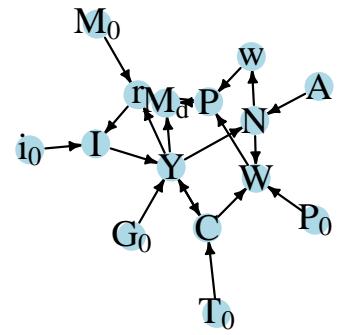


Figure 5.7: Directed graph of Neoclassical Synthesis model

 Python code

```

# Load relevant libraries
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

# Define the Jacobian matrix
M_mat = np.array([
    [0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
    [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
])

# Create adjacency matrix from transpose of auxiliary Jacobian and add column names
A_mat = M_mat.transpose()

# Create the graph from the adjacency matrix
G = nx.DiGraph(A_mat)

# Define node labels
nodelabs = {
    0: "Y",
    1: "C",
    2: "I",
    3: r"$G_0$",
    4: r"$T_0$",
    5: "r",
    6: "w",
    7: "W",
    8: "P",
    9: r"$M_0$",
    10: "N",
    11: r"$i_0$",
    12: "A",
    13: r"$P_0$",
    14: r"$M_d$",
}

# Plot the directed graph

```

In Figure 5.7, it can be seen that productivity (A), taxes (T_0), government spending (G_0), animal spirits (i_0), the money supply (M_0), and the expected price level (P_0) are the key exogenous variables of the model. All other variables are endogenous and form a closed loop (or cycle) within the system. The lower-left part of the graph captures the goods market (IS): aggregate demand (consumption, investment, and government spending) determines output. The upper part of the graph contains the labour market, which determines the price level. Finally, the lower-right part of the graph represents the money market (LM), which determines the interest rate. There is a two-way feedback between the goods market and the money market as output impacts the demand for money, and the interest rate affects investment. There is also a feedback from output into the labour market through employment. The labour market feeds into the money market via its effect on prices and thus money demand, which then also feeds into the goods market through the interest rate.

5.5 Analytical discussion

In the first step, we will reduce the system to three equations: an IS-curve, an LM-curve, and an AS-curve (or Phillips curve). In the second step, the IS-curve and the LM-curve are combined to yield an AD-curve.

To obtain the IS-curve, substitute Equation 5.2 -Equation 5.3 into Equation 6.1 and solve for Y :

$$Y = \left(\frac{1}{1 - c_1} \right) (c_0 + i_0 + G_0 - i_1 r - c_1 T_0). \quad (5.12)$$

To obtain the LM-curve, substitute Equation 5.4 - Equation 5.5 into Equation 5.6 and solve for r :

$$r = \left(\frac{1}{m_2} \right) (m_0 - \frac{M_0}{P} + m_1 Y). \quad (5.13)$$

To obtain the AS-curve, substitute Equation 6.11, Equation 6.10, Equation 6.12, and Equation 5.2 into Equation 6.9 :

$$P = \frac{b(c_0 - c_1 T_0) + P^e b c_1 Y}{(1 - a) \left[(A K^a Y^{-a})^{\frac{1}{1-a}} - \frac{Y}{N^f} \right]}. \quad (5.14)$$

It can readily be seen that the AS-curve is upward-sloping in the (Y, P) -space (recall that $a \in (0, 1)$).

Finally, to obtain the AD-curve, substitute Equation 4.12 into Equation 4.11 :

$$Y = \left[\frac{m_2(c_0 + i_0 + G_0 - c_1 T_0) + i_1(\frac{M_0}{P} - m_0)}{(1 - c_1)m_2 + i_1 m_1} \right] \quad (5.15)$$

It can readily be seen that the AD-curve is downward-sloping in the (Y, P) -space.

References

6 A Post-Keynesian Macro Model with Endogenous Money

6.1 Overview

Post-Keynesian Economics is an economic paradigm that was developed in the 1930s and 1940s by Joan Robinson, Nicholas Kaldor and others.¹ The early post-Keynesian economists sought to develop further key ideas of John Maynard Keynes. They were critical of the **Neoclassical Synthesis** that introduced neoclassical elements into the Keynesian framework. Among many other points, post-Keynesians argued that money is created by commercial banks. Money creation is determined by the demand for credit rather than being under the control of the central bank, and should thus be considered endogenous. Post-Keynesians further assigned a key role to financial factors in the determination of economic activity, but also considered finance as a source of instability. They abandoned the neoclassical approach of deriving labour demand and supply from optimising behaviour and instead assumed oligopolistic market structures. Firms set prices by charging a mark-up on costs and workers set nominal wages based on their bargaining power.

Fontana and Setterfield (2009) present a simple model that could be regarded as a post-Keynesian alternative to the Neoclassical Synthesis. The model highlights the endogenous money creation process. Money is being created when commercial banks make loans to accommodate the demand for credit by creditworthy borrowers. The demand for credit is driven by aggregate demand. The interest rate on loans is determined by the base rate, set by the central bank, on which commercial banks charge a mark-up. Although credit creation is demand-driven, some borrowers will be credit constrained. In times of financial crises, banks tighten credit constraints, which can depress economic activity.

In this short-run model, prices are flexible but the capital stock is fixed. The focus is thus on goods market equilibrium rather than economic growth. As all endogenous variables adjust instantaneously, the model is static. We consider a version of the model due to Fontana and Setterfield (2009) with linear functions.

¹See Lavoie (2006), chap.1 and [Exploring Economics](#) for introductions. Lavoie (2014) and Hein (2014) provide more advanced treatments.

6.2 The Model

$$Y = ND + cD \quad (6.1)$$

$$ND = bY, \quad b \in (0, 1) \quad (6.2)$$

$$D = d_0 - d_1 r, \quad d_1 > 0 \quad (6.3)$$

$$i = i_0 + i_1 P, \quad i_1 > 0 \quad (6.4)$$

$$r = (1 + m)i, \quad m > 0 \quad (6.5)$$

$$dL = cD \quad (6.6)$$

$$dM = dL \quad (6.7)$$

$$dR = k dM, \quad k \in (0, 1) \quad (6.8)$$

$$P = (1 + n)aW, \quad a, n > 0 \quad (6.9)$$

$$W = W_0 - hU, \quad h > 0 \quad (6.10)$$

$$w = \frac{1}{(1 + n)a} \quad (6.11)$$

$$N = aY \quad (6.12)$$

$$U = 1 - \frac{N}{N^f} \quad (6.13)$$

where Y , ND , D , r , i , P , dL , dM , dR , W , w , N , U , and N^f are output, the not debt-financed component of aggregate demand, the desired debt-financed component of aggregate demand,

the lending rate, the policy rate, the price level, the change in loans, the change in money (bank deposits), the change in bank reserves, the nominal wage, the real wage, employment, the unemployment rate, and full employment (or total labour supply), respectively.

Equation 6.1 is the goods market equilibrium condition. Aggregate supply (Y) accommodates to the level of aggregate demand which is the sum of a not debt-financed component (ND) and a (desired) debt-financed component (D). The coefficient c is the proportion of loan applications that are deemed creditworthy and thus captures credit rationing by banks. By Equation 6.2, the not debt-financed component of aggregate demand is a function of current income. In Equation 6.3, the debt-financed component of aggregate demand has an autonomous component (d_0)² and is otherwise negatively related to the lending rate r . Equation 6.4 specifies the monetary policy rule, where it is assumed that the central bank raises the policy rate i when the price level increases.³[This specification is somewhat unrealistic given that most modern central banks target a positive rate of inflation. However, it allows for an AS-AD representation of the model, which facilitates the comparison with the Neoclassical Synthesis model (Chapter 5)] The lending rate in Equation 8.1 is given by a mark-up m that banks charge on the policy rate (which is the rate at which they can borrow reserves). The change in loans in Equation 6.6 is equal to the creditworthy demand for loans (cD). This captures the demand-driven nature of credit creation. The changes in loans translates one-to-one into a change in money, which are bank deposits in this model (Equation 6.7). This reflects the endogenous money creation process where commercial banks create new deposits when they make new loans. By Equation 6.8, banks obtain new reserves from the central bank to maintain a constant reserve-to-deposit ratio k . Thus, the causality in this model runs from debt-financed demand to loans, to deposits, and finally to reserves.

By Equation 6.9, the price level is set by firms based on a mark-up (n) on unit labour cost (which are the product of the nominal wage W and the labour coefficient a). Nominal wages are set by workers based on their bargaining power, which is declining in the unemployment rate (Equation 6.13). The real wage in Equation 6.11 is derived from the pricing Equation 6.9, i.e. through their price setting power, firms ultimately determine the real wage. The level of employment in Equation 6.12 is determined residually based on economic activity and a constant-coefficient production function ($Y = \frac{N}{a}$). Finally, the level of employment in conjunction with an exogenously given labour force N^f (or total available labour time) can be used to obtain an unemployment rate in Equation 6.13.

²For simplicity, it is assumed that all autonomous demand is debt-financed, i.e. there is no spending out of wealth.

6.3 Simulation

6.3.1 Parameterisation

Table 1 reports the parameterisation used in the simulation. Besides a baseline (labelled as scenario 1), five further scenarios will be considered. Scenario 2 is a rise in credit rationing in the form of a fall in c . In scenario 3, autonomous credit-financed demand (d_0) increases. Scenarios 4 and 5 consider a rise in the interest rate (or bank) mark-up (m) and in the price (or firm) mark-up (n), respectively. Scenario 6 considers a rise in productivity reflected in a fall of the labour coefficient a .

Table 1: Parameterisation

Scenario	b	c	d_0	d_1	i_0	$i1$	m	k	n	W_0	h	a	N^f
1: baseline	0.5	0.8	5	0.8	0.01	0.5	0.15	0.3	0.15	2	0.8	0.8	12
2: rise in credit rationing (c)	0.5	0.4	5	0.8	0.01	0.5	0.15	0.3	0.15	2	0.8	0.8	12
3: rise in autonomous demand (d_0)	0.5	0.8	10	0.8	0.01	0.5	0.15	0.3	0.15	2	0.8	0.8	12
4: rise in bank mark-up (m)	0.5	0.8	5	0.8	0.01	0.5	0.3	0.3	0.15	2	0.8	0.8	12
5: rise in firm mark-up (n)	0.5	0.8	5	0.8	0.01	0.5	0.15	0.3	0.3	2	0.8	0.8	12
6: rise in productivity (a)	0.5	0.8	5	0.8	0.01	0.5	0.15	0.3	0.15	2	0.8	0.4	12

6.3.2 Simulation code

```
#Clear the environment
rm(list=ls(all=TRUE))

# Set number of scenarios (including baseline)
S=6

#Create vector in which equilibrium solutions from different parameterisations will be stored
Y_star=vector(length=S) # income/output
D_star=vector(length=S) # (notional) credit-financed aggregate demand
ND_star=vector(length=S) # income-financed aggregate demand
r_star=vector(length=S) # lending rate
N_star=vector(length=S) # employment
```

```

U_star=vector(length=S) # unemployment
P_star=vector(length=S) # price level
w_star=vector(length=S) # real wage
W_star=vector(length=S) # nominal wage
i_star=vector(length=S) # central bank rate
dL_star=vector(length=S) # change in loans
dM_star=vector(length=S) # change in bank deposits
dR_star=vector(length=S) # change in bank reserves

# Set exogenous variables that will be shifted
c=vector(length=S) # share of credit demand that is accommodated
d0=vector(length=S) # autonomous component of debt-financed aggregate demand
m=vector(length=S) # mark-up on lending rate
n=vector(length=S) # mark-up on prices
a=vector(length=S) # productivity

# Baseline parameterisation
c[] = 0.8
d0[] = 5
m[] = 0.15
n[] = 0.15
a[] = 0.8

## Construct scenarios

# scenario 2: increase in credit rationing
c[2] = 0.4

# scenario 3: increase in autonomous demand
d0[3] = 10

# scenario 4: increase in interest rate mark-up
m[4] = 0.3

# scenario 5: increase in price mark-up
n[5] = 0.3

# scenario 6: increase in productivity
a[6] = 0.4

```

```

#Set constant parameter values
b=0.5 # propensity to spend out of income
d1=0.8 # sensitivity of demand with respect to the interest rate
i0=0.01 # discretionary component of central bank rate
i1=0.5 # sensitivity of central bank rate with respect to price level
Nf=12 # full employment/labour force
h=0.8 # sensitivity of nominal wage with respect to unemployment
k=0.3 # desired reserve ratio
W0=2 # exogenous component of nominal wage

# Initialise endogenous variables at some arbitrary positive value
Y = D = ND = r = N = U = P = w = W = i = dL = dR = dM = 1

#Solve this system numerically through 1000 iterations based on the initialisation
for (j in 1:S){

  for (iterations in 1:1000){

    #Model equations

    # (1) Goods market
    Y = ND + c[j]*D

    # (2) Net-debt financed component of aggregate demand
    ND = b*Y

    # (3) Debt-financed component of aggregate demand
    D= d0[j] - d1*r

    # (4) Policy rate
    i = i0 + i1*P

    # (5) Lending rate
    r = (1+m[j])*i

    # (6) Change in loans
    dL = c[j]*D

    # (7) Change in deposits
    dM = dL
  }
}

```

```

# (8) Change in reserves
dR = k*dM

# (9) Price level
P = (1+n[j])*a[j]*W

# (10) Nominal wage
W = W0 - h*(U)

# (11) Real wage
w = 1/((1+n[j])*a[j])

# (12) Employment
N = a[j]*Y

# (13) Unemployment rate
U = (Nf - N)/Nf

}

#Save results for different parameterisations in vector
Y_star[j]=Y
D_star[j]=D
ND_star[j]=ND
r_star[j]=r
N_star[j]=N
U_star[j]=U
P_star[j]=P
w_star[j]=w
W_star[j]=W
i_star[j]=i
dL_star[j]=dL
dM_star[j]=dM
dR_star[j]=dR
}

```

 Python code

```

# Load NumPy library
import numpy as np

# Set the number of scenarios (including baseline)
S = 6

# Create arrays to store equilibrium solutions
Y_star = np.zeros(S) # income/output
D_star = np.zeros(S) # (notional) credit-financed aggregate demand
ND_star = np.zeros(S) # income-financed aggregate demand
r_star = np.zeros(S) # lending rate
N_star = np.zeros(S) # employment
U_star = np.zeros(S) # unemployment
P_star = np.zeros(S) # price level
w_star = np.zeros(S) # real wage
W_star = np.zeros(S) # nominal wage
i_star = np.zeros(S) # central bank rate
dL_star = np.zeros(S) # change in loans
dM_star = np.zeros(S) # change in bank deposits
dR_star = np.zeros(S) # change in bank reserves

# Set exogenous variables that will be shifted
c = np.zeros(S) # share of credit demand that is accommodated
d0 = np.zeros(S) # autonomous component of debt-financed aggregate demand
m = np.zeros(S) # mark-up on lending rate
n = np.zeros(S) # mark-up on prices
a = np.zeros(S) # productivity

# Baseline parameterisation
c[:] = 0.8
d0[:] = 5
m[:] = 0.15
n[:] = 0.15
a[:] = 0.8

# Construct scenarios
# Scenario 2: increase in credit rationing
c[1] = 0.4

# Scenario 3: increase in autonomous demand

```

6.3.3 Plots

Figures Figure 6.1 - Figure 6.7 depict the response of the model's key endogenous variables to various shifts.

```
barplot(Y_star, ylab="Y", names.arg=c("1:baseline", "2:rise credit rat.", "3:rise AD",
                                         "4:rise bank markup", "5:rise firm markup", "6:rise p
```

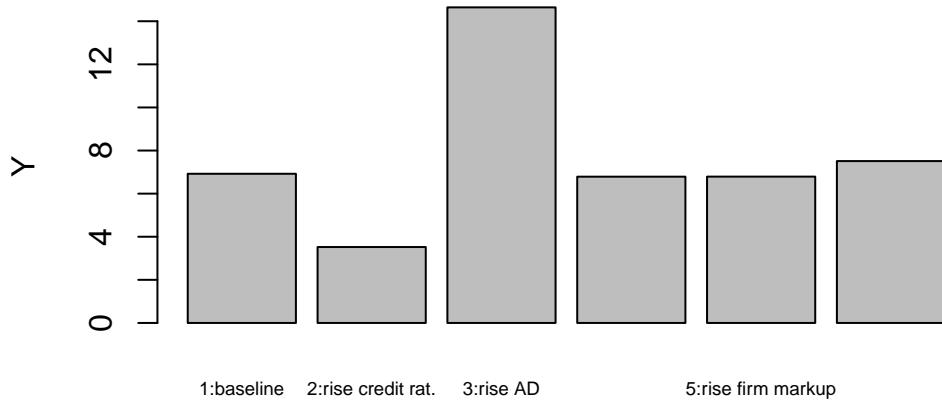


Figure 6.1: Output

An increase in credit rationing (scenario 2) reduces deposit money creation as well as actual (as opposed to desired) aggregate demand. This drags down output and employment. The rise in unemployment reduces workers' nominal wage demands and thus the price level. The lending rate falls as the central bank reduces the policy rate.

```
barplot(P_star, ylab="P", names.arg=c("1:baseline", "2:rise credit rat.", "3:rise AD",
                                         "4:rise bank markup", "5:rise firm markup", "6:rise p
```

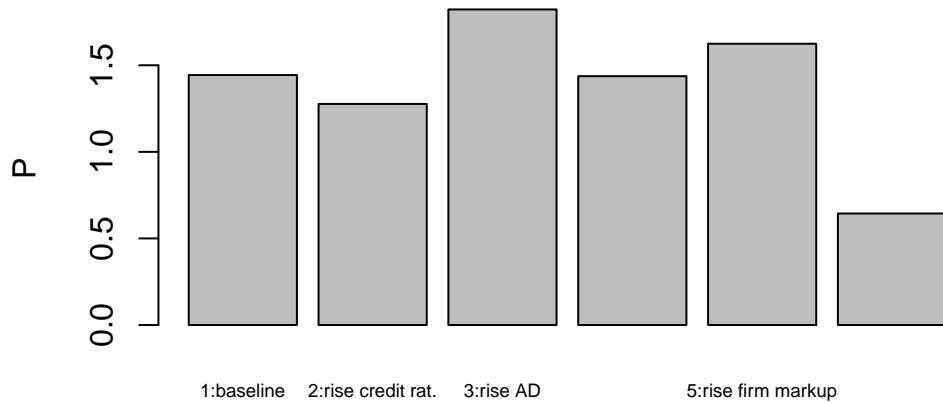


Figure 6.2: Price level

An increase in (debt-financed) autonomous demand (scenario 3) has expansionary effects on output and employment. The money stock accommodates through increased loan creation. The increase in workers' bargaining power leads to higher nominal wages and prices. The central bank reacts by raising the policy rate but this does not completely offset the expansionary effect.

```
barplot(r_star, ylab="r", names.arg=c("1:baseline", "2:rise credit rat.", "3:rise AD",
                                         "4:rise bank markup", "5:rise firm markup", "6:rise p")
```

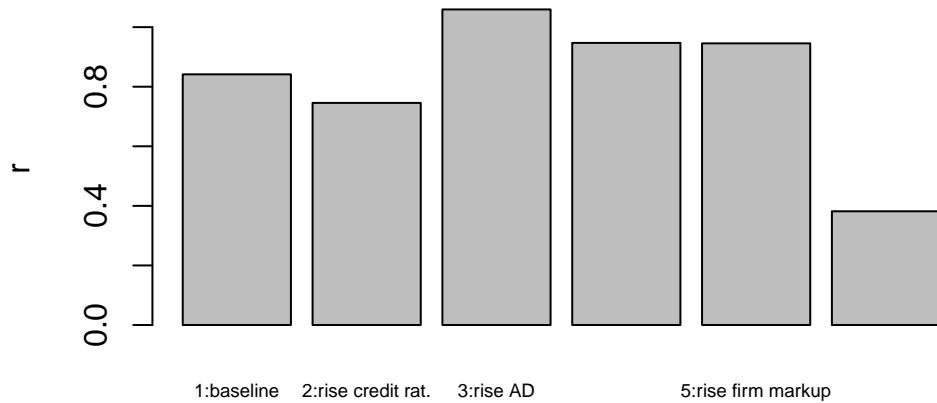


Figure 6.3: Lending rate

```
barplot(dM_star, ylab="dM", names.arg=c("1:baseline", "2:rise credit rat.", "3:rise AD",
                                         "4:rise bank markup","5:rise firm markup", "6:rise"))
```

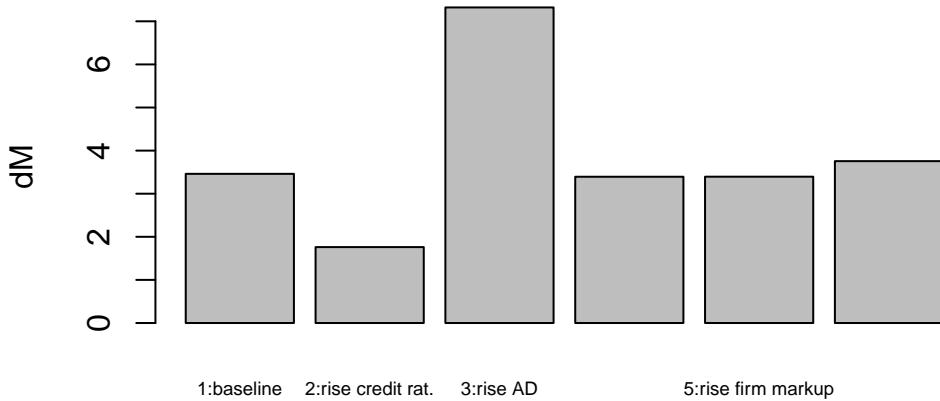


Figure 6.4: Deposit money creation

In scenarios 4 and 5, the interest rate (or bank) mark-up and the price (or firm) mark-up increase, respectively. The increase in the bank mark-up raises the lending rate, which has a contractionary effect. The increase in the firm mark-up raises the price level, which has a contractionary effect through the monetary policy response. Notably, the rise in the price mark-up reduces the real wage.

```
barplot(U_star*100, ylab="U (%)", names.arg=c("1:baseline", "2:rise credit rat.", "3:rise
                                              "4:rise bank markup","5:rise firm markup", "
```

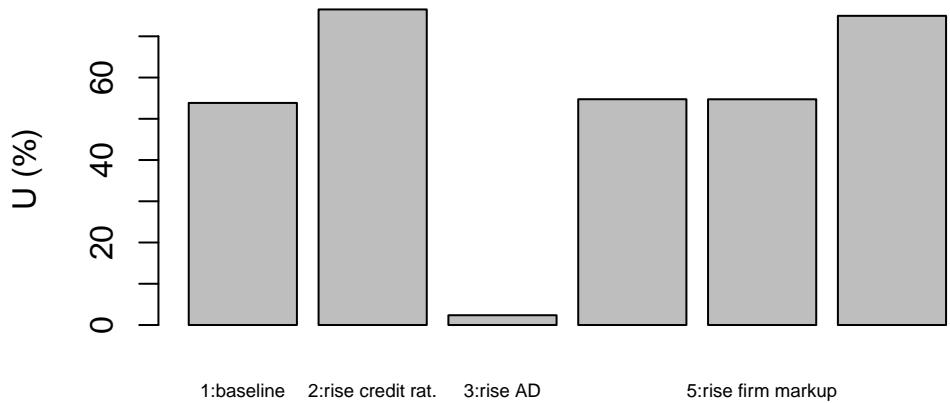


Figure 6.5: Unemployment rate

```
barplot(W_star, ylab="W", names.arg=c("1:baseline", "2:rise credit rat.", "3:rise AD",
                                         "4:rise bank markup", "5:rise firm markup", "6:rise p"))
```

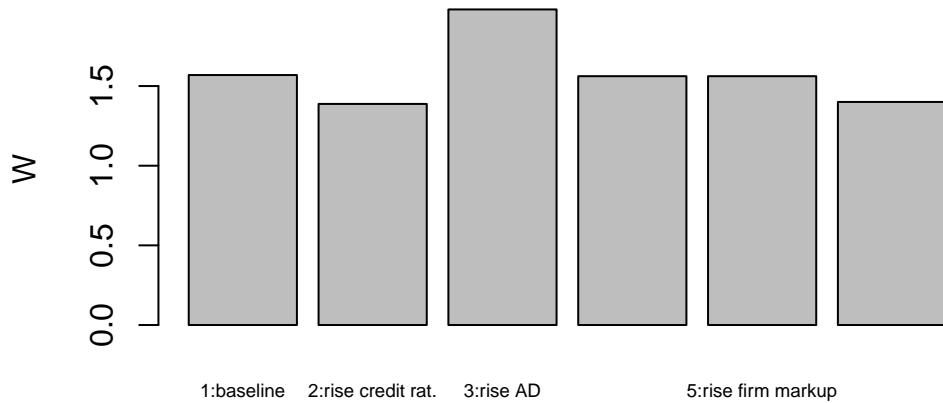


Figure 6.6: Nominal wage

Finally, an increase in productivity (scenario 6) reduces the price level, which induces a lower policy rate, leading to a small expansionary effect. However, it increases the unemployment rate as fewer workers are needed to produce the same level of output. This reduces the nominal wage, but raises the real wage.

```
barplot(w_star, ylab="w", names.arg=c("1:baseline", "2:rise credit rat.", "3:rise AD",
                                         "4:rise bank markup", "5:rise firm markup", "6:rise p"))
```

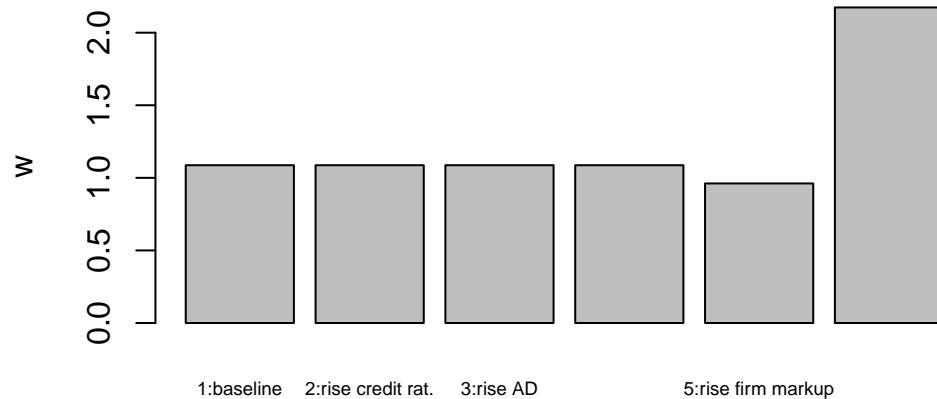


Figure 6.7: Real wage

i Python code

```
# Plot results (here for output only)
import matplotlib.pyplot as plt

scenario_names = ["1:baseline", "2:rise credit rat.", "3:rise AD", "4:rise bank markup",
                  "5:rise firm markup"]

# Create a bar plot
plt.bar(scenario_names, Y_star)
plt.ylabel("Y")
plt.xticks(rotation=45, ha="right") # Rotate x-axis labels for better readability
plt.tight_layout() # Ensure the labels fit within the plot area

# Show the plot
plt.show()
```

6.4 Directed graph

Another perspective on the model's properties is provided by its directed graph. A directed graph consists of a set of nodes that represent the variables of the model. Nodes are connected by directed edges. An edge directed from a node x_1 to node x_2 indicates a causal impact of x_1 on x_2 .

```
## Create directed graph
# Construct auxiliary Jacobian matrix for 18 variables:
# r, Y, ND, D, i, P, W, w, N, U, dL, dM, dR, dO, c, m, a, n

M_mat=matrix(c(0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,
              0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
              0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
              0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,
              0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,
              0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,
              0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
              0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
              0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
              0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0),
              18, 18, byrow=TRUE)

# Create adjacency matrix from transpose of auxiliary Jacobian
A_mat=t(M_mat)

# Create directed graph from adjacency matrix
library(igraph)
dg= graph_from_adjacency_matrix(A_mat, mode="directed", weighted= NULL)

# Define node labels
V(dg)$name=c("r", "Y", "ND", "D", "i", "P", "W", "w", "N", "U", "dL", "dM", "dR", "dO", "c", "m", "a", "n")

# Plot directed graph
```

```
plot(dg, main="", vertex.size=20, vertex.color="lightblue",
  vertex.label.color="black", edge.arrow.size=0.3, edge.width=1.1, edge.size=1.2,
  edge.arrow.width=1.2, edge.color="black", vertex.label.cex=1.2,
  vertex.frame.color="NA", margin=-0.08)
```

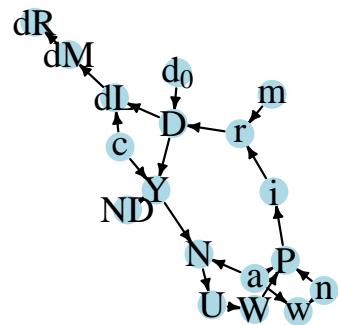


Figure 6.8: Directed graph of post-Keynesian endogenous money model

 Python code

```

#Load relevant libraries
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

# Construct auxiliary Jacobian matrix for 18 variables
#   r   Y   ND  D   i   P   W   w   N   U   dL  dM  dR  d0   c   m   a   n

M_mat = np.array([[0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0],
                  [0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0],
                  [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                  [1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0],
                  [0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1],
                  [0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1],
                  [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0],
                  [0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
                  [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                  [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]])

# Create adjacency matrix from transpose of auxiliary Jacobian and add column names
A_mat = M_mat.transpose()

# Create the graph from the adjacency matrix
G = nx.DiGraph(A_mat)

# Define node labels
nodelabs = {0: "r", 1: "Y", 2: "ND", 3: "D", 4: "i", 5:"P", 6: "W", 7: "w",
            8: "N", 9: "U", 10: "dL", 11: "dM", 12: "dR", 13: "d0", 14: "c",
            15: "m", 16: "a", 17: "n"}

# Plot the directed graph
pos = nx.spring_layout(G, seed=42)
nx.draw(G, pos, with_labels=True, labels=nodelabs, node_size=300, node_color='lightblue',
        font_size=10)
edge_labels = {(u, v): '' for u, v in G.edges}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='black')
plt.axis('off')
plt.show()

```

In Figure Figure 6.8, it can be seen that credit rationing (c), productivity (a), the price mark-up (n), the interest rate mark-up (m), and autonomous demand (d_0), are the key exogenous variables of the model. All other variables are endogenous, and many of them form a closed loop (or cycle) within the system. The lower-right part of the graph captures the goods market: debt- and not debt-financed aggregate demand determine output. The outer right part depicts the endogenous money creation process: creditworthy debt-financed demand determines credit creation, which translates into deposit money creation. Bank reserves are a residual. The lower-left part of the graph represents the labour market. The goods market feeds into the labour market via employment, which determines nominal wages and the price level. The real wage is a residual. The price level feeds into interest rate determination in the upper part of the model, which establishes a causal feedback link from the labour market to the goods market.

6.5 Analytical discussion

Like the Neoclassical Synthesis model, the post-Keynesian macro model can be represented as an AS-AD model. First, we will derive an IS and an MP curve in the (Y, r) -space, the latter representing monetary policy instead of the money market (the conventional LM curve). Then we obtain the AS-AD representation of the model in the (Y, P) -space. Finally, we obtain equilibrium solutions for Y and P .

To obtain the IS-curve, substitute Equation 6.2 - Equation 8.1 into Equation 6.1 and solve for Y :

$$Y = \left(\frac{1}{1-b} \right) [c(d_0 - d_1 r)]. \quad (\text{IS})$$

To obtain the MP-curve, substitute Equation 6.4, Equation 6.9, Equation 6.10, Equation 6.12, and Equation 6.13 into Equation 8.1 :

$$r = (1+m) \left[i_0 + i_1 (1+n) a [W_0 - h (1 - \frac{aY}{N^f})] \right]. \quad (\text{MP})$$

It can readily be seen that the IS-curve is downward-sloping and the MP-curve is upward-sloping in the (Y, r) -space

To obtain the AD-curve, substitute Equation 8.1 and Equation 6.4 into the IS-curve:

$$Y = \left(\frac{1}{1-b} \right) [c(d_0 - d_1(1+m)(i_0 + i_1 P))]. \quad (\text{AD})$$

Finally, to obtain the AS-curve, substitute Equation 6.10 and Equation 6.13 into Equation 6.9:

$$P = (1 + n) a \left[W_0 - h \left(1 - \frac{aY}{N^f} \right) \right]. \quad (\text{AS})$$

It can readily be seen that the AD-curve is downward-sloping in the (Y, P) -space, whereas the AS-curve is upward-sloping.

Finally, by substituting the AS and AD curve into each other, we obtain the following equilibrium solutions for output and the price level:

$$\begin{aligned} Y^* &= \frac{c[d_0 - d_1(1 + m)(i_1(1 + n)a(W_0 - h))]}{1 - b + cd_1(1 + m)i_1(1 + n)a^2h(N^f)^{-1}} \\ P^* &= \frac{(1 + n)a[(1 - b)(W_0 - h) + ha(N^f)^{-1}c(d_0 - d_1(1 + m)i_0)]}{1 - b + cd_1(1 + m)i_1(1 + n)a^2h(N^f)^{-1}}. \end{aligned}$$

References

7 A Kaldor-Robinson Distribution and Growth Model

7.1 Overview

In the 1950s and 1960s in Cambridge UK, Nicholas Kaldor and Joan Robinson developed a theory of growth that aimed to apply John Maynard Keynes' principle of effective demand to the long run.¹ The main Keynesian assumption retained by Kaldor and Robinson was that investment and saving are independent, and that a change in investment may lead to an adjustment in saving. However, unlike Keynes, Kaldor and Robinson assumed a fixed level of capacity utilisation, which they considered a key feature of a long-run equilibrium. As a result, goods market clearing cannot be established via output adjustment. Instead, Kaldor and Robinson assumed price adjustment, which would translate into a change in income distribution. Changes in the distribution of income then affect consumption (and saving), as workers tend to have a higher marginal propensity to consume than capital owners. For example, an increase in investment demand due to improved animal spirits would then lead to excess demand, which raises the price level. For a given level of nominal wages, the rise in the price level lowers real wages, leading to a redistribution of income towards profits. The resulting rise in the profit share increases aggregate saving, thereby leading to an adjustment of saving to investment. Taken together, the Kaldor-Robinson approach highlights the relevance of supply constraints in the long run that can lead to inflationary outcomes of demand shocks.

We consider a simple version of the model proposed in Hein (2014), chap. 4.4. This is a model of long-run steady state growth. In the steady state, all endogenous variables grow at the same rate.² Changes in parameters or exogenous variables lead to an instantaneous adjustment of the model's variables, so that the model can be analysed like a static one. The key question addressed by this model is how changes in aggregate demand affects income distribution and the rate of growth.

7.2 The Model

$$r = h \frac{u_n}{v} \tag{7.1}$$

¹See Hein (2014), chap. 4 for a detailed treatment.

²All variables are normalised by the capital stock and thus rendered stationary.

$$s = s_{\Pi}r, \quad s_{\Pi} \in (0, 1) \quad (7.2)$$

$$c = \frac{u_n}{v} - s \quad (7.3)$$

$$g = g_0 + g_1 r, \quad g_1 > 0 \quad (7.4)$$

$$h = \frac{vg_0}{u_n(s_{\Pi} - g_1)} \quad (7.5)$$

where r , s , c , g , and h are the profit rate, the saving rate, the consumption rate, the investment rate, and the profit share, respectively.

Equation 7.1 decomposes the profit rate into the product of the profit share h (total profits over total output), the normal rate of capacity utilisation (u_n), and the inverse of v (the capital-potential output ratio). Let Y be output, K be the capital stock, and Y^P be potential output, then the decomposition can also be written as $r = \frac{\Pi}{K} = \frac{\Pi}{Y} \frac{Y^P}{Y^P} \frac{Y^P}{K}$. The normal rate of capacity utilisation and the capital-potential output ratio are taken to be exogenous in this model. Note also that the wage share is given by $1 - h$. By Equation 7.2, the economy-wide saving rate is given by saving out of profits ($s_{\Pi}r$). It is assumed that workers don't save, i.e. have a higher marginal propensity to consume than capital owners. Equation 7.3 simply states that consumption is income not saved. According to Equation 7.3, investment is determined by an autonomous component g_0 that may capture Keynesian 'animal spirits' and by the profit rate. The profit rate may stimulate investment if firms use adaptive expectations and predict higher future profits in response to an increase in the current profit rate. Finally, Equation 7.5 is the goods market equilibrium condition $g = s$ solved for the profit share, reflecting the fact that prices are assumed to clear the goods market which translates into an adjustment of the profit share.

7.3 Simulation

7.3.1 Parameterisation

Table 1 reports the parameterisation used in the simulation. We will consider three different parameterisations. Besides a baseline scenario (labelled as scenario 1), we will consider an increase in animal spirits (g_0) and an increase in the propensity to save out of profits (s_{Π}).

Table 1: Parameterisation

Scenario	v	s_{Π}	g_0	g_1	u_n
1: baseline	3	0.6	0.02	0.3	0.9
2: rise in animal spirits (g_0)	3	0.6	0.04	0.3	0.9
3: rise in saving propensity (s_{Π})	3	0.9	0.02	0.3	0.9

7.3.2 Simulation code

```

#Clear the environment
rm(list=ls(all=TRUE))

# Set number of scenarios (including baselines)
S=3

#Create vector in which equilibrium solutions from different parameterisations will be stored
h_star=vector(length=S) # profit share
g_star=vector(length=S) # growth rate of capital stock
s_star=vector(length=S) # saving rate
c_star=vector(length=S) # consumption rate
r_star=vector(length=S) # profit rate

# Set constant parameter values
v=3      # capital-to-potential output ratio
g1=0.3 # sensitivity of investment with respect to profit rate
un=0.9 # normal rate of capacity utilisation

# Set exogenous variables whose parameterisation changes across regimes
g0=vector(length=S) # animal spirits
sp=vector(length=S) # propensity to save out of profits

#### Construct different scenarios

# scenario 1: baseline
g0[] = 0.02
sp[] = 0.6

#scenario 2: increase in animal spirits
g0[2] = 0.04

# scenario 3: increase in propensity to save out of profits
sp[3] = 0.9

```

```

#Check stability condition for all scenarios
for (i in 1:S){
  print(sp[i]>g1)
}

[1] TRUE
[1] TRUE
[1] TRUE

# Initialise endogenous variables at some arbitrary positive value
g = r = s = c = h = 1

#Solve this system numerically through 1000 iterations based on the initialisation
for (i in 1:S){

  for (iterations in 1:1000){

    #(1) Profit rate
    r=(h*un)/v

    #(2) Saving
    s = sp[i]*r

    #(3) Consumption
    c= un/v - s

    #(4) Investment
    g = g0[i]+g1*r

    #(5) Goods market equilibrium profit share
    h=(v/un)*(g0[i]/(sp[i]-g1))

  }

  #Save results for different parameterisations in vector
  h_star[i]=h
  g_star[i]=g
  r_star[i]=r
  s_star[i]=s
  c_star[i]=c
}

```

 Python code

```

import numpy as np

# Clear the environment (not necessary in Python)
# Set number of scenarios (including baselines)
S = 3

# Create arrays to store equilibrium solutions for different parameterizations
h_star = np.empty(S) # profit share
g_star = np.empty(S) # growth rate of capital stock
s_star = np.empty(S) # saving rate
c_star = np.empty(S) # consumption rate
r_star = np.empty(S) # profit rate

# Set constant parameter values
v = 3 # capital-to-potential output ratio
g1 = 0.3 # sensitivity of investment with respect to profit rate
un = 0.9 # normal rate of capacity utilization

# Set exogenous variables whose parameterization changes across regimes
g0 = np.empty(S) # animal spirits
sp = np.empty(S) # propensity to save out of profits

# Construct different scenarios
# Scenario 1: baseline
g0[:] = 0.02
sp[:] = 0.6

# Scenario 2: increase in animal spirits
g0[1] = 0.04

# Scenario 3: increase in propensity to save out of profits
sp[2] = 0.9

# Check stability condition for all scenarios
for i in range(S):
    print(sp[i] > g1)

# Initialize endogenous variables at some arbitrary positive value
g = r = s = c = h = 1

# Solve this system numerically through 1000 iterations based on the initialization
for i in range(S):
    for iterations in range(1000):i38
        # (1) Profit rate
        r = (h * un) / v

        # (2) Saving
        s = sp[i] * r

        # (3) Consumption

```

7.3.3 Plots

Figures Figure 7.1 - Figure 7.3 depict the response of the model's key endogenous variables to changes in aggregate demand. A rise in animal spirits (scenario 2) raises the profit share. This reduces consumption. However, the effect on capital accumulation and thus growth is positive. In that sense, long-run growth is demand-driven, despite the fixed rate of capacity utilisation.

```
barplot(h_star, ylab="h", names.arg=c("1: baseline", "2: rise animal spirits", "3:rise sa
```

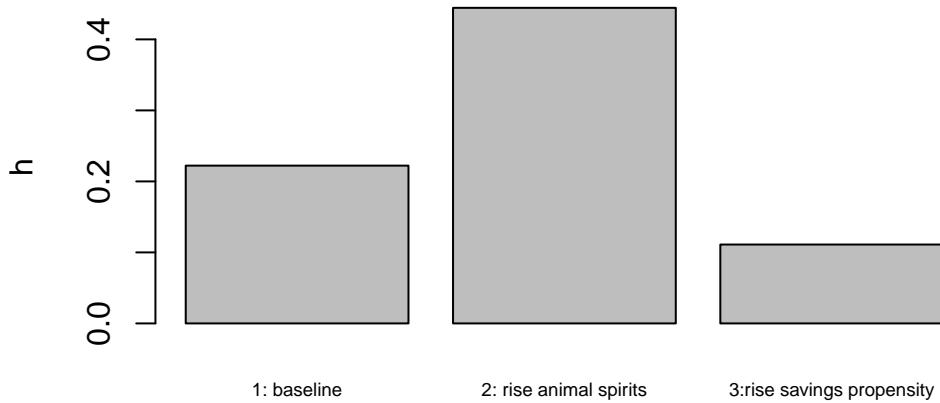


Figure 7.1: Profit share

In the second scenario, the saving propensity of capital owners increases. This constitutes a reduction in aggregate demand, leading to a fall in the profit share, and a fall in the growth rate. Since $g = s$, the effect reflects the Keynesian ‘paradox of saving’: a rise in the saving propensity leads to a fall in the aggregate saving rate.

```
barplot(c_star, ylab="c", names.arg=c("1: baseline", "2: rise animal spirits", "3:rise sa
```

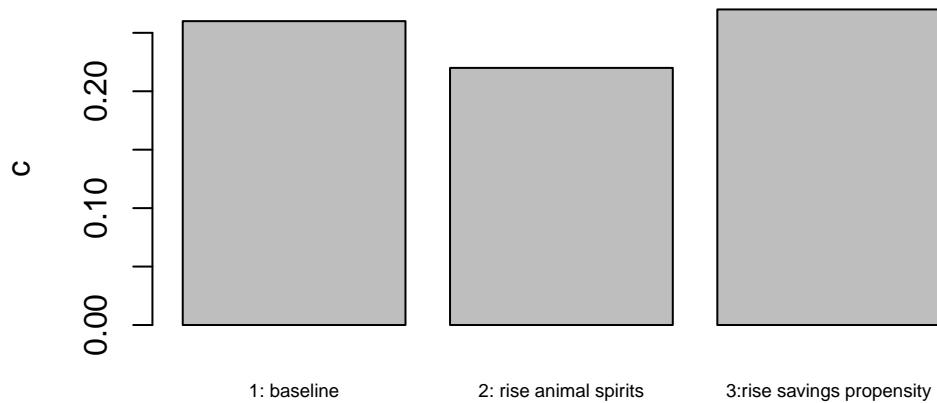


Figure 7.2: Rate of consumption

```
barplot(g_star, ylab="g", names.arg=c("1: baseline", "2: rise animal spirits", "3:rise savings propensity"))
```

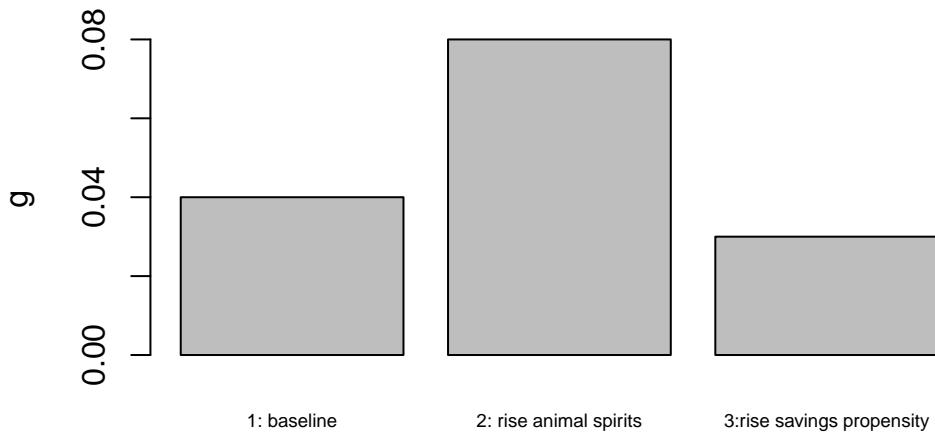


Figure 7.3: Rate of growth

Python code

```
# Plot results (here only for profit share)
import matplotlib.pyplot as plt

# Scenario labels
scenario_names = ["1: baseline", "2: rise animal spirits", "3: rise savings propensity"]

# Bar plot for h_star
plt.bar(scenario_names, h_star)
plt.ylabel('h')
plt.xticks(scenario_names, rotation=45, fontsize=6)
plt.show()
```

7.4 Directed graph

Another perspective on the model's properties is provided by its directed graph. A directed graph consists of a set of nodes that represent the variables of the model. Nodes are connected

by directed edges. An edge directed from a node x_1 to node x_2 indicates a causal impact of x_1 on x_2 .

```
## Create directed graph
# Construct auxiliary Jacobian matrix for 7 variables:
# r,h,s,g,g0,sp,un
M_mat=matrix(c(0,1,0,0,0, 0, 1,
              0,0,1,1,0, 0, 1,
              1,0,0,0,0, 1, 0,
              1,0,0,0,1, 0, 0,
              0,0,0,0,0, 0, 0,
              0,0,0,0,0, 0, 0,
              0,0,0,0,0, 0, 0), 7, 7, byrow=TRUE)

# Create adjacency matrix from transpose of auxiliary Jacobian
A_mat=t(M_mat)

# Create directed graph from adjacency matrix
library(igraph)
dg= graph_from_adjacency_matrix(A_mat, mode="directed", weighted= NULL)

# Define node labels
V(dg)$name=c("r", "h", "s", "g", expression(g[0]), expression(s[Pi]), expression(u[n]))

# Plot directed graph
plot(dg, main="", vertex.size=20, vertex.color="lightblue",
      vertex.label.color="black", edge.arrow.size=0.3, edge.width=1.1, edge.size=1.2,
      edge.arrow.width=1.2, edge.color="black", vertex.label.cex=1.2,
      vertex.frame.color="NA", margin=-0.08)
```

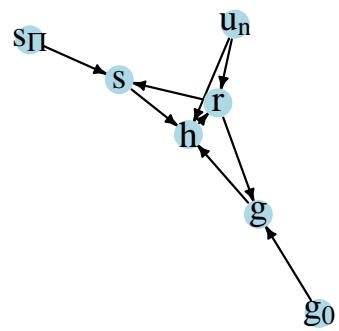


Figure 7.4: Directed graph of Kaldor-Robinson growth model

 Python code

```
# Load relevant libraries
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

# Define the Jacobian matrix
M_mat = np.array([[0, 1, 0, 0, 0, 0, 1],
                  [0, 0, 1, 0, 0, 0, 1],
                  [1, 0, 0, 0, 0, 1, 0],
                  [1, 0, 0, 0, 1, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0]])

# Create adjacency matrix from transpose of auxiliary Jacobian and add column names
A_mat = M_mat.transpose()

# Create the graph from the adjacency matrix
G = nx.DiGraph(A_mat)

# Define node labels
nodelabs = {0: "r", 1: "h", 2: "s", 3: "g", 4: r"$g_0$", 5: r"$s_p$", 6: r"$u_n$"}

# Plot the directed graph
pos = nx.spring_layout(G, seed=43)
nx.draw(G, pos, with_labels=True, labels=nodelabs, node_size=300, node_color='lightblue',
        font_size=10)
edge_labels = {(u, v): '' for u, v in G.edges}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='black')
plt.axis('off')
plt.show()
```

In Figure Figure 7.4, it can be seen that animal spirits (g_0), the propensity to save out of profits (s_Π), and the normal rate of capacity utilisation (u_n) are the key exogenous variable of the model. Saving (s), investment (g), the rate of profit (r), and the profit share (h) are endogenous and form a closed loop (or cycle) within the system.

7.5 Analytical discussion

To find the equilibrium solution for the profit share (Equation 7.5), substitute Equation 7.1, Equation 7.4 and Equation 7.2 and into $g = s$ and solve for h .

The equilibrium solution for h can then be substituted into Equation 7.1 to find:

$$r^* = \frac{g_0}{s_{\Pi} - g_1} \quad (7.6)$$

Similarly, substituting the equilibrium solution for r into Equation 7.4 yields:

$$g^* = \frac{s_{\Pi}g_0}{s_{\Pi} - g_1}. \quad (7.7)$$

The Kaldor-Robinson stability condition requires $s_{\Pi} - g_1 > 0$, i.e. saving needs to react more strongly to the profit rate than investment.

Further analytical results can be found in Hein (2014), chap. 4.4.

7.5.1 Calculate analytical solutions numerically

```
# Profit rate
for (i in 1:S){
  print((g0[i])/(sp[i]-g1))
}

[1] 0.06666667
[1] 0.13333333
[1] 0.03333333

# Growth rate
for (i in 1:S){
  print((sp[i]*g0[i])/(sp[i]-g1))
}

[1] 0.04
[1] 0.08
[1] 0.03
```

i Python code

```
# Profit rate
for i in range(S):
    print(g0[i] / (sp[i] - g1))

# Growth rate
for i in range(S):
    print((sp[i] * g0[i]) / (sp[i] - g1))
```

References

8 A Post-Kaleckian Distribution and Growth Model

8.1 Overview

The post-Kaleckian growth model was developed by Bhaduri and Marglin (1990) and others to synthesise Marxian and Keynesian ideas about the effects of income distribution on economic growth.¹ According to the Marxian view, capital accumulation is driven by profits. By contrast, Michal Kalecki and post-Keynesians such as Nicholas Kaldor argued that a redistribution of income towards profit-earners is likely to reduce consumption, as workers tend to have a higher marginal propensity to consume than capital owners. The post-Kaleckian growth model integrates these two mechanisms in a Keynesian framework in which aggregate demand and growth are demand-determined. It allows for wage-led as well as profit-led demand and growth regimes. In a wage-led regime, a redistribution of income towards workers has expansionary effects on aggregate demand (and possibly growth) as the expansionary effect on consumption outweighs the negative effect on investment. In a profit-led regime, the effect is contradictory as investment falls by more than consumption rises. Whether a regime is wage- or profit-led depends on the relative size of the propensities to consume and the propensity to invest.

This is a model of long-run steady state growth. In the steady state, all endogenous variables grow at the same rate.² Changes in parameters or exogenous variables lead to an instantaneous adjustment of the model's variables, so that the model can be analysed like a static one. We consider a version of the model with linear functions based on Hein (2014), chap. 7.2.2.

8.2 The Model

$$r = h \frac{u}{v} \quad (8.1)$$

$$s = [s_W + (s_{\Pi} - s_W)h] \frac{u}{v}, \quad 0 \geq s_W > s_{\Pi} \geq 1 \quad (8.2)$$

¹See Hein (2014), chap. 7 and Lavoie (2014), chap. 6 for detailed treatments.

²All variables are normalised by the capital stock and thus rendered stationary.

$$c = \frac{u}{v} - s \quad (8.3)$$

$$g = g_0 + g_1 u + g_2 h, \quad g_i > 0 \quad (8.4)$$

$$u = v(c + g) \quad (8.5)$$

where r , s , c , g , and u are the profit rate, the saving rate, the consumption rate, the investment rate, and the rate of capacity utilisation, respectively.

Equation 8.1 decomposes the profit rate into the product of the profit share h (total profits over total output), the rate of capacity utilisation (actual output over potential output), and the inverse of v (the capital-potential output ratio). Let Y be output, K be the capital stock, and Y^P be potential output, the decomposition can also be written as $r = \frac{\Pi}{K} = \frac{\Pi}{Y} \frac{Y}{Y^P} \frac{Y^P}{K}$. The profit share and the capital-potential output ratio are taken to be exogenous in this model. Note also that the wage share is given by $1 - h$. By Equation 8.2, the economy-wide saving rate is given by the sum of saving out of wages ($s_W(1 - h)\frac{u}{v}$) and saving out of profits ($s_{\Pi}h\frac{u}{v}$). It is assumed that workers have a higher marginal propensity to consume than capital owners ($s_W > s_{\Pi}$). Equation 8.3 simply states that consumption is income not saved. According to Equation 8.3, investment is determined by an autonomous component g_0 that may capture Keynesian ‘animal spirits’, by the rate of capacity utilisation, and by the profit share. While the rate of capacity utilisation is a signal of (future) demand, the profit share may stimulate investment as internal funds are typically the cheapest source of finance. Finally, Equation 8.5 is an equilibrium condition that assumes that the rate of capacity utilisation adjusts to clear the goods market.

The key question addressed by this model is how a change in the profit share affects the rate of capacity utilisation and the rate of growth.³ As shown formally in the analytical discussion below, there is no unambiguous answer to this question as the model encompasses different regimes. Three main regimes can be identified. First, a regime in which both the rate of utilisation and the rate of growth are negatively affected by an increase in the profit share. We will call this a wage-led demand and growth regime (WLD/WLG). Second, a regime in which the rate of utilisation is negatively affected and the rate of growth is positively affected by an increase in the profit share. We will call this a wage-led demand regime and profit-led growth regime (WLD/PLG). Third, a regime in which both the rate of utilisation and the rate of growth are positively affected by an increase in the profit share. We will call this a profit-led demand and growth regime (PLD/PLG).

³Bhaduri and Marglin (1990) further discuss the effects on the profit rate.

8.3 Simulation

8.3.1 Parameterisation

Table 1 reports the parameterisation used in the simulation. We will consider three different parameterisations that represent the three regimes outlined above. For each of these regimes, there is a baseline scenario and a scenario in which the profit share (h) rises. This allows to assess the effects on the model's endogenous variables for the different regimes.

Table 1: Parameterisation

Scenario	v	s_W	s_{II}	g_0	g_1	g_2	h
1a: baseline WLD/WLG	3	0.3	0.9	0.02	0.1	0.1	0.2
1b: rise in profit share (h)	3	0.3	0.9	0.02	0.1	0.1	0.3
2a: baseline WLD/PLG	3	0.3	0.9	0.02	0.08	0.1	0.2
2b: rise in profit share (h)	3	0.3	0.9	0.02	0.08	0.1	0.3
3a: baseline PLD/PLG	3	0.3	0.9	-0.01	0.1	0.1	0.2
3b: rise in profit share (h)	3	0.3	0.9	-0.01	0.1	0.1	0.3

8.3.2 Simulation code

```
#Clear the environment
rm(list=ls(all=TRUE))

# Set number of scenarios (including baselines)
S=6

#Create vector in which equilibrium solutions from different parameterisations will be stored
u_star=vector(length=S) # utilisation rate
g_star=vector(length=S) # growth rate of capital stock
s_star=vector(length=S) # saving rate
c_star=vector(length=S) # consumption rate
r_star=vector(length=S) # profit rate

# Set exogenous variables whose parameterisation changes across regimes
g0=vector(length=S) # animal spirits
sw=vector(length=S) # propensity to save out of wages
h=vector(length=S) # profit share
g1=vector(length=S) # sensitivity of investment with respect to utilisation

### Construct different scenarios across 3 regimes: (1) WLD/WLG, (2) WLD/PLG, (3) PLD/PLG
```

```

# baseline WLD/WLG
g0[1]=0.02
g1[1]=0.1
h[1]=0.2

# increase in profit share in WLD/WLG regime
g0[2]=0.02
g1[2]=0.1
h[2]=0.3

# baseline WLD/PLG
g0[3]=0.02
g1[3]=0.08
h[3]=0.2

# increase in profit share in WLD/PLG regime
g0[4]=0.02
g1[4]=0.08
h[4]=0.3

# baseline PLD/PLG
g0[5]=-0.01
g1[5]=0.1
h[5]=0.2

# increase in profit share in PLD/PLG regime
g0[6]=-0.01
g1[6]=0.1
h[6]=0.3

#Set constant parameter values
v=3      # capital-to-potential output ratio
g2=0.1 # sensitivity of investment with respect to profit share
sp=0.9 # propensity to save out of profits
sw=0.3 # propensity to save out of wages

#Check Keynesian stability condition for all scenarios
for (i in 1:S){
print(((sw+(sp-sw)*h[i])*(1/v) -g1[i])>0)
}

```

```

[1] TRUE

# Check demand and growth regime for 3 baseline scenarios
for (i in c(1,3,5)){
  print(paste("Parameterisation", i, "yields:"))
  if(g2*(sw/v - g1[i])-g0[i]*(sp-sw)/v<0){
    print("wage-led demand regime")
  } else{
    print("profit-led demand regime")
  }
  if(g1[i]*(g2*(sw/v - g1[i])-g0[i]*(sp-sw)/v)+g2*((sw+(sp-sw)*h[i])*v^(-1)-g1[i])^2<0){
    print("wage-led growth regime")
  } else{
    print("profit-led growth regime")
  }
}

[1] "Parameterisation 1 yields:"
[1] "wage-led demand regime"
[1] "wage-led growth regime"
[1] "Parameterisation 3 yields:"
[1] "wage-led demand regime"
[1] "profit-led growth regime"
[1] "Parameterisation 5 yields:"
[1] "profit-led demand regime"
[1] "profit-led growth regime"

# Initialise endogenous variables at some arbitrary positive value
g=1
r=1
c=1
u=1
s=1

#Solve this system numerically through 1000 iterations based on the initialisation
for (i in 1:S){

```

```

for (iterations in 1:1000){

  #(1) Profit rate
  r = (h[i]*u)/v

  #(2) Saving
  s = (sw+(sp-sw)*h[i])*(u/v)

  #(3) Consumption
  c= u/v-s

  #(4) Investment
  g = g0[i]+g1[i]*u+g2*h[i]

  #(5) Rate of capacity utilisation
  u = v*(c+g)
}

#Save results for different parameterisations in vector
u_star[i]=u
g_star[i]=g
r_star[i]=r
s_star[i]=s
c_star[i]=c
}

```

 Python code

```

import numpy as np

# Set number of scenarios (including baselines)
S = 6

# Create arrays to store equilibrium solutions for different parameterizations
u_star = np.zeros(S)
g_star = np.zeros(S)
s_star = np.zeros(S)
c_star = np.zeros(S)
r_star = np.zeros(S)

# Set exogenous variables whose parameterization changes across regimes
g0 = np.zeros(S)
sw = np.zeros(S)
h = np.zeros(S)
g1 = np.zeros(S)

# Construct different scenarios across 3 regimes
# Regime 1: WLD/WLG
g0[0] = 0.02
g1[0] = 0.1
h[0] = 0.2

# Regime 2: Increase in profit share in WLD/WLG regime
g0[1] = 0.02
g1[1] = 0.1
h[1] = 0.3
154

# Regime 3: WLD/PLG
g0[2] = 0.02
g1[2] = 0.08
h[2] = 0.2

```

8.3.3 Plots

Figures Figure 8.1 - Figure 8.4 depict the response of the model's key endogenous variables to changes in the profit share. In the first case of a wage-led demand and growth regime (WLD/WLG), investment is equally sensitive to a change in the rate of capacity utilisation (g_1) and a change in the profit share (g_2). A rise in the profit share reduces consumption, which reduces the rate of capacity utilisation and the rate of growth. This is despite a positive effect on the profit rate.⁴

```
barplot(u_star, ylab="u", names.arg=c("1a:baseline WLD/WLG", "1b:rise prof share", "2a:base
```

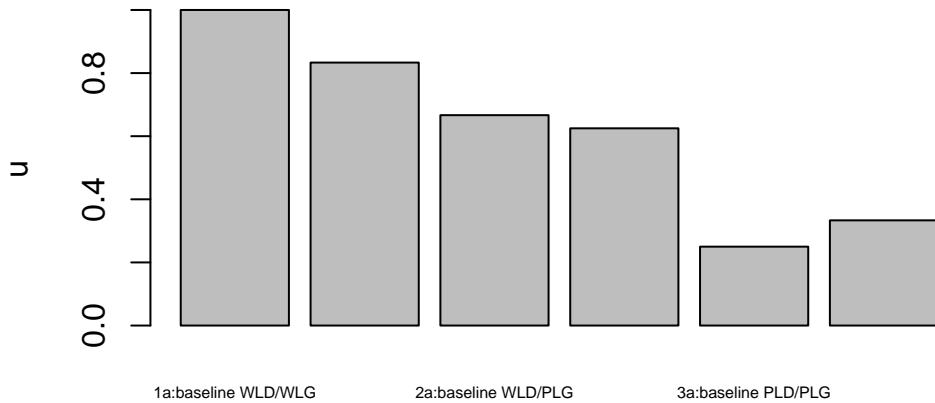


Figure 8.1: Rate of capacity utilisation

In the second case where the demand regime is wage-led but the growth regime is profit-led (WLD/PLG), investment is slightly less sensitive to a change in the rate of capacity utilisation compared to a change in the profit share. The rise in the profit share reduces consumption and the rate of utilisation, but the ultimate effect on investment is positive because investment reacts more strongly to the rise in the profit share than to the fall in demand.

⁴If the negative effect on the rate of capacity utilisation was stronger, the profit rate could fall as well. See the analytical discussion for a formal derivation of the condition under which this may happen.

```
barplot(g_star, ylab="g", names.arg=c("1:baseline WLD/WLG", "2:rise prof share",
                                      "3:baseline WLD/PLG", "4:rise prof share",
                                      "5:baseline PLD/PLG", "6: rise prof share"), cex.names=1)
```

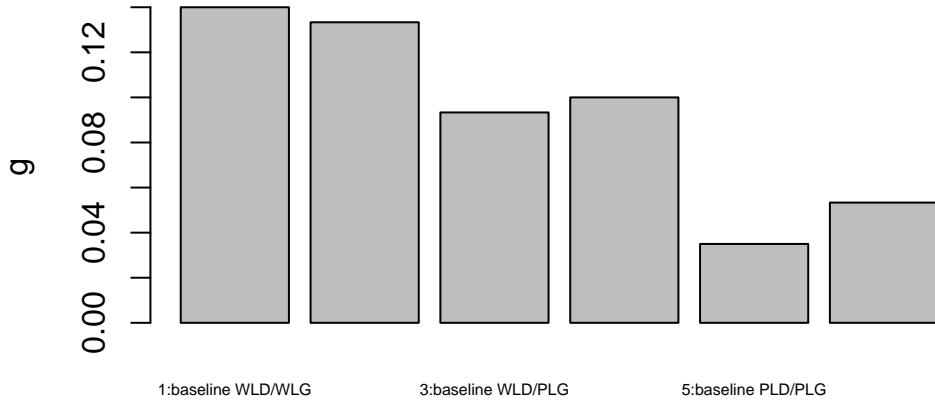


Figure 8.2: Rate of growth

Finally, in the third case where the demand regime and the growth regime are profit-led, investment is again equally sensitive to a change in the rate of capacity utilisation and to a change in the profit share, but now animal spirits are negative. A rise in the profit share now has strong positive effects on investment, which raises the rate of capacity utilisation and consumption.

```
barplot(c_star, ylab="c", names.arg=c("1:baseline WLD/WLG", "2:rise prof share", "3:baseline WLD/PLG",
                                      "4:rise prof share", "5:baseline PLD/PLG", "6: rise prof share"), cex.names=1)
```

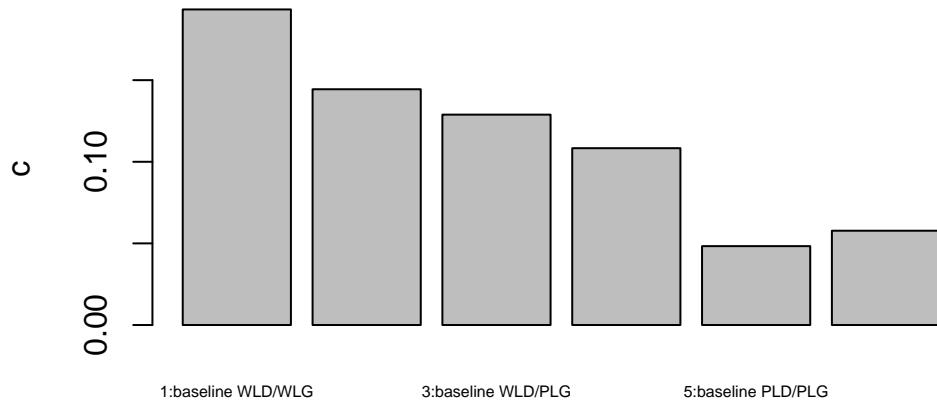


Figure 8.3: Rate of consumption

```
barplot(r_star, ylab="r", names.arg=c("1:baseline WLD/WLG", "2:rise prof share", "3:baseline WLD/PLG", "4:rise prof share", "5:baseline PLD/PLG", "6: rise prof share"), cex.names=1.5)
```

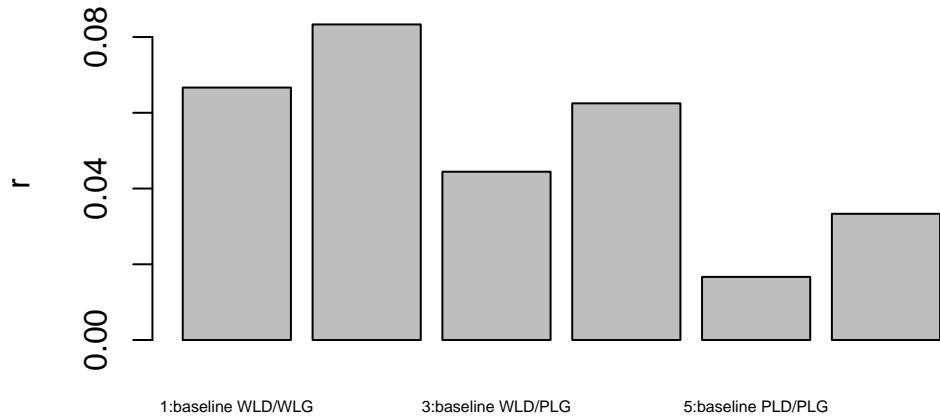


Figure 8.4: Rate of profit

i Python code

```
# Plot results (here only for rate of capacity utilisation)
import matplotlib.pyplot as plt

# Scenario labels
scenario_names = ["1a: baseline WLD/WLG", "1b: rise prof share", "2a: baseline WLD/PLG",
                   "2b: rise prof share"]

# Bar plot for u_star
plt.bar(scenario_names, u_star)
plt.ylabel('u')
plt.xticks(scenario_names, rotation=45, fontsize=6)
plt.show()
```

8.4 Directed graph

Another perspective on the model's properties is provided by its directed graph. A directed graph consists of a set of nodes that represent the variables of the model. Nodes are connected

by directed edges. An edge directed from a node x_1 to node x_2 indicates a causal impact of x_1 on x_2 .

```
## Create directed graph
# Construct auxiliary Jacobian matrix for 6 variables:
# r, h, u, s, c, g

M_mat=matrix(c(0,1,1,0,0,0,
              0,0,0,0,0,0,
              0,0,0,0,1,1,
              0,1,1,0,0,0,
              0,0,1,1,0,0,
              0,1,1,0,0,0), 6, 6, byrow=TRUE)

# Create adjacency matrix from transpose of auxiliary Jacobian
A_mat=t(M_mat)

# Create and plot directed graph from adjacency matrix
library(igraph)
dg= graph_from_adjacency_matrix(A_mat, mode="directed", weighted= NULL)

# Define node labels
V(dg)$name=c("r", "h", "u", "s", "c", "g")

# Plot directed graph
plot(dg, main="", vertex.size=20, vertex.color="lightblue",
      vertex.label.color="black", edge.arrow.size=0.3, edge.width=1.1, edge.size=1.2,
      edge.arrow.width=1.2, edge.color="black", vertex.label.cex=1.2,
      vertex.frame.color="NA", margin=-0.08)
```

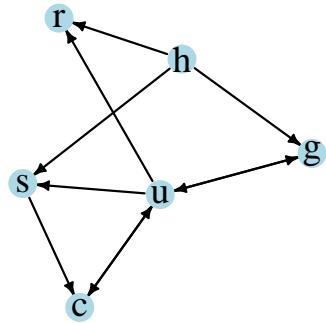


Figure 8.5: Directed graph of post-Kaleckian growth model

 Python code

```
# Load relevant libraries
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

# Define the Jacobian matrix
M_mat = np.array([[0,1,1,0,0,0],
                  [0,0,0,0,0,0],
                  [0,0,0,0,1,1],
                  [0,1,1,0,0,0],
                  [0,0,1,1,0,0],
                  [0,1,1,0,0,0],
                  ])

# Create adjacency matrix from transpose of auxiliary Jacobian and add column names
A_mat = M_mat.transpose()

# Create the graph from the adjacency matrix
G = nx.DiGraph(A_mat)

# Define node labels
nodelabs = {0: "r", 1: "h", 2: "u", 3: "s", 4: "c", 5: "g"}

# Plot the directed graph
pos = nx.spring_layout(G, seed=43)
nx.draw(G, pos, with_labels=True, labels=nodelabs, node_size=300, node_color='lightblue',
        font_size=10)
edge_labels = {(u, v): '' for u, v in G.edges}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='black')
plt.axis('off')
plt.show()
```

In Figure Figure 8.5, it can be seen that the profit share (h) is the key exogenous variable of the model.⁵ Consumption (c), saving (s), investment (g), and the rate of utilisation (u) form a closed loop (or cycle) within the system. The profit share affects both saving and investment, which in turn affect consumption and the rate of capacity utilisation. The profit rate is a

⁵Other important exogenous variables or parameters that may shift but are not depicted here are animal spirits (g_0) or the saving propensities (s_W, s_{Π}). See Hein (2014), chap. 7.2.2, for a detailed discussion of their effects.

residual variable (also called a ‘sink’) in this model.

8.5 Analytical discussion

To find the equilibrium solutions, substitute Equation 8.2 - Equation 8.4 into Equation 8.5 and solve for u :

$$u^* = \frac{g_0 + g_2 h}{[s_W + (s_{\Pi} - s_W)h]v^{-1} - g_1}. \quad (8.6)$$

The equilibrium solution for u can then be substituted into Equation 8.4 to find:

$$g^* = \frac{(g_0 + g_2 h)[s_W + (s_{\Pi} - s_W)h]v^{-1}}{[s_W + (s_{\Pi} - s_W)h]v^{-1} - g_1}. \quad (8.7)$$

The Keynesian stability condition requires $[s_W + (s_{\Pi} - s_W)h]v^{-1} - g_1 > 0$, i.e. saving need to react more strongly to income than investment.

The equilibrium solution for r can be found by substituting u^* into Equation 8.1:

$$r^* = \frac{h(g_0 + g_2 h)}{[s_W + (s_{\Pi} - s_W)h] - vg_1}. \quad (8.8)$$

To assess whether the demand regime is wage- or profit-led, take the derivative of u^* with respect to h :

$$\frac{\partial u^*}{\partial h} = \frac{\frac{s_W}{v}(g_0 + g_2) - (g_0 \frac{s_{\Pi}}{v} + g_1 g_2)}{[[s_W + (s_{\Pi} - s_W)h]v^{-1} - g_1]^2}. \quad (8.9)$$

It can be seen that, e.g., a higher propensity to save out of wages or negative animal spirits make the regime more likely to be profit-led.

By the same token, the sign of the derivative of g^* with respect to h determines whether the growth regime is wage- or profit-led:

$$\frac{\partial g^*}{\partial h} = g_1 \frac{\partial u^*}{\partial h} + g_2. \quad (8.10)$$

It can be seen that, e.g., a higher sensitivity of investment with respect to the profit share makes the regime more likely to be profit-led.

Finally, the effect on the profit rate will depend on the sign of the derivative:

$$\frac{\partial r^*}{\partial h} = \frac{u^*}{v} + \frac{h}{v} \frac{\partial u^*}{\partial h}, \quad (8.11)$$

which is likely to be positive but can become negative if the demand regime is strongly wage-led.

8.5.1 Calculate analytical solutions numerically

```
# Utilisation rate
for (i in 1:S){
  print((g0[i]+g2*h[i])/((sw+(sp-sw)*h[i])/v-g1[i]))
}

[1] 1
[1] 0.8333333
[1] 0.6666667
[1] 0.625
[1] 0.25
[1] 0.3333333

# Growth rate
for (i in 1:S){
  print(((g0[i]+g2*h[i])*(sw+(sp-sw)*h[i])/v)/((sw+(sp-sw)*h[i])/v-g1[i]))
}

[1] 0.14
[1] 0.1333333
[1] 0.09333333
[1] 0.1
[1] 0.035
[1] 0.05333333

# Profit rate
for (i in 1:S){
  print((g0[i]+g2*h[i])*(h[i]/v)/((sw+(sp-sw)*h[i])/v-g1[i]))
}

[1] 0.06666667
[1] 0.08333333
[1] 0.04444444
[1] 0.0625
[1] 0.01666667
[1] 0.03333333
```

Python code

```
# Utilisation rate
for i in range(S):
    print((g0[i]+g2*h[i])/((sw+(sp-sw)*h[i])/v-g1[i]))

# Growth rate
for i in range(S):
    print(((g0[i]+g2*h[i])*(sw+(sp-sw)*h[i])/v)/((sw+(sp-sw)*h[i])/v-g1[i]))

# Profit rate
for i in range(S):
    print((g0[i]+g2*h[i])*(h[i]/v)/((sw+(sp-sw)*h[i])/v-g1[i]))
```

References

Part II

Dynamic Models

9 An Introduction to the Analysis of Dynamic Models

To build and analyse dynamic models, we need to understand the dynamics of *state variables*, which are a function of their own previous values: $y_t = h(y_{t-1})$. The state variables govern the dynamics of the entire model (including the non-state variables). Typically, as model has multiple state variables that interact with each other over time. As briefly shown in Chapter 2, we can simulate dynamic models numerically for a specific parameterisation. However, to study their dynamics in general, we need to mathematically analyse a *system of difference (or differential) equations*. This chapter provides a basic introduction to the mathematical tools to do this. It will help you understand the analytical discussions in the chapters on dynamic models but can be skipped if you are mostly interested in numerical simulation.

9.1 Solution of a single first-order linear difference equation

Consider a first-order linear difference equation:¹

$$y_t = a_0 + a_1 y_{t-1}.$$

One way to find a solution is through (manual) iteration:

$$\begin{aligned} & y_0, \\ & y_1 = a_0 + a_1 y_0, \\ & y_2 = a_0 + a_1(a_0 + a_1 y_0) = a_0(a_1 + 1) + a_1^2 y_0, \\ & y_3 = a_0 + a_1[a_0 + a_1(a_0 + a_1 y_0)] = a_0(a_1^2 + a_1 + 1) + a_1^3 y_0 \\ & \dots, \end{aligned}$$

¹We will focus here on difference instead of differential equations, i.e. on dynamics in discrete as opposed to continuous time. Most of the continuous-time counterpart is analogous to the material covered here. Sayama (2015) provides a very accessible and applied introduction to dynamic systems with Python code. An introductory treatment of the underlying mathematics is Chiang and Wainwright (2005), chaps. 15-19. Gandolfo (2009) provides a more advanced treatment of the mathematics as well as many economic examples. A great introduction to linear algebra is Anthony and Harvey (2012).

$$y_t = a_0 \sum_{i=0}^{t-1} a_1^i + a_1^t y_0$$

Of course, this is effectively the same approach we have used before to solve economic models via simulation.

If $a_1 \neq 1$, the term $a_0 \sum_{i=0}^{t-1} a_1^i$ is a convergent geometric series:

$$a_0 \sum_{i=0}^{t-1} a_1^i = a_0 \frac{(1 - a_1^t)}{1 - a_1}.$$

Thus, the solution thus takes the form:

$$y_t = \frac{a_0(1 - a_1^t)}{1 - a_1} + a_1^t y_0 = \frac{a_0}{1 - a_1} + a_1^t \left(y_0 - \frac{a_0}{1 - a_1} \right).$$

From iteration, we thus know that the solution to a difference equation has two parts:

1. a term that captures the long-run equilibrium y^* (the so-called *particular solution*),
2. a term that captures the dynamics of y_t (the so-called *complementary function*).

$$y_t = \underbrace{\frac{a_0}{1 - a_1}}_{\text{equilibrium } y^*} + \underbrace{a_1^t \left(y_0 - \frac{a_0}{1 - a_1} \right)}_{\text{dynamics}}.$$

The complementary function tells us about the ‘asymptotic stability’ of the equation: does y_t converge to y^* as $t \rightarrow \infty$?

For the case of a first-order difference equation, we can distinguish the following cases:

- if $|a_1| < 1$, then the complementary function will converge to zero and y_t will approach the particular solution y^*
- if $|a_1| > 1$, then the complementary function will grow exponentially or decay, and y_t will thus never converge to the particular solution y^*
- if $a_1 = 1$ and $a_0 \neq 0$, then y_t will grow linearly
- if $a_1 = 1$ and $a_0 = 0$, then y_t will not grow or fall forever, but it will also not approach a unique equilibrium

To better understand the last two cases, note that if $a_1 = 1$, a different (more general) approach to finding the particular solution is required: the so-called *method of undetermined coefficients*. This method consists of substituting a trial solution that contains undetermined coefficients into the difference equation and then attempting to solve for those coefficients. If the trial solution allows to pin down unique values for the coefficients, it constitute a valid particular solution.

In the case above where $a_1 \neq 1$, we could have used the trial solution $y_t = y_{t-1} = y^*$ and then solve for y to obtain $\frac{a_0}{1-a}$ as the particular solution. In the case where $a_1 = 1$ and $a_0 \neq 0$, we can use the trial solution $y^* = kt$, which is a growing equilibrium. This yields $y_t = k(t-1) + a_0$, which solves for $k = a_0$, so that we can conclude $y^* = a_0 t$. This explains why we obtain linear growth. If $a_1 = 1$ and $a_0 = 0$, we have $y_t = y_{t-1}$, so that the equilibrium is given by the initial condition y_0 .

9.2 Solution of a linear system of difference equations

The solution approach just introduced can be extended to N -dimensional systems of linear difference equations of the form:

$$y_t = a_0 + A y_{t-1},$$

where y_t is a $1 \times N$ column vector and A an $N \times N$ square matrix.

If the inverse $(I - A)^{-1}$ exists, which requires $\det(I - A) \neq 0$, the solution will be of the form:

$$y_t = (I - A)^{-1} a_0 + A^t [y_0 - (I - A)^{-1} a_0].$$

The problem with this generic solution is that it is difficult to assess what is going on: the dynamics of any variable in y_t will depend on a lengthy combination of the parameters in A that result from repeated matrix multiplication ($A^t = A \times A \times A \times A \dots$). This makes it impossible to assess whether the system converges to the particular solution. To address this problem, we can use a tool from linear algebra called *matrix diagonalisation*. Under certain conditions, a matrix A can be decomposed into the product of three matrices in which the matrix in the middle is diagonal. As we will see, this trick has a useful application to our problem.

A matrix A is diagonalisable if there is a diagonal matrix D and an invertible matrix P such that $A = PDP^{-1}$. A major advantage of this decomposition is the following property: $A^n = (PDP^{-1})^n = PD^n P^{-1}$.² Thus, the n th power of the matrix A , which typically yields

²This is because in the product $(PDP^{-1})(PDP^{-1})(PDP^{-1})\dots$, each P cancels a P^{-1} , except for the first P and last P^{-1} .

very cumbersome expressions, simplifies to $PD^n P^{-1}$, where the n th power of D is simply applied to each individual element on the main diagonal thanks to D being a diagonal matrix. As a result, diagonalisation allows us to write the complementary function in the solution to a system of difference equations as: $PD^t P^{-1}y_0$. We can further define a vector of arbitrary constants $c = P^{-1}y_0$ so that the complementary function becomes $PD^t c$. For the first variable in the system, the solution would then take the form:

$$y_{1t} = v_{11}c_1\lambda_1^t + v_{12}c_2\lambda_2^t + \dots + y_1^*,$$

where v_j are the column vectors of P and λ_i are the elements on the main diagonal of D . The v_j are called the *eigenvectors* of the matrix A and the λ_i are its *eigenvalues* (more about them in a second). From this representation of the solution, the nature of the dynamics can easily be determined by looking at the eigenvalue λ that is largest in absolute terms. This is also called the ‘dominant eigenvalue’. Only if the dominant eigenvalue is $|\lambda| < 1$ will the system converge to y^* . The elements v_{ij} of the eigenvectors act as multipliers on the eigenvalues and can thus switch off certain eigenvalues (if they happen to be zero) or amplify their dynamics both into the positive and negative domain (depending on their algebraic sign).

How can the diagonal matrix D be found? Notice that $AP = PD$ can also be written as $Av = \lambda v$. We can then write $v(A - \lambda I) = 0$. We want to find the solutions of this linear system other than $v = 0$ (we don’t want the eigenvectors to be zero vectors, otherwise the solution to the dynamic system presented above wouldn’t work). This requires the determinant of the matrix $A - \lambda I$ to become zero, i.e. $\det(A - \lambda I) = 0$. Note that then there will be an infinite number of solutions for the eigenvectors.

Let’s consider an example. Let $a_0 = 0$ for simplicity, so that the dynamic system is $y_t = Ay_{t-1}$. Let the matrix A be given by:

$$A = \begin{bmatrix} 7 & -15 \\ 2 & -4 \end{bmatrix}.$$

Then

$$A - \lambda I = \begin{bmatrix} 7 - \lambda & -15 \\ 2 & -4 - \lambda \end{bmatrix}$$

and

$$\det(A - \lambda I) = (7 - \lambda)(-4 - \lambda) + 30 = \lambda^2 - 3\lambda + 2 = 0.$$

This second-order polynomial solves for $\lambda_1 = 2$ and $\lambda_2 = 1$, which will be the elements on the diagonal of D .

To find v_j , substitute the λ_i into $v_j(A - \lambda_i I) = 0$. For $\lambda_1 = 2$, we get $5v_{11} - 15v_{21} = 0$ and $2v_{11} - 6v_{21} = 0$, yielding the eigenvector $v_1 = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$. However, any scalar multiple of this eigenvector (other than zero) is admissible. It is thus common to normalise the eigenvectors by dividing through one of its elements. Dividing through by the first element yields the normalised eigenvector $v_1 = \begin{bmatrix} 1 \\ \frac{1}{3} \end{bmatrix}$.

For $\lambda_2 = 1$, this yields $6v_{12} - 15v_{22} = 0$ and $2v_{12} - 5v_{22} = 0$ from which we can deduce that $v_2 = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$. The normalised eigenvector is $v_2 = \begin{bmatrix} 1 \\ 0.4 \end{bmatrix}$.

Of course, you can also perform these calculations in *R* or *Python*:

```
#Clear the environment
rm(list=ls(all=TRUE))

## Find eigenvalues and eigenvectors of matrix
# Define matrix
J=matrix(c(7, -15,
          2, -4), 2, 2, byrow=TRUE)

# Obtain eigenvalues and eigenvectors
ev=eigen(J)
(evals = ev$values)

[1] 2 1

(evecs = ev$vector)

[,1]      [,2]
[1,] 0.9486833 0.9284767
[2,] 0.3162278 0.3713907

# Normalise eigenvectors by dividing through by the first element
evecs_norm=evecs
for (i in 1:2){
  evecs_norm[,i]=evecs[,i]/evecs[1,i]
}
evecs_norm
```

```
[,1] [,2]
[1,] 1.0000000 1.0
[2,] 0.3333333 0.4
```

 Python code

```
import numpy as np

# Define matrix
J = np.array([[7, -15],
              [2, -4]])

# Obtain eigenvalues and eigenvectors
evals, evecs = np.linalg.eig(J)

# Print eigenvalues and eigenvectors
print(evals)
print(evecs)

# Initialize an array to store the normalized eigenvectors
evecs_norm = np.copy(evecs)

# Normalize the eigenvectors
for i in range(2):
    evecs_norm[:, i] = evecs[:, i] / evecs[0, i]

# Print normalized eigenvectors
print(evecs_norm)
```

We can now use this solution for the eigenvectors and eigenvalues to write the solution of the dynamic system as:

$$\begin{bmatrix} y_{1t} \\ y_{2t} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \frac{1}{3} & 0.4 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}^t \begin{bmatrix} 1 & 1 \\ \frac{1}{3} & 0.4 \end{bmatrix}^{-1} \begin{bmatrix} y_{10} \\ y_{20} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \frac{1}{3} & 0.4 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}^t \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}.$$

Multiplying the matrices out yields:

$$y_{1t} = c_1 2^t + c_2 1^t$$

$$y_{2t} = \frac{1}{3}c_1 2^t + 0.4c_2 1^t.$$

Before comparing these analytical results with those from a numerical simulation, let's summarise the information we gain from the eigenvalues, eigenvectors, and arbitrary constants about the dynamics of the system:

- since the dominant eigenvalue $\lambda_1 = 2$ is larger than one, we know that the system is unstable
- since both elements in the dominant eigenvector $v_1 = \begin{bmatrix} 1 \\ \frac{1}{3} \end{bmatrix}$ are non-zero, both variables in the system will be driven by that dominant eigenvalue
- since both elements in the dominant eigenvector are positive but the arbitrary constant c_1 is negative for positive initial conditions (see below), both variables will decay
- since both variables will decay at the same rate, their ratio will be constant as $t \rightarrow \infty$ and will approach a value that is given by the ratio of the elements in the dominant eigenvector

To see the last point, observe that in $\frac{y_{2t}}{y_{1t}} = \frac{\frac{1}{3}c_1 2^t + 0.4c_2 1^t}{c_1 2^t + c_2 1^t}$ the first terms in the numerator and denominator, respectively, quickly dominate the second terms as $t \rightarrow \infty$ (you can show this formally using L'Hopital's rule). Thus, $\frac{y_{2t}}{y_{1t}}$ will approach $\frac{1}{3}$ as $t \rightarrow \infty$.

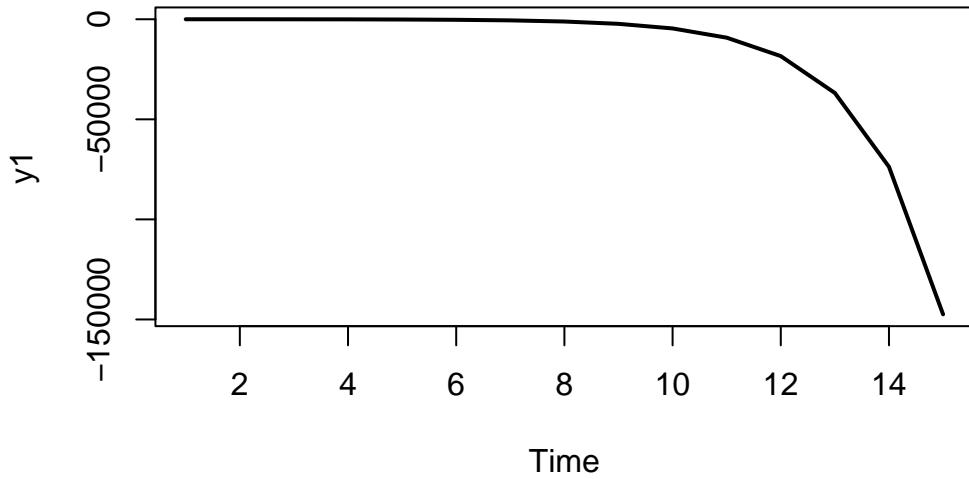
Let us simulate the system and compare the results for, say, $t = 10$ with the analytical solution:

```
# Set number of periods for which you want to simulate
Q=100

# Construct matrices in which values for different periods will be stored; initialise at 1
y1=matrix(data=1, nrow=1, ncol=Q)
y2=matrix(data=1, nrow=1, ncol=Q)

#Solve this system recursively based on the initialisation
for (t in 2:Q){
  y1[,t] = J[1,1]*y1[, t-1] + J[1,2]*y2[, t-1]
  y2[,t] = J[2,1]*y1[, t-1] + J[2,2]*y2[, t-1]
} # close time loop

# Plot dynamics of y1
plot(y1[1, 1:15], type="l", col=1, lwd=2, lty=1, xlab="Time", ylab="y1")
title(main="", cex=0.8)
```



```

# Find arbitrary constants: c=(P^-1)*y0
library(matlib)
y0=c(y1[1,1],y2[1,1]) # create vector with initial conditions y0
c=inv(evecs_norm)%*%y0
c

[,1]
[1,] -9
[2,] 10

## Compute solution manually for y2 at t=10 and compare with simulated solution
t=10
evecs_norm[2,1]*c[1,1]*evals[1]^t + evecs_norm[2,1]*c[2,1]*evals[2]^t # analytical solution

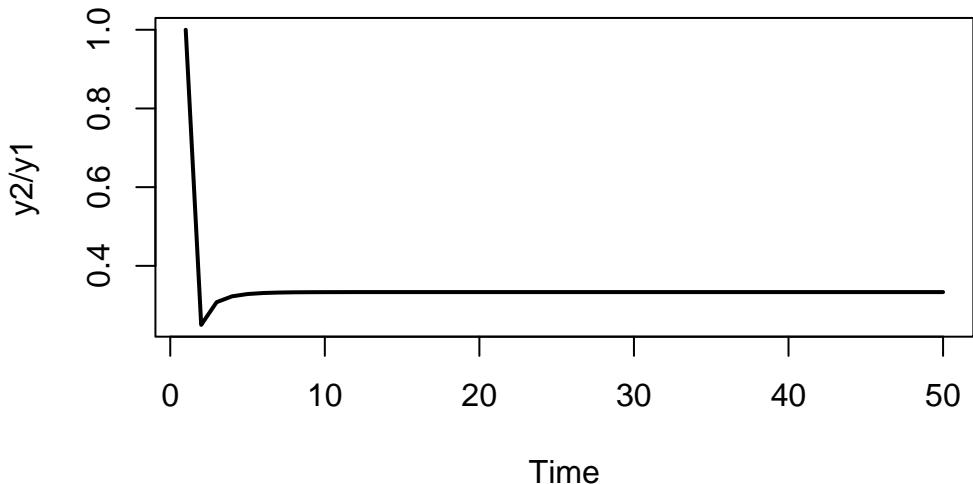
[1] -3068.667

y2[,t+1] # simulated solution

[1] -3068

```

```
# Plot dynamics of y2/y1
y2_y1=y2/y1
plot(y2_y1[, 1:50], type="l", col=1, lwd=2, lty=1, xlab="Time", ylab="y2/y1")
title(main="", cex=0.8)
```



```
# Compare y2/y1 with normalised dominant eigenvector
y2_y1[,Q]
```

```
[1] 0.3333333
```

```
evecs_norm[2,1]
```

```
[1] 0.3333333
```

 Python code

```

import matplotlib.pyplot as plt

# Set the number of periods for simulation
Q = 100

# Initialize arrays to store values for different periods
y1 = np.ones(Q)
y2 = np.ones(Q)

# Solve the system recursively based on the initialization
for t in range(1, Q):
    y1[t] = J[0, 0] * y1[t - 1] + J[0, 1] * y2[t - 1]
    y2[t] = J[1, 0] * y1[t - 1] + J[1, 1] * y2[t - 1]

# Plot dynamics of y1
plt.plot(range(Q), y1, color='b', linewidth=2)
plt.xlabel('Time')
plt.ylabel('y1')
plt.title('Dynamics of y1')
plt.show()

# Define the initial conditions y0
y0 = np.array([y1[0], y2[0]])

# Calculate the arbitrary constants c using the normalized eigenvectors
c = np.linalg.inv(evecs_norm).dot(y0)
c

## Compute solution manually for y2 at t=10 and compare with simulated solution
t = 10 +1
evecs_norm[1, 1] * c[0] * evals[0] ** t + evecs_norm[1, 1] * c[1] * evals[1] ** t
y2[t-1]

# Calculate the ratio y2/y1
y2_y1 = y2 / y1

# Plot dynamics of y2/y1 for the first 50 periods
plt.plot(y2_y1[:50], color='black', linewidth=2, linestyle='--')
plt.xlabel('Time')
plt.ylabel('y2/y1')
plt.show()

# Compare y2/y1 with normalised dominant eigenvector
y2_y1[Q-1]
evecs_norm[1,0]

```

It can be seen that the simulated results are equivalent to the results we obtained analytically. The key takeaway is that by deriving information about the eigenvalues (and possibly eigenvectors) of the Jacobian matrix of the system, we are able to deduce knowledge of the dynamic properties of the system even without numerical simulation. However, the more complex the dynamic system, the more difficult this will be, thereby rendering numerical simulation a key tool to supplement formal analysis.

9.3 Complex eigenvalues and cycles

So far, we have discussed the case where the eigenvalues λ are real numbers. However, what if the polynomial $\det(A - \lambda I) = 0$ does not yield real numbers? Recall that in the case of a second-order polynomial $\lambda^2 + b\lambda + c = 0$, the two roots are given by $\lambda_{1,2} = \frac{-b \pm \sqrt{b^2 - 4c}}{2}$. If the term under the root $\Delta = b^2 - 4c$, also called discriminant, becomes negative, the solution will be a *complex number*. More specifically, we can write:

$$\lambda_{1,2} = \frac{-b \pm \sqrt{b^2 - 4c}}{2} = \frac{-b \pm \sqrt{4c - b^2}\sqrt{-1}}{2} = \frac{-b \pm \sqrt{4c - b^2}i}{2},$$

where $i = \sqrt{-1}$ is the imaginary number. The expression can also be written as:

$$\lambda_{1,2} = h \pm mi,$$

which is a pair of conjugate complex numbers containing a real part given by h and an imaginary part given by m .

Consider the model by Samuelson (1939) discussed in Chapter 2:

$$C_t = c_1(C_{t-1} + I_{t-1} + G_0)$$

$$I_t = \beta[c_1(C_{t-1} + I_{t-1} + G_0) - C_{t-1}]$$

The Jacobian matrix of this model is given by:

$$J = \begin{bmatrix} c_1 & c_1 \\ \beta(c_1 - 1) & \beta c_1 \end{bmatrix}$$

The characteristic polynomial yielding the eigenvalues of the Jacobian is

$$\lambda^2 - \lambda c_1(1 + \beta) + \beta c_1 = 0.$$

Thus we have

$$\lambda_{1,2} = \frac{c_1(1 + \beta) \pm \sqrt{[c_1(1 + \beta)]^2 - 4\beta c_1}}{2}.$$

Thus, the two eigenvalues will be a pair of complex conjugates if $[c_1(1 + \beta)]^2 + 4\beta c_1 < 0$. Suppose we have $c_1 = 0.4$ and $\beta = 2$. Then the discriminant will be negative and the eigenvalues will be complex:

```
#Clear the environment
rm(list=ls(all=TRUE))

# Set parameter values
c1=0.4
beta=2

# Check if discriminant is negative
(c1*(1+beta))^2-4*c1*beta

[1] -1.76

## Find eigenvalues and eigenvectors of matrix
# Define matrix
J=matrix(c(c1, c1,
           beta*(c1-1), beta*c1),
          2, 2, byrow=TRUE)

# Obtain eigenvalues and eigenvectors
ev=eigen(J)
(evals = ev$values)

[1] 0.6+0.663325i 0.6-0.663325i
```

Python code

```
# Set parameter values
c1 = 0.4
beta = 2

# Check if discriminant is negative
(c1 * (1 + beta))**2 - 4 * c1 * beta

# Define the matrix
J = np.array([[c1, c1],
              [beta * (c1 - 1), beta * c1]])

# Calculate eigenvalues and eigenvectors
evals, evecs = np.linalg.eig(J)

print(evals)
print(evals)
```

Another way of understanding the logic behind complex numbers is through a so-called *Argand diagram* that plots the real part of the eigenvalue on the horizontal and the imaginary part on the vertical axis. By Pythagoras' theorem, the distance of the eigenvalue from the origin will then be given by $R = \sqrt{h^2 + m^2}$. The value of R (which is always real-valued and positive) is called the *modulus* (or absolute value) of the complex eigenvalue and will contain important information about the dynamic stability of economic models that exhibit complex eigenvalues.

```
### Draw Argand diagram

# Save real and imaginary part of complex eigenvalue
re=Re(evals[1])
im=Im(evals[1])

# Plot complex eigenvalue
par(bty="l")
plot(re,im, type="o", xlim=c(0, 1), ylim=c(0, 1), lwd=2, xlab="h", ylab="m", main="Argand

# Plot unit circle
X=seq(0, 1, by=0.001)
Y = sqrt(1 - X^2)
```

```

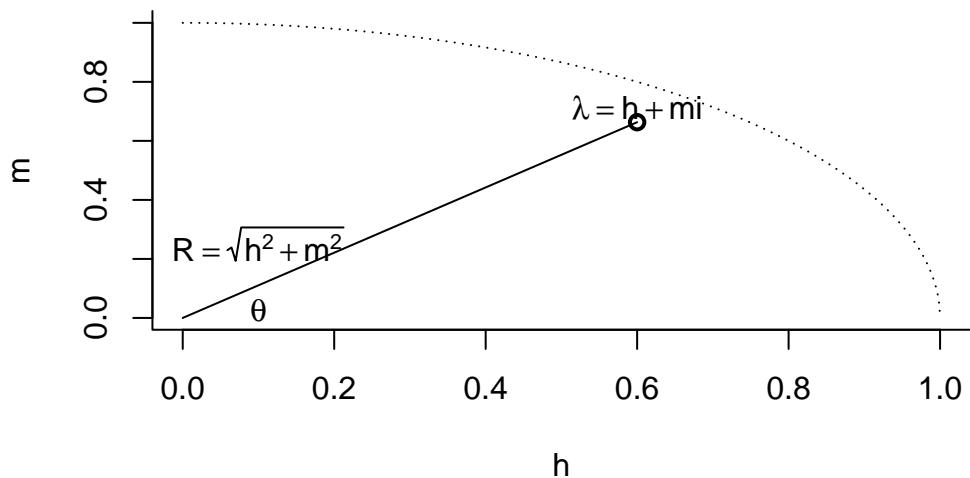
lines(X,Y, type="l", lty="dotted")

# Plot a ray from the origin to eigenvalue
segments(0,0,re,im, lty='solid')

# Add labels
text(0.1, 0.025, expression(theta), cex=1)
text(0.1, 0.25, expression(R==sqrt(h^2+m^2)), cex=1)
text(re, im+0.05, expression(lambda==h+mi), cex=1)

```

Argand diagram of complex eigenvalue



Python code

```
### Draw Argand diagram

# Save real and imaginary part of complex eigenvalue
re = evals[0].real
im = evals[0].imag

# Create a figure
fig, ax = plt.subplots()
ax.set_xlim(0, 1)
ax.set_ylimit(0, 1)
ax.set_xlabel('h')
ax.set_ylabel('m')
ax.set_title('Argand diagram of complex eigenvalue')

# Plot complex eigenvalue
ax.plot(re, im, 'o', markersize=8, color='k')

# Plot unit circle
X = np.linspace(0, 1, 100)
Y= np.sqrt(1-X**2)
ax.plot(X, Y, 'k--')

# Plot a ray from the origin to the eigenvalue
ax.plot([0, re], [0, im], 'k-')

# Add labels
ax.text(0.1, 0.025, r'$\theta$', fontsize=12)
ax.text(0.001, 0.25, r'$R=\sqrt{h^2+m^2}$', fontsize=12)
ax.text(re, im - 0.1, r'$\lambda=h+mi$', fontsize=12)

plt.show()
```

The angle θ of the line that connects the origin and the complex eigenvalue and the x-axis of the Argand diagram also contains information about the dynamics. To see this, note that the geometry of the complex number represented in the Argand diagram can also be expressed in trigonometric form:

$$\sin \theta = \frac{m}{R}$$
$$\cos \theta = \frac{h}{R},$$

where $\theta = \arcsin(\frac{m}{R}) = \arccos(\frac{h}{R}) = \arctan(\frac{m}{h})$

Thus, we can write the complex eigenvalue also as:

$$\lambda_{1,2} = R(\cos \theta \pm \sin \theta \times i).$$

By De Moivre's theorem, we have $(\cos \theta \pm \sin \theta \times i)^t = (\cos \theta t \pm \sin \theta t \times i)$. Thus, the solution to a dynamic system that exhibits complex eigenvalues will be of the form:

$$y_{1t} = v_{11}c_1 R_1^t (\cos \theta_1 t \pm \sin \theta_1 t \times i) + \dots + y_1^*.$$

From this solution we can again deduce key information about the dynamics of the system based on the (complex) eigenvalues:

- stability will depend on the modulus: for $R < 1$ the system will be stable, for $R > 1$ it will be unstable
- from the nature of the trigonometric functions $\sin(\theta t)$ and $\cos(\theta t)$, we know that system will exhibit periodic cyclical dynamics as t increases
- the length of the cycles will be given by $L = \frac{2\pi}{\theta}$ and the frequency by $F = 1/L = \frac{\theta}{2\pi}$
- the amplitude of the cycles will depend on the elements of the eigenvectors, the initial conditions, and R .

Let us simulate the Samuelson model with the parameterisation that yields complex eigenvalues to illustrate these results:

```
# Calculate modulus
mod=Mod(evals[1])
mod
```

```
[1] 0.8944272
```

```
# Calculate cycle length
L=(2*pi)/(acos(re/mod))
L
```

```
[1] 7.520433
```

```
# Set number of periods for which you want to simulate
Q=100
```

```

# Set number of parameterisations that will be considered
S=1

# Construct matrices in which values for different periods will be stored; initialise at 1
C=matrix(data=1, nrow=S, ncol=Q)
I=matrix(data=1, nrow=S, ncol=Q)

#Construct matrices for exogenous variable
G0=matrix(data=5, nrow=S, ncol=Q)

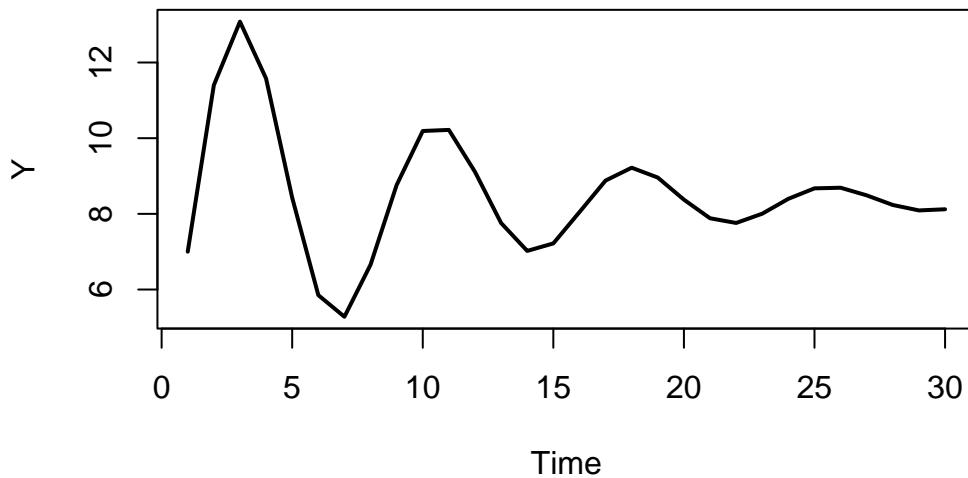
#Solve this system recursively based on the initialisation
for (t in 2:Q){
  C[1,t] = c1*(C[1,t-1] + I[1,t-1] + G0[1,t-1])
  I[1,t] = beta*(c1*(C[1,t-1] + I[1,t-1] + G0[1,t-1]) - C[1,t-1])
} # close time loop

# Calculate output
Y=C+G0+I

# Time series chart of output dynamics in Samuelson (1939) model
plot(Y[1, 1:30], type="l", col=1, lwd=2, lty=1, xlab="Time", ylab="Y")
title(main="Figure 1: Output", cex=0.8)

```

Figure 1: Output



Python code

```
# Calculate modulus
mod = abs(evals[0])
print(mod)

# Calculate cycle length
import math
L = (2 * math.pi) / math.acos(re / mod)
print(L)

# Set the number of periods and parameterizations
Q = 100
S = 1

# Initialize matrices for consumption, investment, and exogenous government spending
C = np.ones((S, Q))
I = np.ones((S, Q))
G0 = np.full((S, Q), 5)

# Solve the system recursively based on the initialization
for t in range(1, Q):
    C[0, t] = c1 * (C[0, t - 1] + I[0, t - 1] + G0[0, t - 1])
    I[0, t] = beta * (c1 * (C[0, t - 1] + I[0, t - 1] + G0[0, t - 1]) - C[0, t - 1])

# Calculate output
Y = C + G0 + I

# Plot the time series chart of output dynamics
plt.plot(Y[0, :30], color='k', linewidth=2, linestyle='--')
plt.xlabel("Time")
plt.ylabel("Y")
plt.title("Figure 1: Output")
plt.show()
```

You can see that the model generates cycles with a length (from peak/trough to peak/trough) of around 7.5 periods. Since the modulus is $R < 1$, the system is stable and eventually converges to the equilibrium.

9.4 Nonlinear systems

So far, we have analysed dynamic systems that are linear. However, in the more general case, a dynamic system may be nonlinear and of the form:

$$y_t = f(y_{t-1}).$$

An n -dimensional nonlinear system may have multiple equilibria y^* . To analyse the dynamic properties of such a system, we normally conduct a linear approximation in the neighbourhood of one of the equilibria. In that sense, the stability analysis of a nonlinear system has only local as opposed to global validity.

Mathematically, linearisation around an equilibrium point can be done by conducting a first-order Taylor expansion around that equilibrium:

$$y_t = f^i(y^*) + \sum_{j=1}^n \frac{\partial f^i(y^*)}{\partial y_{jt-1}} (y_{jt-1} - y_j^*),$$

where $i = 1, 2, \dots, n$.

This yields a linear version of the system that can be written as:

$$y_t = Ay_{t-1} + B,$$

where $A_{11} = \frac{\partial f^1(y^*)}{\partial y_{1t-1}}$ and so forth. Thus, A is simply the Jacobian matrix of $f(y_{t-1})$ evaluated at y^* .

In practice, this means that to analyse the local stability of a nonlinear system, one needs to:

- find the equilibrium solution y^* whose neighbourhood you want to analyse
- compute the Jacobian matrix of $f(y_{t-1})$
- substitute y^* into $f(y_{t-1})$ and analyse the resulting matrix.

An example for the stability analysis of a simple nonlinear system can be found in Chapter 12.

9.5 Key takeaways

- dynamic models are systems of difference (or differential) equations
- the stability of a system depends on (a combination of) its coefficients
- more generally, the system's dynamic properties (including stability) are encapsulated in the Jacobian matrix
- the (dominant) eigenvalues of the Jacobian matrix indicate whether a system is
 - stable ($\lambda < 1$) or unstable ($\lambda > 1$)
 - acyclical ($\lambda \in \mathbb{R}$) or cyclical ($\lambda \in \mathbb{C}$)
- the (dominant) eigenvectors mediate the impact of the eigenvalues in the dynamics
- nonlinear systems are analysed locally around one of its equilibria

9.6 References

10 A New Keynesian 3-Equation Model

10.1 Overview

New Keynesian dynamic general equilibrium models were developed in the 1990s and 2000s to guide monetary policy.¹ They build on real business cycle models with rational expectations but introduce Keynesian frictions such as imperfect competition and nominal rigidities. While the structural forms of these models are typically complex as behavioural functions are derived from the intertemporal optimisation, the reduced-form of the benchmark models can be represented by three main equations: (i) an IS curve, (ii) a Phillips curve, (iii) and an interest rate rule.

The IS curve establishes a negative relationship between real income and the real interest rate. For a higher real interest rate, households will save more and thus consume less. The Phillips curve models inflation as a function of the output gap. A positive output gap (an economic expansion) leads to higher inflation. The monetary policy rule specifies how the central bank reacts to deviations of actual inflation from a politically determined inflation target.

The simplified version of the 3-equation model we consider here is directly taken from chapter 4 of Carlin and Soskice (2014). This is a short-run model in which prices are flexible but the capital stock is fixed. The focus is thus on goods market equilibrium rather than economic growth. In the Carlin-Soskice version, inflation expectations are assumed to be adaptive and the response of aggregate demand to a change in the interest rate is sluggish. This renders the model dynamic.²

10.2 The Model

$$y_t = A - a_1 r_{t-1} \quad (10.1)$$

$$\pi_t = \pi_{t-1} + a_2 (y_t - y_e) \quad (10.2)$$

¹See Galí (2018) for an overview.

²Note that this is quite different from conventional New Keynesian dynamic general equilibrium models in which the dynamic element stems from agents with rational expectations that react to serially correlated shocks.

$$r_s = \frac{(A - y_e)}{a_1} \quad (10.3)$$

$$r_t = r_s + a_3(\pi_t - \pi^T) \quad (10.4)$$

where y , A , r , π , y_e , r_s , and π^T are real output, autonomous demand (times the multiplier), the real interest rate, inflation, equilibrium output, the stabilising real interest rate, and the inflation target, respectively.

Equation 10.1 is the IS curve or goods market equilibrium condition. Aggregate output adjusts to the level of aggregate demand, which is given by autonomous demand (times the multiplier) and a component that is negatively related to the (lagged) real interest rate via households' saving ($a_1 > 0$). Equation 10.2 is the Phillips curve. It is assumed that inflation is driven by adaptive expectations ($E[\pi_{t+1}] = \pi_{t-1}$) and positively related to the output gap ($y_t - y_e$), i.e. $a_2 > 0$. By Equation 10.3, the stabilising real interest rate is that real interest rate that is consistent with equilibrium output ($y_e = A - a_1 r_s$). Finally, the interest rate rule in Equation 10.4 specifies the real interest rate the central bank needs to set to minimise its loss function (see Section 10.5 below for a derivation). The parameter a_3 is a composite one given by $a_3 = \frac{1}{a_1(\frac{1}{a_2 b} + a_2)} > 0$. Although the central bank only sets the nominal interest rate $i = r + E[\pi_{t+1}]$ directly, the fact that expected inflation is predetermined in every period allows it to indirectly control the real interest rate.

10.3 Simulation

10.3.1 Parameterisation

Table 1 reports the parameterisation used in the simulation. For all parameterisations, the system is initialised at the equilibrium $(y^*, \pi^*, r^*) = (y_e, \pi^T, r_s)$. Three scenarios will then be considered. In scenario 1, there is an increase in autonomous aggregate demand (A). In scenario 2, the central bank sets a higher inflation target (π^T). Scenario 3 considers a rise in equilibrium output (y_e).

Table 1: Parameterisation

Scenario	a_1	a_2	b	A	π^T	y_e
1: rise in aggregate demand (A)	0.3	0.7	1	12	2	5

Scenario	a_1	a_2	b	A	π^T	y_e
2: higher inflation target (π^T)	0.3	0.7	1	10	2.5	5
3: rise in equilibrium output (y_e)	0.3	0.7	1	10	2	7

10.3.2 Simulation code

```

#Clear the environment
rm(list=ls(all=TRUE))

# Set number of periods
Q=50

# Set number of scenarios
S=3

# Set period in which shock/shift will occur
s=5

# Create (S x Q)-matrices that will contain the simulated data
y=matrix(data=0,nrow=S,ncol=Q) # Income/output
p=matrix(data=0,nrow=S,ncol=Q) # Inflation rate
r=matrix(data=0,nrow=S,ncol=Q) # Real interest rate
rs=matrix(data=0,nrow=S,ncol=Q) # Stabilising interest rate

# Set constant parameter values
a1=0.3 # Sensitivity of inflation with respect to output gap
a2=0.7 # Sensitivity of output with respect to interest rate
b=1    # Sensitivity of central bank to inflation gap
a3=(a1*(1/(b*a2) + a2))^(−1)

# Set parameter values for different scenarios
A=matrix(data=10,nrow=S,ncol=Q) # autonomous spending
pt=matrix(data=2,nrow=S,ncol=Q) # Inflation target

```

```

ye=matrix(data=5,nrow=S,ncol=Q) # Potential output

A[1,s:Q]=12 # scenario 1: AD boost
pt[2,s:Q]=3 # scenario 2: higher inflation target
ye[3,s:Q]=7 # scenario 3: higher potential output

# Initialise endogenous variables at equilibrium values
y[,1]=ye[,1]
p[,1]=pt[,1]
rs[,1]=(A[,1] - ye[,1])/a1
r[,1]=rs[,1]

# Simulate the model by looping over Q time periods for S different scenarios
for (i in 1:S){

  for (t in 2:Q){

    #(1) IS curve
    y[i,t] = A[i,t] - a1*r[i,t-1]

    #(2) Phillips Curve
    p[i,t] = p[i,t-1] + a2*(y[i,t]-ye[i,t])

    #(3) Stabilising interest rate
    rs[i,t] = (A[i,t] - ye[i,t])/a1

    #(4) Monetary policy rule, solved for r
    r[i,t] = rs[i,t] + a3*(p[i,t]-pt[i,t])

  } # close time loop
} # close scenarios loop

```

 Python code

```

import numpy as np

# Set number of periods
Q = 50

# Set number of scenarios
S = 3

# Set period in which shock/shift will occur
s = 5

# Create (S x Q) arrays to store simulated data
y = np.zeros((S, Q)) # Income/output
p = np.zeros((S, Q)) # Inflation rate
r = np.zeros((S, Q)) # Real interest rate
rs = np.zeros((S, Q)) # Stabilizing interest rate

# Set constant parameter values
a1 = 0.3 # Sensitivity of inflation with respect to output gap
a2 = 0.7 # Sensitivity of output with respect to interest rate
b = 1 # Sensitivity of the central bank to inflation gap
a3 = (a1 * (1 / (b * a2) + a2)) ** (-1)

# Set parameter values for different scenarios
A = np.full((S, Q), 10) # Autonomous spending
pt = np.full((S, Q), 2) # Inflation target
ye = np.full((S, Q), 5) # Potential output

A[0, s:Q] = 12 # Scenario 1: AD boost
pt[1, s:Q] = 3 # Scenario 2: Higher inflation target
ye[2, s:Q] = 7 # Scenario 3: Higher potential output

# Initialize endogenous variables at equilibrium values
y[:, 0] = ye[:, 0]
p[:, 0] = pt[:, 0]
rs[:, 0] = (A[:, 0] - ye[:, 0]) / a1
r[:, 0] = rs[:, 0]

# Simulate the model by looping over Q time periods for S different scenarios
for i in range(S):
    for t in range(1, Q):
        # (1) IS curve
        y[i, t] = A[i, t] - a1 * r[i, t - 1]
        # (2) Phillips Curve
        p[i, t] = p[i, t - 1] + a2 * (y[i, t] - ye[i, t])
        # (3) Stabilizing interest rate
        rs[i, t] = (A[i, t] - ye[i, t]) / a1
        # (4) Monetary policy rule, solved for r
        r[i, t] = rs[i, t] + a3 * (p[i, t] - pt[i, t])

```

10.3.3 Plots

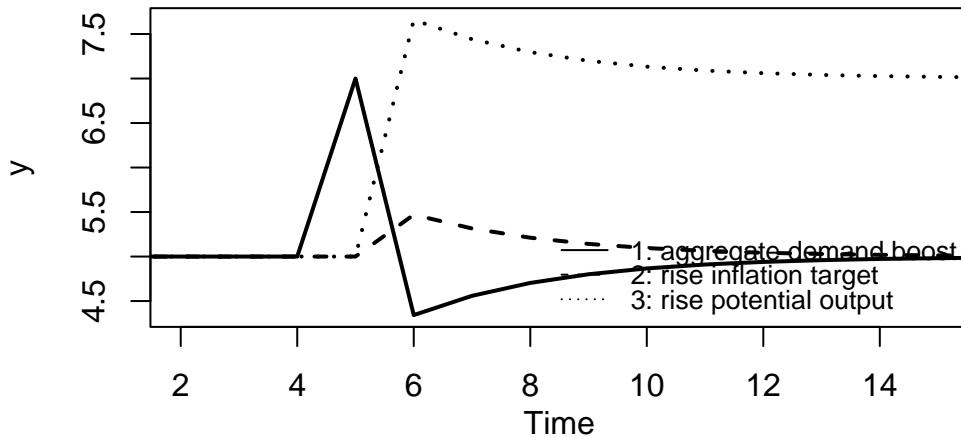
Figures 1-3 depict the response of the model's key endogenous variables to various shifts. A permanent rise in aggregate demand (scenario 1) has an instantaneous expansionary effect on output, but also pushes inflation above the target. This induces the central bank to raise the interest rate, which brings down output below equilibrium in the next period. The central bank then gradually lowers the policy rate towards its new higher equilibrium value, where inflation is again stabilised at its target level.

```
#### Plot results

#### Plots
# Set maximum period for plots
Tmax=15

# Output under different scenarios
plot(y[1, 1:(Tmax+1)], type="l", col=1, lwd=2, lty=1, xlab="", xlim=range(2:(Tmax)), ylab=""
title(main="Figure 1: Output under different scenarios", xlab = "Time", cex=0.8 ,line=2)
lines(y[2, 1:(Tmax+1)], lty=2, lwd=2)
lines(y[3, 1:(Tmax+1)], lty=3, lwd=2)
legend("bottomright", legend=c("1: aggregate demand boost", "2: rise inflation target", "3:
```

Figure 1: Output under different scenarios

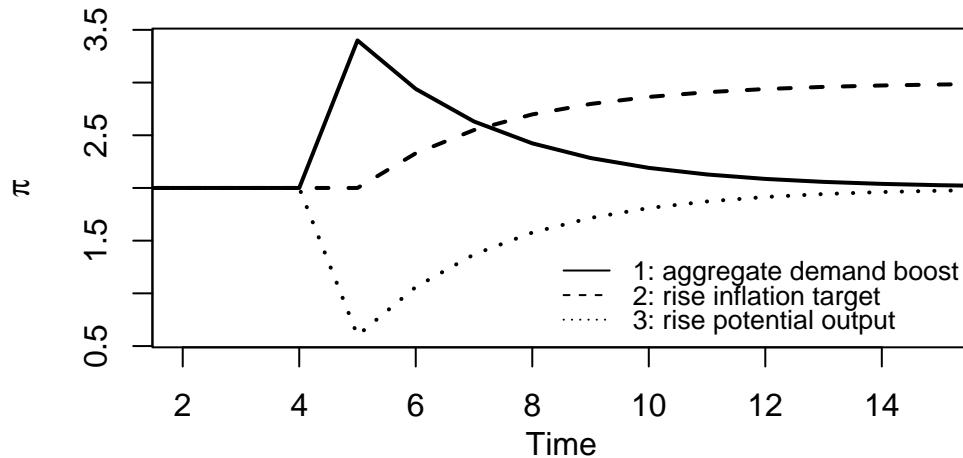


```

# Inflation under different scenarios
plot(p[1, 1:(Tmax+1)],type="l", col=1, lwd=2, lty=1, xlab="", xlim=range(2:(Tmax)), ylab=
title(main="Figure 2: Inflation under different scenarios", xlab = "Time",cex=0.8 ,line=2)
lines(p[2, 1:(Tmax+1)],lty=2, lwd=2)
lines(p[3, 1:(Tmax+1)],lty=3, lwd=2)
legend("bottomright", legend=c("1: aggregate demand boost", "2: rise inflation target", "3:

```

Figure 2: Inflation under different scenarios

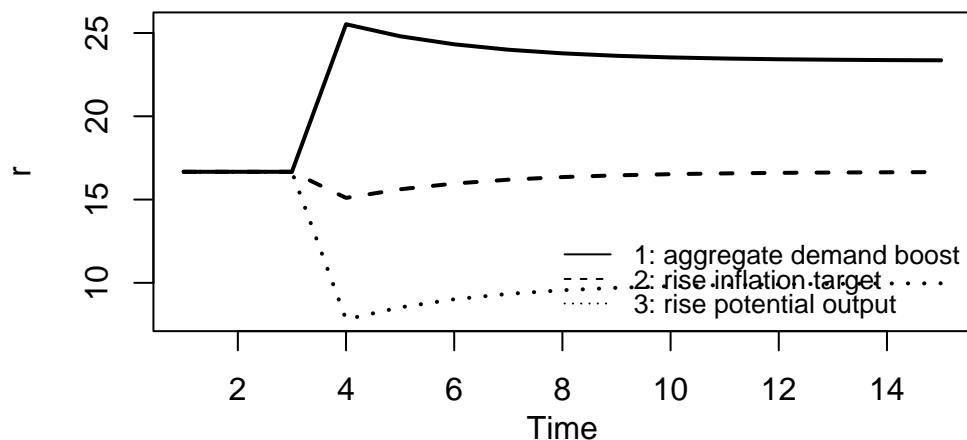


```

# Policy rate under different scenarios
plot(r[1, 2:(Tmax+1)],type="l", col=1, lwd=2, lty=1, xlab="", xlim=range(1:(Tmax)), ylab=
title(main="Figure 3: Policy rate under different scenarios", xlab = "Time",cex=0.8 ,line=2)
lines(r[2, 2:(Tmax+1)],lty=2, lwd=2)
lines(r[3, 2:(Tmax+1)],lty=3, lwd=2)
legend("bottomright", legend=c("1: aggregate demand boost", "2: rise inflation target", "3:

```

Figure 3: Policy rate under different scenarios



Python code

```
import matplotlib.pyplot as plt

# Set maximum period for plots
Tmax = 15

# Plot output under different scenarios
plt.figure(figsize=(8, 6))
plt.plot(y[0, :Tmax + 1], label="Scenario 1: aggregate demand boost",
          color='k', linestyle='solid', linewidth=2)
plt.plot(y[1, :Tmax + 1], label="Scenario 2: Rise inflation target",
          color='k', linestyle='dashed', linewidth=2)
plt.plot(y[2, :Tmax + 1], label="Scenario 3: Rise potential output",
          color='k', linestyle='dotted', linewidth=2)

plt.title("Output under Different Scenarios")
plt.xlabel("Time")
plt.ylabel("y")
plt.xlim(1, Tmax)
plt.ylim(np.min(y), np.max(y))
plt.legend()
plt.show()
```

An increase in the central bank's inflation target (scenario 2) gradually raises the inflation rate to a new level. During the adjustment period, the interest rate falls, which temporarily allows for a higher level of output. However, there is no permanent expansionary effect.

By contrast, an increase in potential or equilibrium output (scenario 3) allows for a permanently higher level of output and a lower real interest rate.

10.4 Directed graph

Another perspective on the model's properties is provided by its directed graph. A directed graph consists of a set of nodes that represent the variables of the model. Nodes are connected by directed edges. An edge directed from a node x_1 to node x_2 indicates a causal impact of x_1 on x_2 .

```

# Construct auxiliary Jacobian matrix for 7 variables: y, p, r, A, ye, rs, pt
# where non-zero elements in regular Jacobian are set to 1 and zero elements are unchanged

M_mat=matrix(c(0,0,1,1,0,0,0,
              1,0,0,0,1,0,0,
              0,1,0,0,0,1,1,
              0,0,0,0,0,0,0,
              0,0,0,0,0,0,0,
              0,0,0,1,1,0,0,
              0,0,0,0,0,0,0),7,7, byrow=TRUE)

# Create adjacency matrix from transpose of auxiliary Jacobian and add column names
A_mat=t(M_mat)

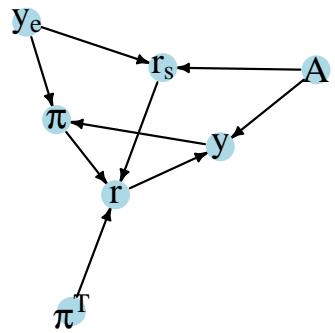
# Create directed graph from adjacency matrix
library(igraph)
dg=graph_from_adjacency_matrix(A_mat, mode="directed", weighted=NULL)

# Define node labels
V(dg)$name=c("y", expression(pi), "r", "A", expression(y[e]), expression(r[s]), expression(r[t]))

# Plot directed graph
plot(dg, main="Figure 4: Directed graph of 3-Equation model", vertex.size=20, vertex.color="black",
      vertex.label.color="black", edge.arrow.size=0.3, edge.width=1.1, edge.size=1.2,
      edge.arrow.width=1.2, edge.color="black", vertex.label.cex=1.2,
      vertex.frame.color="NA", margin=-0.08)

```

Figure 4: Directed graph of 3–Equation model



 Python code

```

# Directed graph
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

# Define the Jacobian matrix
M_mat = np.array([[0, 0, 1, 0, 0, 0],
                  [1, 0, 0, 0, 1, 0, 0],
                  [0, 1, 0, 0, 0, 1, 1],
                  [0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 1, 1, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0],
                  ])
]

# Create adjacency matrix from transpose of auxiliary Jacobian and add column names
A_mat = M_mat.transpose()

# Create the graph from the adjacency matrix
G = nx.DiGraph(A_mat)

# Define node labels
nodelabs = {
    0: "y",
    1: "",
    2: "r",
    3: "A",
    4: "y",
    5: "r",
    6: ""
}

# Plot the directed graph
pos = nx.spring_layout(G, seed=43)
nx.draw(G, pos, with_labels=True, labels=nodelabs, node_size=300, node_color='lightblue',
        font_size=10)
edge_labels = {(u, v): '' for u, v in G.edges}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='black')
plt.axis('off')
plt.show()

```

In Figure 4, it can be seen that aggregate demand (A), equilibrium output (y_e), and the inflation target (π^T) are the key exogenous variables of the model. All other variables are endogenous and form a closed loop (or cycle) within the system. The upper-right side of the graph represents the supply side, given by the equilibrium level of output and its effect on inflation. The upper-left side captures the demand side and its effect on actual output. The key endogenous variables, output, inflation, and the interest rate form the centre of the graph, where they stand in a triangular relationship to each other. Output drives inflation, which in turn impacts the real interest rate. The latter then feeds back into output. Structural changes in the relationship between demand and supply (e.g. excess demand) also impact the system through their effect on the stabilising interest rate (r_s).

10.5 Analytical discussion

10.5.1 Derivation of core equations

10.5.1.1 IS curve

The IS curve in Equation 10.1 is loosely based on the consumption Euler equation introduced in Chapter 3. Suppose there are two periods and the household maximises its utility function $U = \ln(C_t) + \beta \ln(C_{t+1})$ subject to the intertemporal budget constraint $C_t + \frac{C_{t+1}}{1+r} = Y_t + \frac{Y_{t+1}}{1+r}$. Substituting the constraint into the objective function and differentiating with respect to C_t yields the first-order condition:

$$C_t = \frac{C_{t+1}}{\beta(1+r)}.$$

This consumption Euler equation establishes the negative relationship between the real interest rate and expenditures in Equation 10.1.

10.5.1.2 PC curve

The Phillips curve Equation 10.2 is derived from wage- and price-setting in imperfect labour markets.³ Consider the following wage- and price-setting functions:

$$\begin{aligned} \frac{W}{P_E} &= B + \alpha(y_t - y_e) + z_w \\ P &= (1 + \mu) \frac{W}{\lambda}, \end{aligned} \tag{10.5}$$

³See chapter 2 of Carlin and Soskice (2014) for details.

i.e. the nominal wage W , adjusted for the expected price level, is increasing in the output gap, a factor B capturing unemployment benefits and the disutility of work as well as a vector z_w of wage-push factors. Prices are set based on a constant mark-up (μ) on unit labour cost ($\frac{W}{\lambda}$).

In equilibrium, the real wage is given by: $w_e = B + z_w$. In a dynamic setting, wage setters will raise the expected real wage by $\left(\frac{W_t}{P_t^E}\right) - \left(\frac{W_{t-1}}{P_{t-1}^E}\right) = \alpha(y_t - y_e)$. Together with adaptive expectations for prices $\hat{P}_t^E = \hat{P}_{t-1}$ and the approximation $\hat{W}_t - P_{t-1}^E \approx \left(\frac{W_t}{P_t^E}\right) - \left(\frac{W_{t-1}}{P_{t-1}^E}\right)$, this yields the following equation for wage inflation:

$$\hat{W}_t = \hat{P}_{t-1} + \alpha(y_t - y_e). \quad (10.6)$$

Transforming equation Equation 10.5 into growth rates ($\hat{P} = \hat{W}$) and combining it with the wage-inflation equation Equation 10.6 yields the Phillips curve Equation 10.2.

10.5.1.3 Monetary policy rule

Finally, to derive the interest rate rule, start from the following central bank loss function:⁴

$$L = (y_t - y_e)^2 + b(\pi_t - \pi^T)^2.$$

Substituting the Phillips curve (Equation 10.2) into the loss function, differentiating with respect to y_t , and simplifying yields the first-order condition:

$$y_t - y_e = -a_2 b (\pi - \pi^T),$$

which can also be regarded as a monetary policy rule. Next, substitute the Phillips curve (Equation 10.2), the IS-curve (Equation 10.1), and the stabilising interest rate (Equation 10.3) into the monetary policy rule and define $a_3 = \frac{1}{a_1(\frac{1}{a_2 b} + a_2)}$, which yields the interest rate rule (Equation 10.4).

10.5.2 Equilibrium solutions and stability analysis

By definition, in the steady state we have $y^* = y_e$. This implies that $r^* = r_s$. From this, it follows that $\pi^* = \pi^T$.

To analyse the dynamic stability of the model, we rewrite it as a system of first-order difference equations. To this end, substitute Equation 10.1 into Equation 10.2, which yields:

$$\pi_t = \pi_{t-1} + a_2(A - a_1 r_{t-1} - y_e) \quad (10.7)$$

⁴See chapter 2 of Carlin and Soskice (2014) for details.

Substitute this equation into Equation 10.4, which yields:

$$r_t = r_s + a_3[\pi_{t-1} + a_2(A - a_1 r_{t-1} - y_e) - \pi^T]. \quad (10.8)$$

The Jacobian matrix of the system in Equation 10.1, Equation 10.7, and Equation 10.8 is given by:

$$J = \begin{bmatrix} 0 & 0 & -a_1 \\ 0 & 1 & -a_1 a_2 \\ 0 & a_3 & -a_1 a_2 a_3 \end{bmatrix}.$$

The eigenvalues of the Jacobian can be obtained from the characteristic polynomial $\lambda^3 - Tr(J)\lambda^2 + [Det(J_1) + Det(J_2) + Det(J_3)]\lambda - Det(J) = 0$, where $Tr(J)$ and $Det(J)$ are the trace and determinant, respectively, and $Det(J_i)$ refers to the i_{th} principal minor of the matrix. As there is a column in the Jacobian that only contains zeros, it follows that the matrix is singular and will have a zero determinant. In addition, all principal minors turn out to be zero. The characteristic polynomial thus reduces to $\lambda^2[\lambda - Tr(J)] = 0$. From this, it is immediate that $\lambda_{1,2} = 0$ and $\lambda_3 = Tr(J)$, where $Tr(J) = 1 - a_1 a_2 a_3 = \frac{1}{1+a_2^2 b}$. Stability requires the single real eigenvalue to be smaller than unity (in absolute terms). With $\lambda_3 = \frac{1}{1+a_2^2 b}$, stability thus only requires $a_2 \neq 0$ and $b > 0$, i.e. the output gap needs to impact inflation (otherwise the key channel through which interest rate policy brings inflation back on target is blocked) and the central bank needs to assign a (non-negative) loss to deviations of actual inflation from its target.⁵

We can verify these analytical solutions by comparing them with the results from the numerical solution:

```
# Construct Jacobian matrix
J=matrix(c(0,0,-a1,
          0,1,-a1*a2,
          0,a3,-a1*a2*a3), 3, 3, byrow=TRUE)

# Obtain eigenvalues
ev=eigen(J)
(values <- ev$values)
```

[1] 0.6711409 0.0000000 0.0000000

⁵As mentioned in footnote 2, this property of the Carlin-Soskice model is very different from conventional New Keynesian models with rational expectations. In these models, variables such as output and inflation are driven by the ‘forward-looking’ behaviour of rational agents, i.e. they depend on expectational terms for their current values rather than lagged values. To ensure what is called ‘determinancy’, these forward-looking variables must adjust fast (or ‘jump’) to bring the economy back onto a path that is consistent with the optimising equilibrium. This requires the number of jump variables to be matched by an equal number of unstable roots (i.e. being outside the unit circle).

```
# Obtain determinant and trace  
det(J)      # determinant
```

```
[1] 0
```

```
sum(diag(J)) # trace
```

```
[1] 0.6711409
```

 Python code

```
import numpy as np  
  
# Construct Jacobian matrix  
J = np.array([[0, 0, -a1],  
             [0, 1, -a1 * a2],  
             [0, a3, -a1 * a2 * a3],])  
  
# Calculate eigenvalues  
eigenvalues = np.linalg.eigvals(J)  
  
# Print the resulting eigenvalues  
print(eigenvalues)  
  
# Calculate the determinant and trace of the Jacobian matrix  
determinant = np.linalg.det(J)  
trace = np.trace(J)  
print(determinant)  
print(trace)
```

10.6 References

11 A Sraffian Supermultiplier Model

11.1 Overview

The Sraffian supermultiplier model was proposed by Serrano (1995) to integrate a Sraffian long-run equilibrium into a post-Keynesian growth model.¹ The model requires the long-run rate of capacity utilisation to settle on an exogenously given normal rate. This requires investment to fully adjust to any changes in economic activity so as to bring back actual utilisation to the desired normal rate. As a result, investment expenditures (in the long-run) are assumed to be free of any idiosyncratic components such as Keynesian ‘animal spirits’. Long-run growth is then driven by those components of autonomous demand that do not create productive capacity – autonomous consumption in the simplest version of the model. An increase in the growth rate of autonomous consumption will stimulate economic activity and induce firms to adjust their expectations about long-run growth towards the new rate given by autonomous demand growth.

Income distribution is exogenous in this model. An increase in the wage share has an expansionary effect on economic activity and growth in the short-run as it increases consumption (investment is assumed to be independent of income distribution). However, this expansionary effect is only temporary as economic activity will eventually settle back on the normal rate of capacity utilisation, and the growth rate towards the rate given by autonomous demand growth. The absence of long-run effects of income distribution on output and growth constitutes a key difference between the Sraffian supermultiplier model and the post-Kaleckian model, in which there is no normal rate of capacity utilisation and no autonomous (non-capacity creating) demand.

This is a model of long-run steady state growth. In the steady state, all endogenous variables grow at the same rate.² The model contains two state variables that determine the model’s dynamics: the ratio of autonomous demand to the capital stock (which changes during adjustment periods where the growth rate has not yet settled on the rate given by autonomous demand growth) and the expected growth rate of the capital stock, which sluggishly adjusts to the rate given by autonomous demand growth. We consider a continuous-time version of the model presented in chapter 6.5.8 of Lavoie (2022).³

¹See chapter 7 in Blecker and Setterfield (2019), Dutt (2018), and chapter 6 in Lavoie (2022) for useful introductions. Note that contrary to what the name may suggest, this is a one-sector model.

²All variables are normalised by the capital stock and thus rendered stationary.

³Section 2.5 explains how continuous time models can be solved numerically.

11.2 The Model

The following equations describe the model:

$$r_t = \pi u_t \quad (11.1)$$

$$s_t = -z_t + s_r r_t \quad (11.2)$$

$$c_t = u_t - s_t \quad (11.3)$$

$$g_t = g_t^0 + g_1(u_t - u_n) \quad (11.4)$$

$$u_t = c_t + g_t \quad (11.5)$$

$$\dot{g}_t^0 = \mu(g_t - g_t^0) \quad (11.6)$$

$$\dot{z}_t = z_t(g_z - g_t), \quad (11.7)$$

where r , s , c , g , u , g^0 , and z are the profit rate, the saving rate, the consumption rate, the investment rate, the rate of capacity utilisation, the expected growth rate, and the rate of autonomous demand, respectively. A dot over a variable represents the derivative with respect to time ($\dot{x} = \frac{dx}{dt}$).

Equation 11.1 decomposes the profit rate (total profits over capital stock) into the product of the profit share π (total profits over total output) and the rate of capacity utilisation (actual output over capital stock).⁴ Note that the wage share is given by $1 - \pi$. By Equation 11.2, the economy-wide saving rate is given by the negative of the rate of autonomous demand (z), which in this version of the model is autonomous consumption, i.e. dissaving, and saving out of profits (s_r). It is assumed that workers do not save. Equation 11.3 simply states that consumption is income not saved. According to Equation 11.4, investment is determined by an autonomous component g_0 that will be specified below and by the deviation of capacity utilisation from its normal rate u_n . In other words, firms expand capacity whenever the actual rate of utilisation exceeds the desired normal rate. Equation 11.5 is the goods market equilibrium condition assuming that the rate of capacity utilisation adjusts to clear the goods market in the short run. Equation 11.6 is a key equation in the Sraffian supermultiplier approach, which posits that firms (sluggishly) adjust the expected growth rate to the actual growth rate. Finally, Equation 11.7 is an identity that traces changes in the rate of autonomous demand that stem from (temporary) mismatches between the exogenously given growth rate of autonomous demand (g_z) and the actual growth rate.

⁴For simplicity, it is assumed that the capital-potential output ratio is equal to unity. This implies that the ratio of actual output to potential output is equal to the ratio of actual output to the capital stock, so that the latter can be taken as a measure of the rate of capacity utilisation.

11.3 Simulation

11.3.1 Parameterisation

Table 1 reports the parameterisation used in the simulation. Besides a baseline (labelled as scenario 1), three further scenarios will be considered. In scenario 2, the growth rate of autonomous demand g_z increases. In scenario 3, the profit share π rises. In scenario 4, the normal rate of capacity utilisation u_n increases. The model is initialised at the equilibrium of the baseline parameterisation and the various shifts then occur in period 50.

Table 1: Parameterisation

Scenario	π	s_r	g_1	u_n	μ	g_z
1: baseline	0.35	0.8	0.2	0.75	0.08	0.02
2: rise in autonomous demand growth (g_z)	0.35	0.8	0.2	0.75	0.08	0.03
3: rise in profit share (π)	0.4	0.8	0.2	0.75	0.08	0.02
4: rise in normal rate of capacity utilisation (u_n)	0.35	0.8	0.2	0.8	0.08	0.02

11.3.2 Simulation code

```
#Clear the environment
rm(list=ls(all=TRUE))

#Set number of periods
Q = 1000

# Set number of scenarios (including baselines)
S=4

# Set period in which exogenous shift will occur
q=50

#Create (S x Q) matrices in which equilibrium solutions from different parameterisations w
u=matrix(data=0,nrow=S,ncol=Q) # rate of capacity utilisation
g=matrix(data=0,nrow=S,ncol=Q) # growth rate of capital stock
s=matrix(data=0,nrow=S,ncol=Q) # saving rate
c=matrix(data=0,nrow=S,ncol=Q) # consumption rate
r=matrix(data=0,nrow=S,ncol=Q) # profit rate
g0=matrix(data=0,nrow=S,ncol=Q) # expected growth rate of capital stock
```

```

z=matrix(data=0,nrow=S,ncol=Q) # autonomous demand rate

#Set constant parameter values
g1=0.2 # Sensitivity of investment with respect to utilisation
sr=0.8 # propensity to save out of profits
mu=0.08 # adjustment speed of expected growth rate
d=0.1 # time increment

# Set and initialise exogenous variables/parameters that will be shifted
pi=matrix(data=0.35,nrow=S,ncol=Q) # profit share
gz=matrix(data=0.02,nrow=S,ncol=Q) # growth rate of autonomous demand
un=matrix(data=0.75,nrow=S,ncol=Q) # normal rate of capacity utilisation

# Set parameter values for different scenarios
gz[2,q:Q]=0.03 # scenario 2: rise in autonomous demand growth
pi[3,q:Q]=0.4 # scenario 3: rise in profit share
un[4,q:Q]=0.8 # scenario 4: rise in normal rate of utilisation

# Initialise endogenous variables at equilibrium values
u[,1]=un[,1]
g[,1]=gz[,1]
s[,1]=g[,1]
c[,1]=un[,1]-s[,1]
g0[,1]=gz[,1]
z[,1]=sr*pi[,1]*un[,1]-gz[,1]
r[,1]=pi[,1]*un[,1]

# Simulate the model by looping over Q time periods for S different scenarios
for (i in 1:S){

  for (t in 2:Q){

    for (iterations in 1:1000){ # iterate the model 1000-times in each period

      #(1) Profit rate
      r[i,t] =pi[i,t]*u[i,t]

      #(2) Saving
      s[i,t] = -z[i,t] + sr*r[i,t]

      #(3) Consumption
    }
  }
}

```

```

c[i,t] = u[i,t] - s[i,t]

#(4) Investment
g[i,t] = g0[i,t] + g1*(u[i,t] - un[i,t])

#(5) Capacity utilisation
u[i,t] = c[i,t] + g[i,t]

#(6) Dynamic adjustment of expected growth rate of capital stock
g0[i,t] = g0[i,t-1] + mu*(g[i,t-1]-g0[i,t-1])*d

#(7) Dynamic adjustment of autonomous demand
z[i,t] = z[i,t-1] + z[i,t-1]*(gz[i,t-1] - g[i,t-1])*d

} # close iterations loop
} # close time loop
} # close scenarios loop

```

 Python code

```

import numpy as np

# Set number of periods
Q = 1000

# Set number of scenarios (including baselines)
S = 4

# Set period in which exogenous shift will occur
q = 50

# Create (S x Q) matrices for equilibrium solutions
u = np.zeros((S, Q)) # rate of capacity utilization
g = np.zeros((S, Q)) # growth rate of capital stock
s = np.zeros((S, Q)) # saving rate
c = np.zeros((S, Q)) # consumption rate
r = np.zeros((S, Q)) # profit rate
g0 = np.zeros((S, Q)) # expected growth rate of capital stock
z = np.zeros((S, Q)) # autonomous demand rate

# Set constant parameter values
g1 = 0.2 # Sensitivity of investment with respect to utilization
sr = 0.8 # propensity to save out of profits
mu = 0.08 # adjustment speed of expected growth rate
d = 0.1 # time increment

# Set and initialize exogenous variables/parameters that will be shifted
pi = np.full((S, Q), 0.35) # profit share
gz = np.full((S, Q), 0.02) # growth rate of autonomous demand
un = np.full((S, Q), 0.75) # normal rate of capacity utilization

# Set parameter values for different scenarios
gz[1, q:] = 0.03 # scenario 2: rise in autonomous demand growth
pi[2, q:] = 0.4 # scenario 3: rise in profit share
un[3, q:] = 0.8 # scenario 4: rise in normal rate of utilization

# Initialize endogenous variables at equilibrium values
u[:, 0] = un[:, 0]
g[:, 0] = gz[:, 0]
s[:, 0] = g[:, 0]
c[:, 0] = un[:, 0] - s[:, 0]
g0[:, 0] = gz[:, 0]
z[:, 0] = sr * pi[:, 0] * un[:, 0] - gz[:, 0]
r[:, 0] = pi[:, 0] * un[:, 0]

# Simulate the model by looping over Q time periods for S different scenarios
for i in range(S):
    for t in range(1, Q):
        for iterations in range(1000): # iterate the model 1000 times in each period

            # (1) Profit rate
            r[i, t] = pi[i, t] * u[i, t]

```

11.3.3 Plots

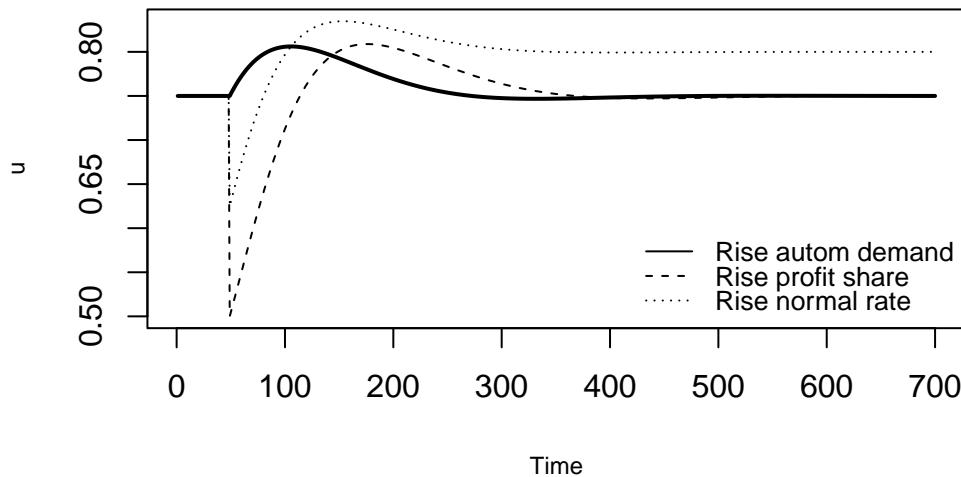
Figures 1-3 depict the response of the three main endogenous variables to changes in the exogenous variables. In the second scenario (solid line), the growth rate of autonomous demand increases from 2% to 3%. As a result, the rate of capacity temporarily increases but then returns to the level given by the normal rate, as the rate of autonomous demand falls due to the increase in the capital stock. By contrast, the growth rate permanently settles to the new rate given by the autonomous rate.

In the third scenario (dashed line), the profit share rises, which initially has a contractionary effect on the rate of utilisation and growth. Both variables then briefly overshoot due to the increase in the autonomous demand rate and then return to their previous values.

```
# Set maximum period for plots
Tmax=700

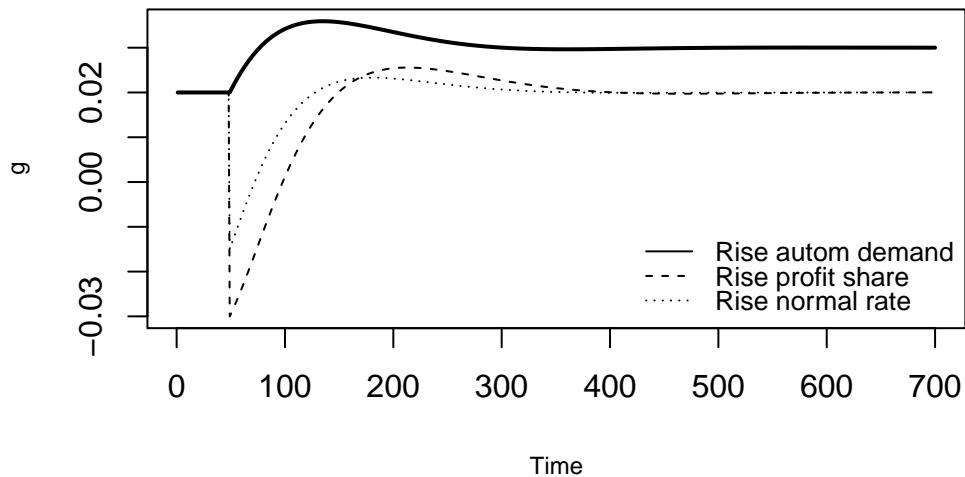
# Capacity utilisation
plot(u[2, 2:(Tmax+1)], type="l", col=1, lwd=2, lty=1, font.main=1,cex.main=1,
      ylab = 'u',xlab = 'Time',ylim=range(max(u[, 2:Tmax]),min(u[, 2:Tmax])),cex.axis=1,cex=
title(main="Figure 1: Rate of capacity utilisation under different scenarios",cex=0.8 ,lin
lines(u[3, 2:(Tmax+1)],lty=2)
lines(u[4, 2:(Tmax+1)],lty=3)
legend("bottomright", legend=c("Rise autom demand", "Rise profit share", "Rise normal rate",
      lty=1:3, cex=0.8, bty = "n", y.intersp=0.8)
```

Figure 1: Rate of capacity utilisation under different scenarios



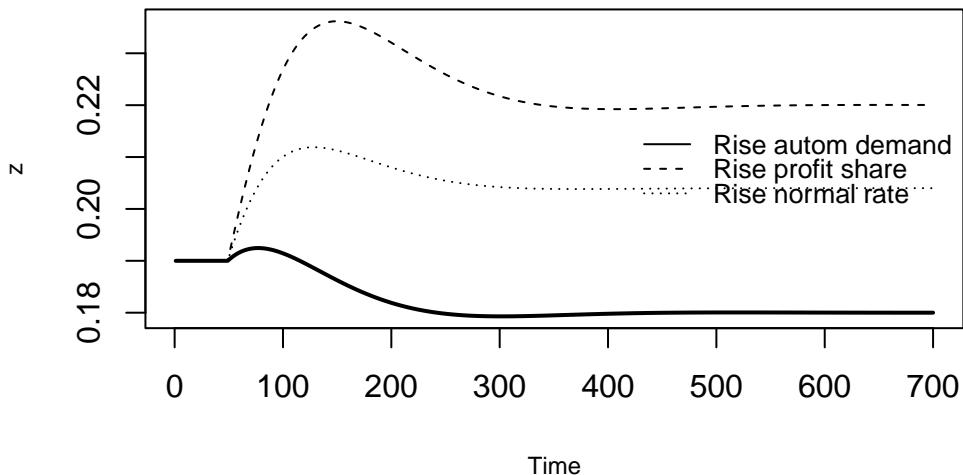
```
# Growth
plot(g[2, 2:(Tmax+1)], type="l", col=1, lwd=2, lty=1, font.main=1,cex.main=1,
      ylab = 'g',xlab = 'Time',ylim=range(max(g[, 2:Tmax]),min(g[, 2:Tmax])),cex.axis=1,cex=1,
      title(main="Figure 2: Growth rate under different scenarios",cex=0.8 ,line=2)
lines(g[3, 2:(Tmax+1)],lty=2)
lines(g[4, 2:(Tmax+1)],lty=3)
legend("bottomright", legend=c("Rise autom demand", "Rise profit share", "Rise normal rate",
      lty=1:3, cex=0.8, bty = "n", y.intersp=0.8)
```

Figure 2: Growth rate under different scenarios



```
# Autonomous demand rate
plot(z[2, 2:(Tmax+1)], type="l", col=1, lwd=2, lty=1, font.main=1,cex.main=1,
      ylab = 'z',xlab = 'Time',ylim=range(max(z[, 2:Tmax]),min(z[, 2:Tmax])),cex.axis=1,cex=1,
      title(main="Figure 3: Rate of autonomous demand under different scenarios",cex=0.8 ,line=2
lines(z[3, 2:(Tmax+1)],lty=2)
lines(z[4, 2:(Tmax+1)],lty=3)
legend("right", legend=c("Rise autom demand", "Rise profit share", "Rise normal rate"),
      lty=1:3, cex=0.8, bty = "n", y.intersp=0.8)
```

Figure 3: Rate of autonomous demand under different scenarios



i Python code

```
## Plots (here for capacity utilisation only)

import matplotlib.pyplot as plt

# Set maximum period for plots
Tmax = 700

# Plot capacity utilization
plt.plot(u[1, 1:Tmax], label='Rise autom demand', linestyle='-', linewidth=2, color='k')
plt.plot(u[2, 1:Tmax], label='Rise profit share', linestyle='--', linewidth=2, color='k')
plt.plot(u[3, 1:Tmax], label='Rise normal rate', linestyle='-.', linewidth=2, color='k')
plt.title('Rate of Capacity Utilization under Different Scenarios')
plt.xlabel('Time')
plt.ylabel('u')
plt.legend(loc='lower right')
plt.show()
```

Finally, a rise in the normal rate (dotted line) initially has contractionary effects on utilisation and growth but eventually raises utilisation to a permanently higher level. The growth rate

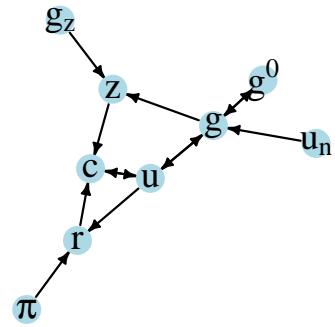
returns to its previous value.

11.4 Directed graph

Another perspective on the model's properties is provided by its directed graph. A directed graph consists of a set of nodes that represent the variables of the model. Nodes are connected by directed edges. An edge directed from a node x_1 to node x_2 indicates a causal impact of x_1 on x_2 .

```
# Construct auxiliary Jacobian matrix for 9 variables:  
# r, c, g, u, g0, z, pi, gz, un  
  
M_mat=matrix(c(0,0,0,1,0,0,1,0,0,  
              1,0,0,1,0,1,0,0,0,  
              0,0,0,1,1,0,0,0,1,  
              0,1,1,0,0,0,0,0,0,  
              0,0,1,0,0,0,0,0,0,  
              0,0,1,0,0,0,0,1,0,  
              0,0,0,0,0,0,0,0,0,  
              0,0,0,0,0,0,0,0,0,  
              0,0,0,0,0,0,0,0,0), 9, 9, byrow=TRUE)  
  
# Create adjacency matrix from transpose of auxiliary Jacobian and add column names  
A_mat=t(M_mat)  
  
# Create directed graph from adjacency matrix  
library(igraph)  
dg=graph_from_adjacency_matrix(A_mat, mode="directed", weighted=NULL)  
  
# Define node labels  
V(dg)$name=c("r", "c", "g", "u", expression(g^0), "z", expression(pi), expression(g[z]), e  
  
# Plot directed graph  
plot(dg, main="Figure 4: Directed graph of Sraffian Supermultiplier Model", vertex.size=20,  
      vertex.label.color="black", edge.arrow.size=0.3, edge.width=1.1, edge.size=1.2,  
      edge.arrow.width=1.2, edge.color="black", vertex.label.cex=1.2,  
      vertex.frame.color="NA", margin=-0.08)
```

Figure 4: Directed graph of Sraffian Supermultiplier Model



 Python code

```

# Directed graph
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

# Define the Jacobian matrix
M_mat = np.array([[0, 0, 0, 1, 0, 0, 1, 0, 0],
                  [1, 0, 0, 1, 0, 1, 0, 0, 0],
                  [0, 0, 0, 1, 1, 0, 0, 0, 1],
                  [0, 1, 1, 0, 0, 0, 0, 0, 0],
                  [0, 0, 1, 0, 0, 0, 0, 0, 0],
                  [0, 0, 1, 0, 0, 0, 0, 1, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0, 0]])

# Create adjacency matrix from transpose of auxiliary Jacobian and add column names
A_mat = M_mat.transpose()

# Create the graph from the adjacency matrix
G = nx.DiGraph(A_mat)

# Define node labels
nodelabs = {0: "r",
            1: "c",
            2: "g",
            3: "u",
            4: r"$g^0$",
            5: "z",
            6: r"$\pi$",
            7: r"$g_z$",
            8: r"$u_n$"}

# Plot the directed graph
pos = nx.spring_layout(G, seed=43)
nx.draw(G, pos, with_labels=True, labels=nodelabs, node_size=300, node_color='lightblue',
        font_size=10)
edge_labels = {(u, v): '' for u, v in G.edges}
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_color='black')
plt.axis('off')
plt.show()

```

In Figure 4, it can be seen that the growth rate of autonomous demand (g_z), the profit share (π), and the normal rate of capacity utilisation (u_n) are the key exogenous variable of the model. The profit rate (r), consumption (c), the autonomous demand rate (z), investment (g), the rate of utilisation (u), and the expected growth rate (g_0) form a closed loop (or cycle) within the system. For example, an increase in the growth rate of autonomous demand increases consumption, which raises the rate of capacity utilisation, growth, and the expected growth rate. In a second-round effect, the increase in the growth rate then feeds back negatively into the autonomous demand rate, which leads to a return of the rate of capacity utilisation to its previous value.

11.5 Analytical discussion

To find the short-run equilibrium solutions for u and g , first substitute Equation 11.1–Equation 11.4 into Equation 11.5 and solve for u :

$$u^* = \frac{g_0 + z - g_1 u_n}{s_r \pi - g_1}.$$

From this, we get:

$$g^* = g_0 + g_1(u^* - u_n).$$

The long-run equilibrium is given by $u^{**} = u_n$, $g^{**} = g_z$, and (from Equation 11.7) $z^{**} = u_n s_r \pi - g_z$.

The dynamics are governed by Equation 11.6–Equation 11.7. The Jacobian matrix is:

$$J(g^0, z) = \begin{bmatrix} \frac{\mu g_1}{s_r \pi - g_1} & \frac{\mu g_1}{s_r \pi - g_1} \\ -z \left(\frac{g_1}{s_r \pi - g_1} + 1 \right) & \frac{-z g_1}{s_r \pi - g_1} \end{bmatrix}.$$

The determinant of the Jacobian matrix evaluated at the long-run equilibrium is:

$$\det(J^*) = \frac{(u_n s_r \pi - g_z) \mu g_1}{s_r \pi - g_1} > 0,$$

which is positive provided $s_r \pi - g_1$, i.e. if the Keynesian stability condition holds.

The trace is:

$$\text{tr}(J^*) = \frac{g_1(\mu - u_n s_r \pi + g_z)}{s_r \pi - g_1}.$$

Stability requires a negative trace, yielding a second stability condition: $\mu < u_n s_r \pi - g_z$.

We can verify these analytical solutions by comparing them with the results from the numerical solution:

```

# Construct Jacobian matrix at the equilibrium

J=matrix(c((mu*g1)/(sr*pi[1,Q]-g1), (mu*g1)/(sr*pi[1,Q]-g1),
           -z[1,Q]*(sr*pi[1,Q]/(sr*pi[1,Q]-g1)), -z[1,Q]*(g1/(sr*pi[1,Q]-g1))), 2, 2, byro
J

[,1]   [,2]
[1,] 0.200  0.200
[2,] -0.665 -0.475

# Obtain eigenvalues
ev=eigen(J)
(values <- ev$values)

[1] -0.1375+0.1381801i -0.1375-0.1381801i

# Obtain determinant and trace
det(J) # determinant

[1] 0.038

sum(diag(J)) # trace

[1] -0.275

# Check stability conditions for all scenarios
for (i in 1:S){
  print(paste0("Scenario ", i, ":"))
  print(sr*pi[i,1]>g1) # Keynesian stability condition
  print(mu<sr*un[i,1]*pi[i,1]-gz[i,1])
}

[1] "Scenario 1:"
[1] TRUE
[1] TRUE
[1] "Scenario 2:"
[1] TRUE

```

```
[1] TRUE  
[1] "Scenario 3:"  
[1] TRUE  
[1] TRUE  
[1] "Scenario 4:"  
[1] TRUE  
[1] TRUE
```

 Python code

```
# Construct Jacobian matrix  
J = np.array([  
    [(\mu * g1) / (sr * pi[0, Q-1] - g1), (\mu * g1) / (sr * pi[0, Q-1] - g1)],  
    [-z[0, Q-1] * (sr * pi[0, Q-1] / (sr * pi[0, Q-1] - g1)), -z[0, Q-1] * (g1 / (sr * p  
)])  
print(J)  
  
# Obtain eigenvalues, determinant, and trace  
  
eigenvalues, eigenvectors = np.linalg.eig(J)  
print(eigenvalues)  
  
determinant_J = np.linalg.det(J)  
print(determinant_J)  
  
trace_J = np.trace(J)  
print(trace_J)  
  
# Check stability conditions for all scenarios  
for i in range(1, S + 1):  
    print(f"Scenario {i}:")  
    print(sr * pi[i - 1, 0] > g1) # Keynesian stability condition  
    print(mu < sr * un[i - 1, 0] * pi[i - 1, 0] - gz[i - 1, 0])
```

11.6 References

12 A Malthusian Model

12.1 Overview

This model captures some key feature of Thomas Malthus' theory of population dynamics as developed in his 1798 book [An Essay on the Principle of Population](#). The theory revolves around the interaction between living standards and population growth.¹ It assumes that birth rates increase with rising living standards, while death rates decline. Economic growth thus spurs population growth. However, due to supply constraints in agricultural production, population growth drives up food prices and thereby undermines real income, bringing population growth to a halt. The model is adapted from [Karl Whelan's lecture notes](#).

12.2 The Model

The following equations describe the model:

$$N_t = N_{t-1} + B_{t-1} - D_{t-1} \quad (12.1)$$

$$\frac{B_t}{N_t} = b_0 + b_1 Y_t \quad (12.2)$$

$$\frac{D_t}{N_t} = d_0 - d_1 Y_t \quad (12.3)$$

$$Y_t = a_0 - a_1 N_t \quad (12.4)$$

where N_t , B_t , D_t , and Y_t represent population, number of births, number of deaths, and real income, respectively.

Equation 12.1 describes population dynamics as driven by births and deaths. Equation 12.2 and Equation 12.3 the Malthusian hypothesis that birth rates are positively and death rates negatively related to income. Equation 12.4 makes real income a negative function of the population, which captures the idea of supply constraints in agriculture.

¹See chapter 2 of Foley (2006) for an excellent introduction.

12.3 Simulation

12.3.1 Parameterisation

Table reports the parameterisation and initial values used in the simulation. Besides a baseline (labelled as scenario 1), three further scenarios will be considered. Scenario 2 models what Malthus called preventative checks: a fall in the exogenous component of the birth rate (b_0) due to an increased use of contraception, changes in marriage norms etc. Scenario 3 models ‘positive checks’: a rise in the sensitivity of real income with respect to the population (a_1), capturing factors such as increased food scarcity. Scenario 4 considers a rise in the exogenous component of real income (a_0), which could be interpreted as a productivity boost due to the invention of better fertilisers. All scenarios initialise the population below its steady state value at $N_0 = 1$ and the other variables at their steady state values.

Table 1: Parameterisation

Scenario	b_0	b_1	d_0	d_1	a_0	a_1
1: baseline	0.5	0.5	2.5	0.5	2.5	0.05
2: fall in exog birth rate (b_0)	0.4	0.5	2.5	0.5	2.5	0.05
3: rise in sensitivity of income (a_1)	0.5	0.5	2.5	0.5	2.5	0.07
4: productivity boost (a_0)	0.5	0.5	2.5	0.5	2.6	0.05

12.3.2 Simulation code

```
# Clear the environment
rm(list=ls(all=TRUE))

# Set number of periods
T=100

# Set number of scenarios (including baseline)
S=4

# Set period in which shock/shift will occur
s=5

# Create (S x T)-matrices that will contain the simulated data
N=matrix(data=0,nrow=S,ncol=T) # population
Y=matrix(data=0,nrow=S,ncol=T) # real income
B=matrix(data=0,nrow=S,ncol=T) # births
D=matrix(data=0,nrow=S,ncol=T) # deaths
```

```

N_eq=vector(length=S)           # equilibrium population
Y_eq=vector(length=S)           # equilibrium real income
B_eq=vector(length=S)           # equilibrium births
D_eq=vector(length=S)           # equilibrium deaths

# Set baseline parameter values
b0=matrix(data=0.5,nrow=S,ncol=T) # Exogenous birth rate
b1=0.5 # Sensitivity of births with respect to real income
d0=2.5 # Exogenous death rate
d1=0.5 # Sensitivity of deaths with respect to real income
a0=matrix(data=2.5,nrow=S,ncol=T) # Exogenous component of real income
a1=matrix(data=0.05,nrow=S,ncol=T) #Sensitivity of the real income with respect to populat

# Set parameter values for different scenarios
b0[2,s:T]=0.4 # scenario 2: reduction in birth rate
a1[3,s:T]=0.07 # scenario 3: increase in sensitivity of real income with respect to popula
a0[4,s:T]=2.6 # scenario 4: improvement in productivity

# Initialise
N[,1]=1
Y[,1]=1
B[,1]=1
D[,1]=1

# Simulate the model by looping over T time periods for S different scenarios
for (i in 1:S){

  for (t in 2:T){

    for (iterations in 1:1000){ # run the model 1000-times in each period

      # (1) Population dynamics
      N[i,t] = N[i,t-1] + B[i,t-1] - D[i,t-1]

      # (2) Births
      B[i,t] = (b0[i,t] + b1*Y[i,t])*N[i,t]

      # (3) Deaths
      D[i,t] = (d0 - d1*Y[i,t])*N[i,t]

      # (4) Real income
    }
  }
}

```

```
Y[i,t] = a0[i,t] - a1[i,t]*N[i,t]

} # close iterations loop
} # close time loop
} # close scenario loop
```

 Python code

```

# Load relevant libraries
import numpy as np

# Set number of periods
T = 100

# Set number of scenarios (including baseline)
S = 4

# Set period in which shock/shift will occur
s = 5

# Create (S x T)-matrices that will contain the simulated data
N = np.zeros((S, T)) # population
Y = np.zeros((S, T)) # real income
B = np.zeros((S, T)) # births
D = np.zeros((S, T)) # deaths
N_eq = np.zeros((S)) # equilibrium population
Y_eq = np.zeros((S)) # equilibrium real income
B_eq = np.zeros((S)) # equilibrium births
D_eq = np.zeros((S)) # equilibrium deaths

# Set baseline parameter values
b0 = np.zeros((S, T)) + 0.5 # Exogenous birth rate
b1 = 0.5 # Sensitivity of births with respect to real income
d0 = 2.5 # Exogenous death rate
d1 = 0.5 # Sensitivity of deaths with respect to real income
a0 = np.zeros((S, T)) + 2.5 # Exogenous component of real income
a1 = np.zeros((S, T)) + 0.05 # Sensitivity of the real income with respect to population

# Set parameter values for different scenarios
b0[1, s:T] = 0.4 # scenario 2: reduction in birth rate
a1[2, s:T] = 0.07 # scenario 3: increase in sensitivity of real income with respect to
a0[3, s:T] = 2.6 # scenario 4: improvement in productivity

# Initialise
N[:,0] = 1
Y[:,0] = Y_eq[0]
B[:,0] = B_eq[0]
D[:,0] = D_eq[0]

# Simulate the model by looping over T time periods for S different scenarios
for i in range(S):
    for t in range(1, T):
        for iterations in range(1000): # run the model 1000-times in each period (to make sure it converges)
            # (1) Population dynamics
            N[i,t] = N[i,t-1] + B[i,t-1] - D[i,t-1]
            # (2) Births
            B[i,t] = (b0[i,t] + b1*Y[i,t])*N[i,t]
            # (3) Deaths
            D[i,t] = (d0 - d1*Y[i,t])*N[i,t]
            # (4) Real income

```

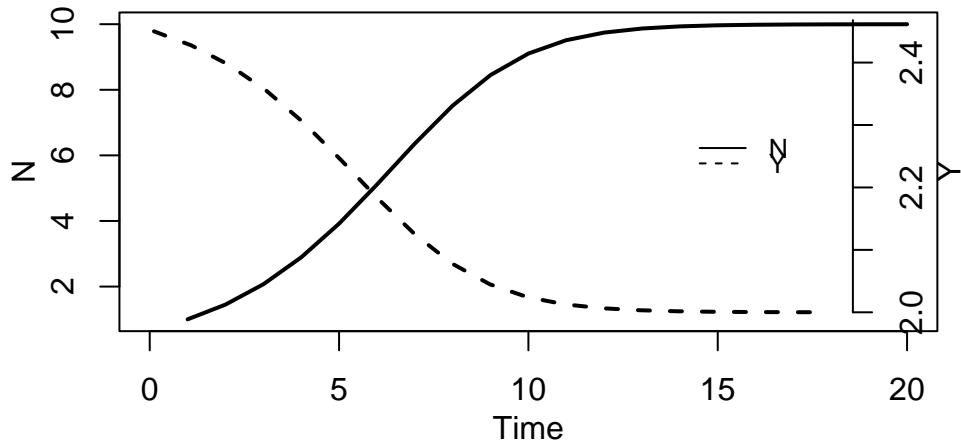
12.3.3 Plots

Figure 1 displays population and real income dynamics for the baseline scenario. Starting from a below-equilibrium level of population, the population initially grows rapidly (seemingly exponentially) but then approaches a steady state. During the adjustment phase, real income is driven down to its steady state level (which can be interpreted as the subsistence level). Figure 2 displays the corresponding dynamics of births and deaths.

```
# Set maximum period for plots
Tmax=20

## Baseline
#Population and real income
plot(N[1, 2:(Tmax+1)],type="l", lwd=2, lty=1, xlim=range(0:(Tmax)), ylab = '', xlab = '')
title(main="Figure 1: Population and real income, baseline",ylab = 'N', xlab = 'Time',cex=1)
par(mar = c(5, 4, 4, 4) + 0.3)
par(new = TRUE)
plot(Y[1, 2:Tmax],type="l", col=1, lwd=2, lty=2, font.main=1,cex.main=1,ylab = '', axes=FALSE,
      xlab = '',ylim = range(Y[1, 2:20]),cex=0.8)
axis(side = 4, at = pretty(range(Y[1, 2:Tmax])))
mtext("Y", side = 4, line = 2)
legend(15, 2.3, legend=c("N", "Y"),
       lty=1:2, cex=0.8, bty = "n", y.intersp=0.5)
```

Figure 1: Population and real income, baseline



```
# Births and deaths
plot(B[1, 2:(Tmax+1)], type="l", col=1, lwd=2, lty=1, xlim=range(0:(Tmax)), xlab="", ylab="")
title(main="Figure 2: Births and deaths, baseline", xlab = 'Time', cex=0.8, line=2)
lines(D[1, 2:Tmax], lty=2)
legend("bottomright", legend=c("B", "D"),
       lty=1:2, cex=0.8, bty = "n", y.intersp=0.5)
```

Figure 2: Births and deaths, baseline

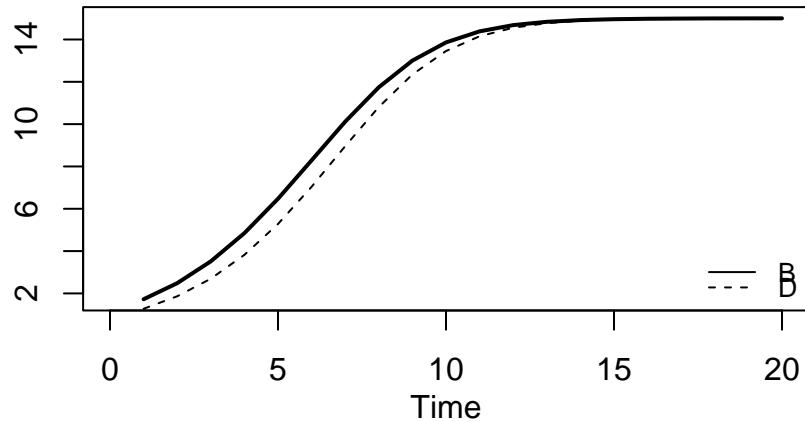
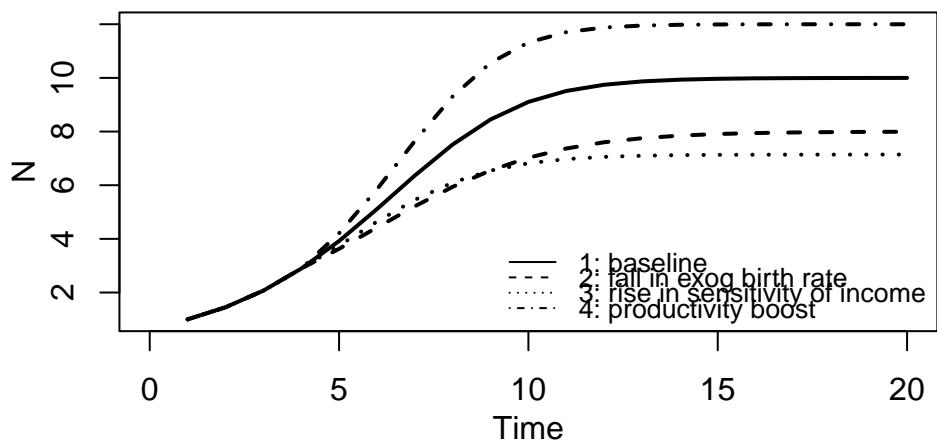


Figure 3 displays population dynamics under the different scenarios described in Table 1. As expected, both preventative and positive checks are effective: a fall in the exogenous component of the birth rate and an increase in the sensitivity of real income slow down population dynamics and lower its steady state value. By contrast, a productivity boost allows for a higher equilibrium level of population.

```
## Population dynamics under different scenarios
plot(N[1, 2:(Tmax+1)], type="l", lwd=2, lty=1, xlim=range(0:(Tmax)), ylim=range(N[4, 2:Tmax])
title(main="Figure 3: Population dynamics under different scenarios", ylab = 'N', xlab = 'T'
lines(N[2, 2:(Tmax+1)], lty=2, lwd=2)
lines(N[3, 2:(Tmax+1)], lty=3, lwd=2)
lines(N[4, 2:(Tmax+1)], lty=4, lwd=2)
legend("bottomright", legend=c("1: baseline", "2: fall in exog birth rate", "3: rise in sen
"4: productivity boost"), lty=1:4, cex=0.8, bty = "n", y.intersp=0.5)
```

Figure 3: Population dynamics under different scenarios



Python code

```
### Plots (here for population and real income only)

import matplotlib.pyplot as plt

# Set maximum period for plots
Tmax = 20

## Baseline
# Population and real income
fig, ax1 = plt.subplots()
ax1.plot(N[0, 2:(Tmax+1)], linestyle='solid', label='N', linewidth=0.8, color="black")
ax1.set_xlabel('Time')
ax1.set_ylabel('N', rotation=0)
ax2 = ax1.twinx()
ax2.plot(Y[0, 2:Tmax], linestyle='dashed', label='Y', linewidth=0.8, color="black")
ax2.set_ylabel('Y', rotation=0)
lines, labels = ax1.get_legend_handles_labels() #collect legend in one box
lines2, labels2 = ax2.get_legend_handles_labels()
ax2.legend(lines + lines2, labels + labels2, loc=5)
plt.show()
```

12.4 Directed graph

Another perspective on the model's properties is provided by its directed graph. A directed graph consists of a set of nodes that represent the variables of the model. Nodes are connected by directed edges. An edge directed from a node x_1 to node x_2 indicates a causal impact of x_1 on x_2 .

```
## Create directed graph
# Construct auxiliary Jacobian matrix for 6 variables: N, B, D, Y, b0, a0,
# where non-zero elements in regular Jacobian are set to 1 and zero elements are unchanged

M_mat=matrix(c(0,1,1,0,0,0,
              1,0,0,1,1,0,
              1,0,0,1,0,0,
              1,0,0,0,0,1,
              0,0,0,0,0,0,
```

```

0,0,0,0,0,0), 6, 6, byrow=TRUE)

# Create adjacency matrix from transpose of auxiliary Jacobian and add column names
A_mat=t(M_mat)

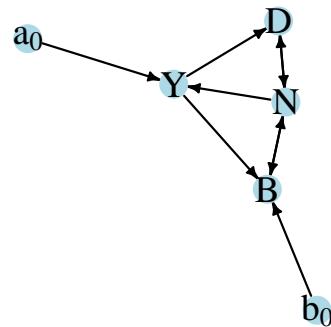
# Create directed graph from adjacency matrix
library(igraph)
dg=graph_from_adjacency_matrix(A_mat, mode="directed", weighted= NULL)

# Define node labels
V(dg)$name=c("N", "B", "D", "Y", expression(b[0]), expression(a[0]))

# Plot directed graph matrix
plot(dg, main="Figure 4: Directed graph of Malthusian model", vertex.size=20, vertex.color=
    vertex.label.color="black", edge.arrow.size=0.3, edge.width=1.1, edge.size=1.2,
    edge.arrow.width=1.2, edge.color="black", vertex.label.cex=1.2,
    vertex.frame.color="NA", margin=-0.08)

```

Figure 4: Directed graph of Malthusian model



 Python code

```
import networkx as nx

#Construct auxiliary Jacobian matrix for 6 variables: N, B, D, Y, b0, a0
M_mat = np.array([[0,1,1,0,0,0],
                  [1,0,0,1,1,0],
                  [1,0,0,1,0,0],
                  [1,0,0,0,0,1],
                  [0,0,0,0,0,0],
                  [0,0,0,0,0,0]])

# Create adjacency matrix from transpose of auxiliary Jacobian and add column names
A_mat = M_mat.transpose()

# Create the graph from the adjacency matrix
G = nx.DiGraph(A_mat)

# Define node labels
nodelabs = {0: "N", 1: "B", 2: "D", 3: "Y", 4: "$b_0$", 5: "$a_0$"}

# Plot the graph
pos = nx.spring_layout(G)
nx.draw_networkx(G, pos, node_size=500, node_color="lightblue",
                 edge_color="black", width=1.2, arrowsize=10,
                 arrowstyle='->', font_size=12, font_color="black",
                 with_labels=True, labels=nodelabs)

plt.axis("off")
plt.title("Figure 4: Directed graph of Malthusian model")
plt.show()
```

In Figure 4, it can be seen that the exogenous birth rate (b_0) and productivity (a_0) are exogenous variables that impact births and income, respectively. Births, deaths, employment and income are endogenous and form a closed loop (or cycle) within the system. Births and deaths affect the population size (with simultaneous feedback from population to births and deaths), and the population affects income. Income, in turn, feeds back into population size.

12.5 Analytical discussion

To find the steady state solution for N , substitute Equation 12.2 - Equation 12.4 into Equation 12.1 and collect terms::

$$N_t = N_{t-1}[1 + b_0 - d_0 + a_0(b_1 + d_1)] - N_{t-1}^2[a_1(b_1 + d_1)]. \quad (12.5)$$

Subtract N_{t-1} and divide through by N_{t-1} :

$$\frac{N_t - N_{t-1}}{N_{t-1}} = [b_0 - d_0 + a_0(b_1 + d_1)] - N_{t-1}[a_1(b_1 + d_1)].$$

Set $\frac{N_t - N_{t-1}}{N_{t-1}} = 0$ and solve for N_t to find the non-trivial steady state:²

$$N^* = \frac{b_0 - d_0 + a_0(b_1 + d_1)}{a_1(b_1 + d_1)}.$$

Substitution of N^* into Equation 12.4 and simplifying yields:

$$Y^* = \frac{d_0 - b_0}{b_1 + d_1}.$$

Finally, to assess the dynamic stability of the model, differentiate Equation 12.5 with respect to N_{t-1} :

$$\frac{\partial N_t}{\partial N_{t-1}} = 1 + b_0 - d_0 + a_0(b_1 + d_1) - 2N_{t-1}[a_1(b_1 + d_1)].$$

Due to the non-linearity of the model, stability can only be assessed locally around the steady state. To do this, substitute the steady state solution and simplify:

$$\frac{\partial N_t}{\partial N_{t-1}} = 1 - b_0 + d_0 - a_0(b_1 + d_1).$$

From this, we can conclude that the steady state is stable iff:

$$|1 - b_0 + d_0 - a_0(b_1 + d_1)| < 1.$$

We can verify these analytical solutions by comparing them with the results from the numerical solution:

²A trivial steady state is at $N^* = 0$.

```

# Calculate analytical equilibrium solutions
for (i in 1:S){
  N_eq[i]=(b0[i,T]-d0+a0[i,T]*(b1+d1))/(a1[i,T]*(b1+d1))
  Y_eq[i]=(d0-b0[i,T])/(b1+d1)
  B_eq[i]=(b0[i,T] + b1*Y_eq[i])*N_eq[i]
  D_eq[i]=(d0 - d1*Y_eq[i])*N_eq[i]
}

# Compare with numerical solutions (here for the example of Y, scenario1)
Y_eq[1]

```

[1] 2

Y[1,T]

[1] 2

```

# Check stability condition for all scenarios
for (i in 1:S){
  print(paste0("Scenario ", i, ":"))
  print(abs(1-b0[i,T]+d0-a0[i,T]*(b1+d1)) < 1)
}

```

[1] "Scenario 1:"
[1] TRUE
[1] "Scenario 2:"
[1] TRUE
[1] "Scenario 3:"
[1] TRUE
[1] "Scenario 4:"
[1] TRUE

 Python code

```
# Calculate analytical equilibrium solutions
for i in range(S):
    N_eq[i] = (b0[i, T-1] - d0 + a0[i, T-1] * (b1 + d1)) / (a1[i, T-1] * (b1 + d1))
    Y_eq[i] = (d0 - b0[i, T-1]) / (b1 + d1)
    B_eq[i] = (b0[i, T-1] + b1 * Y_eq[i]) * N_eq[i]
    D_eq[i] = (d0 - d1 * Y_eq[i]) * N_eq[i]

# Compare with numerical solutions (here for the example of Y, scenario1)
print(Y_eq[0])
print(Y[0,T-1])

# Check stability condition for all scenarios
for i in range(S):
    print(f"Scenario {i + 1}:")
    print(abs(1 - b0[i, T-1] + d0 - a0[i, T-1] * (b1 + d1)) < 1)
```

12.6 References

13 A Ricardian One-Sector Model

13.1 Overview

This model captures some key feature of David Ricardo's theory of growth and distribution as developed in his 1817 book [On the Principles of Political Economy and Taxation](#). The model revolves around the determination of real wages, rents, and profits, and how profitability in turn drives capital accumulation.¹ It assumes a corn economy with a single good (corn) that serves both as an investment and consumption good.² Corn production is subject to diminishing marginal returns. Real wages are driven down to a subsistence level and rent is a differential surplus landowners gain based on the fertility of their land relative to the marginal plot of land (the plot of land where fertility is lowest and no rent is earned). Profits are a residual. As employment increases and more land is utilised, marginal productivity falls and differential rents increase. As a result, profits are driven down and capital accumulation comes to a halt. A stationary state is reached. Landowners are the main beneficiaries of this process. The model is adapted from Pasinetti (1960).

13.2 The Model

The following equations describe the model:

$$Y_t = AN_t^a \quad (13.1)$$

$$W_t = K_t \quad (13.2)$$

$$w_t = W_t/N_t \quad (13.3)$$

$$MPL_t = \frac{\partial Y_t}{\partial N_t} = aAN_t^{a-1} \quad (13.4)$$

¹See chapter 2 of Foley (2006) for an excellent introduction.

²See Chapter 14 for a two-sector extension of the model.

$$R_t = Y_t - N_t MPL_t \quad (13.5)$$

$$P_t = Y_t - R_t - N_t w_t \quad (13.6)$$

$$K_t = K_{t-1} + \beta P_{t-1} \quad (13.7)$$

$$N_t = N_{t-1} + \gamma(w_{t-1} - w^S) \quad (13.8)$$

where Y_t , A , N_t , W_t , K_t , w_t , P_t , and w^S are real output (measured in units of corn), productivity, employment, the real wage bill (or wage fund), the capital stock, the real wage rate, the marginal product of labour, rents, profits, and the subsistence wage, respectively.

Equation 13.1 is the production function with $\alpha \in (0, 1)$, i.e. exhibiting diminishing marginal returns to labour.³ By Equation 13.2, the wage fund is defined as the capital stock of this model (reflecting the fact that the production of corn only involves labour). Equation 13.3 defines the real wage rate. Equation 13.3 specifies the marginal product of labour. Equation 13.5 captures the determination of (differential) rents as a negative function of the marginal product of labour.⁴ Thus, the lower the productivity on the marginal land, the higher the rents. In Equation 13.6, profits are determined residually. Capital accumulation in Equation 13.7 is driven by the reinvestment of profits (with β determining the proportion of profits that are reinvested). Finally, Equation 13.8 specifies population dynamics, whereby the population increases whenever the actual real wage is above the subsistence wage, echoing the Malthusian population mechanism.

13.3 Simulation

13.3.1 Parameterisation

Table 1 reports the parameterisation and initial values used in the simulation. In line with the Classical tradition, it will be assumed that all profits are reinvested, i.e. $\beta = 1$. Besides a baseline (labelled as scenario 1), three further scenarios will be considered. Scenarios 2 and 3 model two different forms of technological change: an increase in the productivity parameter A and an increase in the elasticity of output with respect to labour (a). Scenario 4 considers a

³Pasinetti (1960) specifies a generic function $f(N_t)$ with $f(0) \geq 0$, $f'(0) > w^*$, and $f''(N_t) < 0$. Equation 13.1 satisfies these conditions.

⁴Equation 13.5 is based on the definition of total rent as the sum of the net gains of the non-marginal landowners. See Pasinetti (1960) for a formal derivation.

higher subsistence wage (w^S). In all scenarios the population/employment is initialised below its equilibrium value.

Table 1: Parameterisation

Scenario	A	a	w^S
1: baseline	2	0.7	0.5
2: productivity boost I (A)	3	0.7	0.5
3: productivity boost II (a)	2	0.75	0.5
4: higher subsistence wage (w^S)	2	0.7	0.7

13.3.2 Simulation code

```
# Clear the environment
rm(list=ls(all=TRUE))

# Set number of periods
Q=500

# Set number of scenarios (including baseline)
S=4

# Set period in which shock/shift will occur
s=20

# Create (S x Q)-matrices that will contain the simulated data
Y=matrix(data=1,nrow=S,ncol=Q) # Income/output
R=matrix(data=1,nrow=S,ncol=Q) # Rent
P=matrix(data=1,nrow=S,ncol=Q) # Profits
N=matrix(data=1,nrow=S,ncol=Q) # employment
w=matrix(data=1,nrow=S,ncol=Q) # real wage
K=matrix(data=1,nrow=S,ncol=Q) # capital stock
MPL=matrix(data=1,nrow=S,ncol=Q) # marginal product of labour
W=matrix(data=1,nrow=S,ncol=Q) # wage bill
N_eq=vector(length=S)          # equilibrium population
K_eq=vector(length=S)          # equilibrium capital

# Set baseline parameter values
A=matrix(data=2,nrow=S,ncol=Q) # productivity
a=matrix(data=0.7,nrow=S,ncol=Q) # labour elasticity of output
beta=1 # Sensitivity of investment with respect to profits
```

```

gamma=5 # adjustment speed of population
wS=matrix(data=0.5,nrow=S,ncol=Q) # subsistence wage rate

# Set parameter values for different scenarios
A[2,s:Q]=3 # scenario 2: productivity boost I
a[3,s:Q]=0.75 # scenario 3: productivity boost II
wS[4,s:Q]=0.6 # scenario 4: increase in subsistence wage

# Initialise variables such that employment and the capital stock are below the equilibrium
N[,1]=1
K[,1]=1
Y[,1]=A[,1]*N[,1]^(a[,1])
MPL[,1]=a[,1]*A[,1]*(N[,1]^(a[,1]-1))
w[,1]=wS[,1]

# Simulate the model by looping over Q time periods for S different scenarios
for (i in 1:S){

  for (t in 2:Q){

    for (iterations in 1:1000){ # run the model 1000-times in each period

      #Model equations

      #(1) Output
      Y[i,t] = A[i,t]*N[i,t]^(a[i,t])

      #(2) Wage bill
      W[i,t] = K[i,t]

      #(3) Real wage rate
      w[i,t] = W[i,t]/N[i,t]

      #(4) Marginal product of labour
      MPL[i,t] = a[i,t]*A[i,t]*(N[i,t]^(a[i,t]-1))

      #(5) Rents
      R[i,t] = Y[i,t] - N[i,t]*MPL[i,t]

      #(6) Profits
      P[i,t] = Y[i,t] - R[i,t] - N[i,t]*w[i,t]
    }
  }
}

```

```
# (7) Capital accumulation
K[i,t] = K[i,t-1] + beta*P[i,t-1]

#(8) Employment/population dynamics
N[i,t] = N[i,t-1] + gamma*(w[i,t-1] - wS[i,t-1])

} # close iterations loop
} # close time loop
} # close scenario loop
```

 Python code

```

import numpy as np

# Set number of periods
Q = 500

# Set number of scenarios (including baseline)
S = 4

# Set period in which shock/shift will occur
s = 20

# Create (S x Q)-matrices that will contain the simulated data
Y = np.ones((S, Q)) # Income/output
R = np.ones((S, Q)) # Rent
P = np.ones((S, Q)) # Profits
N = np.ones((S, Q)) # employment
w = np.ones((S, Q)) # real wage
K = np.ones((S, Q)) # capital stock
MPL = np.ones((S, Q)) # marginal product of labour
W = np.ones((S, Q)) # wage bill

# Set baseline parameter values
A = np.ones((S, Q)) * 2 # productivity
a = np.ones((S, Q)) * 0.7 # labour elasticity of output
beta = 1 # Sensitivity of investment with respect to profits
gamma = 5 # adjustment speed of population
wS = np.ones((S, Q)) * 0.5 # subsistence wage rate

# Set parameter values for different scenarios
A[1, s:Q] = 3 # scenario 2: productivity boost I
a[2, s:Q] = 0.75 # scenario 3: productivity boost II
wS[3, s:Q] = 0.6 # scenario 4: increase in subsistence wage

# Initialise variables such that employment and the capital stock are below the equilibrium
N[:, 0] = 1
K[:, 0] = 1
Y[:, 0] = A[:, 0] * N[:, 0]**(a[:, 0])
MPL[:, 0] = a[:, 0] * A[:, 0] * (N[:, 0]**(a[:, 0] - 1))
w[:, 0] = wS[:, 0]

# Simulate the model by looping over Q time periods for S different scenarios
for i in range(S):
    for t in range(1, Q):
        for iterations in range(1000): # run the model 1000 times in each period
            # Model equations          245
            # (1) Output
            Y[i, t] = A[i, t] * N[i, t]**(a[i, t])
            # (2) Wage bill
            W[i, t] = K[i, t]
            # (3) Real wage rate
            w[i, t] = W[i, t] / N[i, t]
            # (4) Marginal product of labour

```

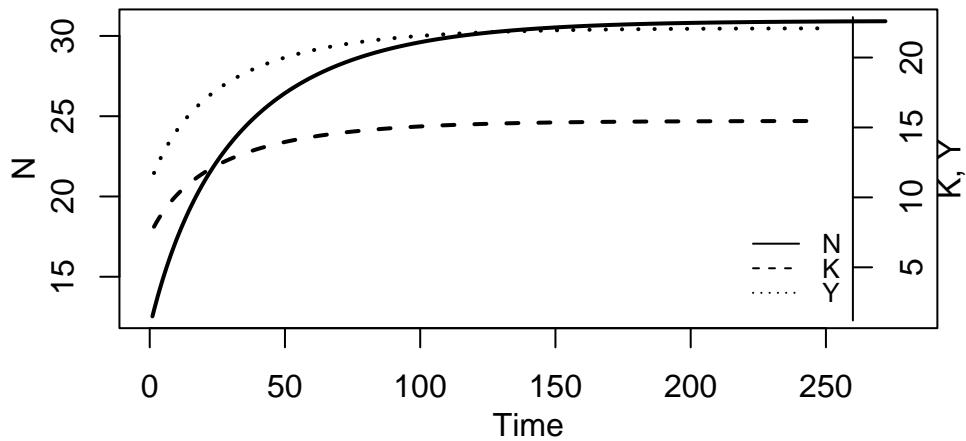
13.3.3 Plots

Figure 1 displays employment, capital accumulation, and income for the baseline scenario. Starting from a below-equilibrium level of population, the economy grows in terms of output, capital, and employment but then approaches what Ricardo famously called a ‘stationary state’. Figure 2 shows that during the adjustment phase, the MPL declines reflecting diminishing marginal returns in the production of corn. This captures the idea that a growing economy will have to utilise less fertile lands. The real wage is initially below the MPL, allowing for profits. Over time, the MPL and actual real wage converge to the exogenously given subsistence wage. Figure 3 shows that profits initially increase but are then squeezed to zero as differential rents increase.

```
# Set start and end periods for plots
Tmax=280
Tmin =10

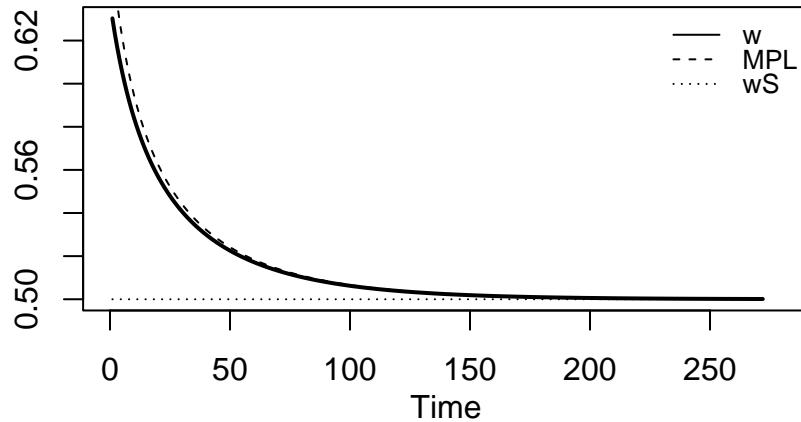
## Baseline
#Employment, capital accumulation, and income
plot(N[1, Tmin:(Tmax+1)],type="l", lwd=2, lty=1, xlim=range(0:(Tmax)), ylab = '', xlab = ''
      title(main="Figure 1: Employment, capital accumulation, and income",ylab = 'N', xlab = 'Ti
      par(mar = c(5, 4, 4, 4) + 0.3)
      par(new = TRUE)
plot(K[1, Tmin:Tmax],type="l", col=1, lwd=2, lty=2, font.main=1,cex.main=1,ylab = '', axes
      xlab = '',ylim = range(Y[1, 2:(Tmax+1)]),cex.axis=1,cex.lab=0.75)
lines(Y[1, Tmin:(Tmax+1)],lty=3, lwd=2)
axis(side = 4, at = pretty(range(Y[1, 2:(Tmax+1)])))
mtext("K, Y", side = 4, line = 2)
legend("bottomright", legend=c("N", "K", "Y"),
      lty=1:3, cex=0.8, bty = "n", y.intersp=0.8)
```

Figure 1: Employment, capital accumulation, and income



```
# Real wage, subsistence wage, and MPL
plot(w[1, Tmin:(Tmax+1)], type="l", col=1, lwd=2, lty=1, xlim=range(0:(Tmax)), xlab="", ylab="Real wage, subsistence wage, and MPL", main="Figure 2: Real wage, marginal product of labour, and subsistence wage", xlab = "Time", ylab = "Real wage, subsistence wage, and MPL")
title(main="Figure 2: Real wage, marginal product of labour, and subsistence wage", xlab = "Time", ylab = "Real wage, subsistence wage, and MPL")
lines(MPL[1, Tmin:Tmax], lty=2)
lines(wS[1, Tmin:Tmax], lty=3)
legend("topright", legend=c("w", "MPL", "wS"),
       lty=1:3, cex=0.8, bty = "n", y.intersp=0.8)
```

➲ 2: Real wage, marginal product of labour, and subsistence



```
# Profits and Rents
plot(P[1, Tmin:(Tmax+1)], type="l", col=1, lwd=2, lty=1, xlim=range(0:(Tmax)), xlab="", yla
title(main="Figure 3: Profits and rents", xlab = 'Time', cex=0.8, line=2)
par(mar = c(5, 4, 4, 4) + 0.3)
par(new = TRUE)
plot(R[1, Tmin:(Tmax+1)], type="l", col=1, lwd=2, lty=2, xlim=range(0:(Tmax)), xlab="", yla
      ylim=range(R[1, 3:Tmax]), axes=FALSE)
axis(side = 4, at = pretty(range(R[1, Tmin:(Tmax+1)])))
mtext("R", side = 4, line = 2)
legend("right", legend=c("P", "R"), lty=1:2, cex=0.8, bty = "n", y.intersp=0.8)
```

Figure 3: Profits and rents

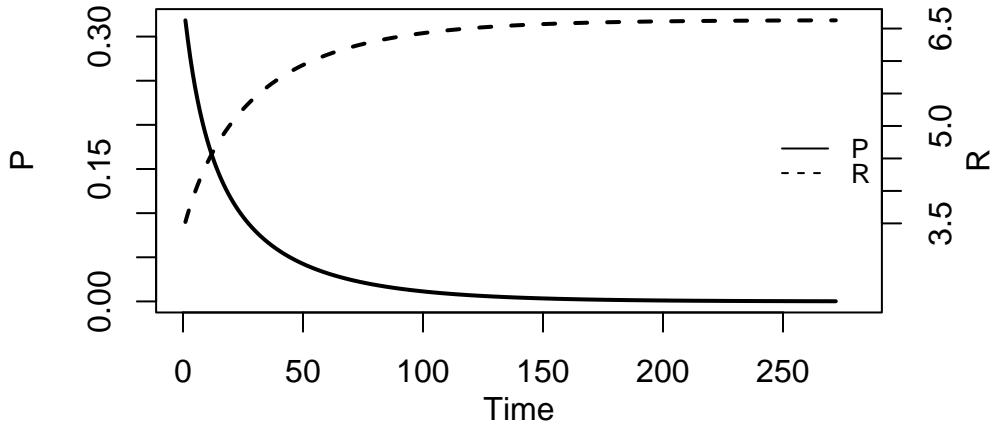
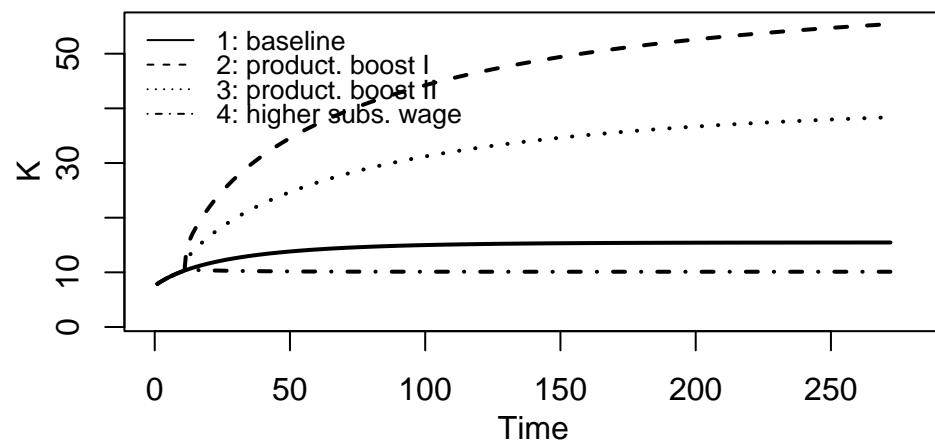


Figure 4 displays capital accumulation under the different scenarios described in Table 1. As expected, both forms of technical change boost both the speed of capital accumulation and the equilibrium level of capital. An increase in the subsistence wage reduces the pace of capital accumulation and leads to a lower equilibrium level of capital.

```
## Different scenarios
# Capital accumulation
plot(K[1, Tmin:(Tmax+1)], type="l", lwd=2, lty=1, xlim=range(0:(Tmax)), ylim=range(K[1, 2:Tmax+1]))
title(main="Figure 4: Capital accumulation under different scenarios", ylab = 'K', xlab = 'Time')
lines(K[2, Tmin:(Tmax+1)], lty=2, lwd=2)
lines(K[3, Tmin:(Tmax+1)], lty=3, lwd=2)
lines(K[4, Tmin:(Tmax+1)], lty=4, lwd=2)
legend("topleft", legend=c("1: baseline", "2: product. boost I", "3: product. boost II", "4: high wage"))
```

Figure 4: Capital accumulation under different scenarios



Python code

```
## Plots (here for employment, capital accumulation, and income only)

import matplotlib.pyplot as plt

# Set start and end periods for plots
Tmax = 280
Tmin = 10

# Baseline
# Employment, capital accumulation, and income
fig, ax1 = plt.subplots()
ax1.plot(N[0, 2:(Tmax+1)], linestyle='solid', label='N', linewidth=0.8, color="black")

ax1.set_xlabel('Time')
ax1.set_ylabel('N', rotation=0)
ax2 = ax1.twinx()
ax2.plot(K[0, 2:Tmax], linestyle='dashed', label='K', linewidth=0.8, color="black")
ax2.plot(Y[0, 2:Tmax], linestyle='dotted', label='Y', linewidth=0.8, color="black")
ax2.set_ylabel('Y, K', rotation=0)
lines, labels = ax1.get_legend_handles_labels() #collect legend in one box
lines2, labels2 = ax2.get_legend_handles_labels()
ax2.legend(lines + lines2, labels + labels2, loc=5)
plt.title("Figure 1: Employment, capital accumulation, and income")
plt.show()
```

13.4 Directed graph

Another perspective on the model's properties is provided by its directed graph. A directed graph consists of a set of nodes that represent the variables of the model. Nodes are connected by directed edges. An edge directed from a node x_1 to node x_2 indicates a causal impact of x_1 on x_2 .

```
## Create directed graph
# Construct auxiliary Jacobian matrix for 10 variables: Y W w MPL R P K N A wS,
# where non-zero elements in regular Jacobian are set to 1 and zero elements are unchanged
M_mat=matrix(c(0,0,0,0,0,0,0,1,1,0,
              0,0,0,0,0,0,1,0,0,0,
```

```

0,1,0,0,1,0,0,1,0,0,
0,0,0,0,0,0,0,1,1,0,
1,0,0,1,0,0,0,1,0,0,
1,0,1,0,1,0,0,1,0,0,
0,0,0,0,0,1,0,0,0,0,
0,0,1,0,0,0,0,0,0,1,
0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0), 10,10, byrow=TRUE)

# Create adjacency matrix from transpose of auxiliary Jacobian and add column names
A_mat=t(M_mat)

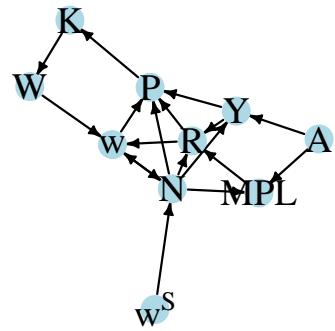
# Create and plot directed graph from adjacency matrix
library(igraph)
dg= graph_from_adjacency_matrix(A_mat, mode="directed", weighted= NULL)

# Define node labels
V(dg)$name=c("Y", "W", "w", "MPL", "R", "P", "K", "N", "A", expression(w^S))

# Plot directed graph
plot(dg, main="Figure 5: Directed graph of Ricardian One-Sector Model", vertex.size=20, ve
    vertex.label.color="black", edge.arrow.size=0.3, edge.width=1.1, edge.size=1.2,
    edge.arrow.width=1.2, edge.color="black", vertex.label.cex=1.2,
    vertex.frame.color="NA", margin=-0.08)

```

Figure 5: Directed graph of Ricardian One-Sector Model



 Python code

```
# Create directed graph

import networkx as nx

# Construct auxiliary Jacobian matrix for 10 variables: Y W w MPL R P K N A wS,
# where non-zero elements in regular Jacobian are set to 1 and zero elements are unchanged
M_mat = np.array([
    [0, 0, 0, 0, 0, 0, 0, 1, 1, 0],
    [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
    [0, 1, 0, 0, 1, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 1, 1, 0],
    [1, 0, 0, 1, 0, 0, 0, 1, 0, 0],
    [1, 0, 1, 0, 1, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
    [0, 0, 1, 0, 0, 0, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
])

# Create adjacency matrix from transpose of auxiliary Jacobian and add column names
A_mat = M_mat.transpose()

# Create the graph from the adjacency matrix
G = nx.DiGraph(A_mat)

# Define node labels
nodelabs = {0: 'Y', 1: 'W', 2: 'w', 3: 'MPL', 4: 'R', 5: 'P', 6: 'K', 7: 'N', 8: 'A', 9: 'wS'}

# Plot the graph
pos = nx.spring_layout(G)
nx.draw_networkx(G, pos, node_size=500, node_color="lightblue",
                  edge_color="black", width=1.2, arrowsize=10,
                  arrowstyle='->', font_size=12, font_color="black",
                  with_labels=True, labels=nodelabs)
plt.axis("off")
plt.title("Figure 5: Directed graph of Ricardian One-Sector Model")
plt.show()
```

In Figure 5, it can be seen that productivity (A) and the subsistence wage (w^S) are the key

exogenous variables that impact income and the marginal product of labour, and population dynamics, respectively. Most other variables are endogenous and form a closed loop (or cycle) within the system. Profits are a residual. The directed graph illustrates the supply-driven nature of the model, where productivity determines employment and distribution, which in turn feed back into income and capital accumulation. At the same time, income distribution has an exogenous element in the form of the subsistence wage, which feeds into the system.

13.5 Analytical discussion

To analyse the dynamics, combine Equation 13.1 to Equation 13.6 and substitute into Equation 13.7. Further use Equation 13.2 and Equation 13.3 in Equation 13.8. This yields the two-dimensional dynamic system in K_t and N_t :

$$K_t = (1 - \beta)K_{t-1} + \beta(aAN_{t-1}^a)$$

$$N_t = N_{t-1} + \gamma \left(\frac{K_{t-1}}{N_{t-1}} - w^S \right)$$

The Jacobian matrix is given by:

$$J(K, N) = \begin{bmatrix} 1 - \beta & \beta a^2 A N^{\alpha-1} \\ \frac{\gamma}{N} & 1 - \frac{\gamma K}{N^2} \end{bmatrix}.$$

From equations Equation 13.7 and Equation 13.8, it can readily be seen that an equilibrium is reached when

$$P^* = 0$$

and

$$w^* = w^S.$$

Using $P^* = 0$ with Equation 13.5 and Equation 13.6, yields $w^* = w^S = MPL$. Thus, in equilibrium, profits are zero, and the real wage is equal to the MPL and the subsistence wage. Setting $K_t = K_{t-1}$ and $N_t = N_{t-1}$, we can further derive:

$$K^* = aA \left(\frac{w^S}{aA} \right)^{-\frac{a}{1-a}}$$

and

$$N^* = \left(\frac{w^S}{aA} \right)^{-\frac{1}{1-a}}$$

With this, we can evaluate the Jacobian at the steady state:

$$J(K^*, N^*) = \begin{bmatrix} 1 - \beta & \beta a w^S \\ \gamma \left(\frac{w^S}{aA}\right)^{\frac{1}{1-a}} & 1 - \gamma a A \left(\frac{w^S}{aA}\right)^{\frac{2-a}{1-a}} \end{bmatrix}.$$

For the system to be stable, both eigenvalues of the Jacobian need to be inside the unit circle. This requires the following three conditions to hold:

$$\begin{aligned} 1 + \text{tr}(J) + \det(J) &> 0 \\ 1 + \text{tr}(J) - \det(J) &> 0 \\ 1 - \det(J) &> 0, \end{aligned}$$

where $\text{tr}(J)$ is the trace and $\det(J)$ is the determinant of the Jacobian.

Let us consider the Classical case where $\beta = 1$, i.e. all profits are reinvested. Then we have

$$\det(J) = -aw^S\gamma \left(\frac{w^S}{aA}\right)^{\frac{1}{1-a}} < 0,$$

so that the third condition is always satisfied and it is the first one that is binding. The first condition then becomes

$$2 - \gamma a \left[A \left(\frac{w^S}{aA}\right)^{\frac{2-a}{1-a}} + w^S \left(\frac{w^S}{aA}\right)^{\frac{1}{1-a}} \right] > 0$$

We can check the analytical solutions and stability conditions numerically:

```
# Calculate equilibrium solutions
for (i in 1:S){
  N_eq[i]=(wS[i,Q]/(a[i,Q]*A[i,Q]))^(-1/(1-a[i,Q]))
  K_eq[i]=a[i,Q]*A[i,Q]*(wS[i,Q]/(a[i,Q]*A[i,Q]))^(-a[i,Q]/(1-a[i,Q]))
}

# Compare with numerical solutions (here only for baseline)
N_eq[1]
```

[1] 30.94046

N[1,Q]

[1] 30.94031

```
K_eq[1]
```

```
[1] 15.47023
```

```
K[1,Q]
```

```
[1] 15.47018
```

```
### Examine model properties (here for the baseline scenario only)
```

```
# Construct Jacobian matrix at the equilibrium
```

```
J=matrix(c(1-beta, beta*a[1,Q]*wS[1,Q],  
         beta*(wS[1,Q]/(a[1,Q]*A[1,Q]))^(1/(1-a[1,Q])),  
         1-gamma*a[1,Q]*A[1,Q]*(wS[1,Q]/(a[1,Q]*A[1,Q]))^((2-a[1,Q])/(1-a[1,Q])), 2, 2,  
  
         # Obtain eigenvalues  
         ev=eigen(J)  
         values = ev$values)
```

```
[1] 0.93134557 -0.01214592
```

```
# Obtain determinant and trace  
det=det(J)      # determinant  
tr=sum(diag(J)) # trace
```

```
#Check general stability conditions  
print(1+tr+det>0)
```

```
[1] TRUE
```

```
print(1-tr-det>0)
```

```
[1] TRUE
```

```
print(1-det>0)
```

```
[1] TRUE
```

```
# Check specific stability condition for the case beta=1
for (i in 1:S){
  print(paste0("Scenario ", i, ":"))
  print(2-gamma*a[i,Q]*(A[i,Q]*(wS[i,Q]/(a[i,Q]*A[i,Q]))^((2-a[i,Q])/(1-a[i,Q]))
    + wS[i,Q]*(wS[i,Q]/(a[i,Q]*A[i,Q]))^(1/(1-a[i,Q])))>0)
}

[1] "Scenario 1:"
[1] TRUE
[1] "Scenario 2:"
[1] TRUE
[1] "Scenario 3:"
[1] TRUE
[1] "Scenario 4:"
[1] TRUE
```

 Python code

```

# Initialize arrays for equilibrium solutions
N_eq = np.zeros(S)
K_eq = np.zeros(S)

# Calculate equilibrium solutions
for i in range(S):
    N_eq[i] = (wS[i, Q-1] / (a[i, Q-1] * A[i, Q-1])) ** (-1 / (1 - a[i, Q-1]))
    K_eq[i] = a[i, Q-1] * A[i, Q-1] * (wS[i, Q-1] / (a[i, Q-1] * A[i, Q-1])) ** (-a[i, Q-1])

# Compare with numerical solutions (here only for baseline)
N_eq[0]
N[0,Q-1]

# Construct Jacobian matrix at the equilibrium
J = np.array([
    [1 - beta, beta * a[0, Q-1] * wS[0, Q-1]],
    [beta * (wS[0, Q-1] / (a[0, Q-1] * A[0, Q-1])) ** (1 / (1 - a[0, Q-1])),
     1 - gamma * a[0, Q-1] * A[0, Q-1] * (wS[0, Q-1] / (a[0, Q-1] * A[0, Q-1])) ** ((2 - a[0, Q-1]) / (1 - a[0, Q-1]))]
])

# Obtain eigenvalues
eigenvalues, eigenvectors = np.linalg.eig(J)
print(eigenvalues)

# Obtain determinant and trace
det = np.linalg.det(J)
tr = np.trace(J)

# Check general stability conditions
print(1+tr+det>0)
print(1-tr+det>0)
print(1-det>0)

# Check specific stability condition for the case beta=1
for i in range(S):
    print(f"Scenario {i + 1}:")
    print(2 - gamma * a[i, Q-1] * (
        A[i, Q-1] * (wS[i, Q-1] / (a[i, Q-1] * A[i, Q-1])) ** ((2 - a[i, Q-1]) / (1 - a[i, Q-1])) -
        wS[i, Q-1] * (wS[i, Q-1] / (a[i, Q-1] * A[i, Q-1])) ** (1 / (1 - a[i, Q-1])))
    ) > 0)

```

13.6 References

14 A Ricardian Two-Sector Model

This model captures some key feature of David Ricardo's theory of growth and distribution as developed in his 1817 book [On the Principles of Political Economy and Taxation](#). The model revolves around the determination of real wages, rents, and profits, and how profitability in turn drives capital accumulation.¹ It assumes an economy with two sectors: an agricultural sector producing corn subject to diminishing marginal returns and a luxury good sector with constant marginal returns.² Prices are determined by the quantity of labour required for production. Rent on the land used for agricultural production is a differential surplus landowners gain based on the fertility of their land relative to the marginal plot of land (the plot of land where fertility is lowest and no rent is earned). Real wages are determined by the subsistence level in the long run. Profits in agriculture are a residual and set the economy-wide profit rate. As employment increases and more land is utilised, marginal productivity in agriculture falls and differential rents increase. As a result, profits are driven down to zero and capital accumulation comes to a halt. A 'stationary state' is reached. Landowners are the main beneficiaries of this process. The model is adapted from Pasinetti (1960).

14.1 The Model

The following equations describe the model:

$$Y_t = AN_{1t}^{a_1} \quad (14.1)$$

$$MPL_t = \frac{\partial Y_{1t}}{\partial N_{1t}} = a_1 AN_{1t}^{a_1-1} \quad (14.2)$$

$$N_{1t} = N_t - N_{2t} \quad (14.3)$$

$$W_t = K_t \quad (14.4)$$

¹See chapter 2 of Foley (2006) for an excellent introduction.

²See Chapter 13 for a simpler one-sector version of the model,

$$w_t = \frac{W_t}{N_t} \quad (14.5)$$

$$R_t = Y_{1t} - N_{1t}MPL_t \quad (14.6)$$

$$P_{1t} = Y_{1t} - R_t - N_{1t}w_t \quad (14.7)$$

$$p_{1t} = \frac{1}{MPL_t} \quad (14.8)$$

$$Y_{2t} = \left(\frac{p_{1t}}{p_2} \right) R_t \quad (14.9)$$

$$N_{2t} = \frac{Y_{2t}}{a_2} \quad (14.10)$$

$$p_2 = \frac{1}{a_2} \quad (14.11)$$

$$P_{2t} = Y_{2t} - \left(\frac{p_{1t}}{p_2} \right) N_{2t}w_t \quad (14.12)$$

$$P_t = p_{1t}Y_{1t} + p_2Y_{2t} - p_{1t}R_t - p_{1t}W_t \quad (14.13)$$

$$K_t = K_{t-1} + g \left(\frac{P_{t-1}}{p_{1t-1}} \right) \quad (14.14)$$

$$N_t = N_{t-1} + \gamma(w_{t-1} - w^S) \quad (14.15)$$

where Y_t , A , N_t , W_t , K_t , w_t , Y_t , MPL_t , R_t , P_t , p , and w^S are real output, productivity, employment, the real wage bill (or wage fund), the capital stock (in terms of corn), the real wage rate (in terms of corn), the marginal product of labour (in the corn sector), rents, profits, prices, and the subsistence wage, respectively. The subscripts 1 and 2 denote the corn, i.e. agricultural, sector and the luxury goods sectors, respectively.

Equation 14.1 is the production function with $\alpha \in (0, 1)$, i.e. exhibiting diminishing marginal returns to labour.³ By Equation 14.4, the wage fund is defined as the capital stock of this model

³Pasinetti (1960) specifies a generic function $f(N_t)$ with $f(0) \geq 0$, $f'(0) > w^*$, and $f''(N_t) < 0$. Equation 14.1 satisfies these conditions.

(reflecting the fact that the production of corn only involves labour). Equation 14.5 defines the real wage rate. Equation 14.5 specifies the marginal product of labour. By Equation 14.3, employment in agriculture is residually determined after employment in the luxury goods sector has been determined (more on this below) Equation 14.6 captures the determination of (differential) rents as a negative function of the marginal product of labour.⁴ Thus, the lower the productivity on the marginal land, the higher the rents. By Equation 14.7, profits in agriculture are determined residually. Equation 14.8 specifies price determination and captures Ricardo's labour theory of value according to which the value of a good (net of rent) is determined by the quantity of labour required to produce it.⁵

Equation 14.9 specifies that the production of the luxury good is demand determined. Only landlords consume luxuries and they spend all their income (rent) on luxuries.⁶ With production in sector 2 demand determined, employment in sector 2 as given by Equation 14.10 must accommodate based on the production function $Y_{2t} = a_2 N_{2t}$. With employment in sector 2 pinned down in this way and total employment given by the wage fund (Equation 14.4), employment in sector 1 must be the residual (as specified in equation Equation 14.3). From the labour theory of value, $p_2 Y_2 = N_2$ must hold. Together with the production function $Y_{2t} = a_2 N_{2t}$ this yields Equation 14.11 for the price of the luxury good. Note that due to the constant marginal returns in this sector, its price is constant too.⁷ By Equation 14.12, profits in the luxuries sector are determined residually (note that no rent is paid by this sector).

Equation 14.13 specifies total profits (in nominal terms).⁸ Capital accumulation in equation Equation 14.14 is driven by the reinvestment of profits (with β determining the proportion of profits that are reinvested). Finally, Equation 14.15 specifies population dynamics, whereby the population increases whenever the actual real wage is above the subsistence wage, echoing the Malthusian population mechanism.

14.2 Simulation

14.2.1 Parameterisation

Table 1 reports the parameterisation and initial values used in the simulation. In line with the Classical tradition, it will be assumed that all profits are reinvested, i.e. $\beta = 1$. Besides a baseline (labelled as scenario 1), three further scenarios will be considered. Scenarios 2-4 model

⁴Equation 14.6 is based on the definition of total rent as the sum of the net gains of the non-marginal landowners. See Pasinetti (1960) for a formal derivation. Note that by using Equation 14.2, Equation 14.6 can also be written as $R_t = Y_{1t}(1 - a_1)$.

⁵To see this, notice that equation Equation 14.8 can be derived from $p_{t1} Y_{t1} - p_{t1} R_t = N_{t1}$ if combined with equation Equation 14.6.

⁶Output in equation Equation 14.9 is expressed in real terms and can be derived from $p_2 Y_{2t} = p_{1t} R_t$.

⁷The luxury good may therefore serve as Ricardo's 'invariable standard of value' in terms of which the value of all commodities could be expressed.

⁸Note that by combining Equation 14.13 with Equation 14.9, total profits can also be written as $P_t = p_{1t}(Y_{1t} - W_t)$. In other words, total profits are independent of output in sector 2.

three different forms of technological change: an increase in the productivity parameter A (scenario 2), an increase in the elasticity a_1 of agricultural output with respect to labour (scenario 3), and an increase in labour productivity a_2 in the luxury good sector (scenario 4). Scenario 5 considers a higher subsistence wage (w^S). In all scenarios the population/employment is initialised below its equilibrium value.

Table 1: Parameterisation

Scenario	A	a_1	a_2	w^S
1: baseline	2	0.7	0.5	0.5
2: productivity boost I (A)	3	0.7	0.5	0.5
3: productivity boost II (a_1)	2	0.75	0.5	0.5
4: productivity boost III (a_2)	2	0.7	0.55	0.5
5: higher subsistence wage (w^S)	2	0.7	0.5	0.6

14.2.2 Simulation code

```

# Clear the environment
rm(list=ls(all=TRUE))

# Set number of periods
Q=600

# Set number of scenarios (including baseline)
S=5

# Set period in which shock/shift will occur
s=15

# Create (S x Q)-matrices that will contain the simulated data
Y1=matrix(data=1,nrow=S,ncol=Q) # Output in sector 1
Y2=matrix(data=1,nrow=S,ncol=Q) # Output in sector 2
R=matrix(data=1,nrow=S,ncol=Q) # Rent
P=matrix(data=1,nrow=S,ncol=Q) # Qotal profits
P1=matrix(data=1,nrow=S,ncol=Q) # Profits in sector 1
P2=matrix(data=1,nrow=S,ncol=Q) # Profits in sector 2
N=matrix(data=1,nrow=S,ncol=Q) # total employment
N1=matrix(data=1,nrow=S,ncol=Q) # employment in sector 1
N2=matrix(data=1,nrow=S,ncol=Q) # employment in sector 2
w=matrix(data=1,nrow=S,ncol=Q) # real wage
wn=matrix(data=1,nrow=S,ncol=Q) # nominal wage

```

```

K=matrix(data=1,nrow=S,ncol=Q) # capital stock
MPL=matrix(data=1,nrow=S,ncol=Q) # marginal product of labour (in sector 1)
r=matrix(data=1,nrow=S,ncol=Q) # profit rate
p1=matrix(data=1,nrow=S,ncol=Q) # price of good from sector 1
p2=matrix(data=1,nrow=S,ncol=Q) # price of good from sector 2
N_eq=vector(length=S)           # equilibrium population
K_eq=vector(length=S)           # equilibrium capital

test=matrix(data=1,nrow=S,ncol=Q) # price of good from sector 2

# Set baseline parameter values
A=matrix(data=2,nrow=S,ncol=Q) # productivity
a1=matrix(data=0.7,nrow=S,ncol=Q) # labour elasticity of output, sector 1
a2=matrix(data=0.5,nrow=S,ncol=Q) # labour coefficient, sector 2
gamma=5 # adjustment speed of population
beta=1 # Sensitivity of investment with respect to profits
wS=matrix(data=0.5,nrow=S,ncol=Q) # natural wage rate

# Set parameter values for different scenarios
A[2,s:Q]=3      # scenario 2: productivity boost I
a1[3,s:Q]=0.75 # scenario 3: productivity boost II
a2[4,s:Q]=0.55 # scenario 4: productivity boost III
wS[5,s:Q]=0.6  # scenario 5: higher subsistence wage

# Initialise variables such that employment and the capital stock are below the equilibrium
N1[,1]=1
N2[,1]=1
N[,1]=N1[,1]+N2[,1]
K[,1]=1
w[,1]=wS[,1]
Y1[,1]=A[,1]*N1[,1]^(a1[,1])
MPL[,1]=a1[,1]*A[,1]*(N1[,1]^(a1[,1]-1))

# Simulate the model by looping over Q time periods for S different scenarios
for (i in 1:S){

  for (t in 2:Q){

    for (iterations in 1:1000){ # run the model 1000-times in each period

```

```

#Model equations

#(2) Wage bill (omitted for simplicity)
#W[i,t]=K[i,t]

#(3) Output sector 1
Y1[i,t] = A[i,t]*(N1[i,t]^a1[i,t])

#(4) Employment sector 1
N1[i,t] = N[i,t] - N2[i,t]

#(5) Marginal product of labour (sector 1)
MPL[i,t]=a1[i,t]*A[i,t]*(N1[i,t]^(a1[i,t]-1))

#(6) Rent (simplified equation)
R[i,t]= Y1[i,t]*(1-a1[i,t])

#(7) Profits sector 1
P1[i,t] = Y1[i,t] - R[i,t] - N1[i,t]*w[i,t]

#(8) Prices sector 1
p1[i,t] = 1/(MPL[i,t])

#(9) Output sector 2
Y2[i,t]=(p1[i,t]/p2[i,t])*R[i,t]

#(3 Real wage rate
w[i,t] = K[i,t]/N[i,t]

#(10) Employment sector 2
N2[i,t]= Y2[i,t]/a2[i,t]

#(11) Prices
p2[i,t] = 1/a2[i,t]

#(12) Profits sector 2
P2[i,t] = Y2[i,t] - (p1[i,t]/p2[i,t])*N2[i,t]*w[i,t]

#(13) Total profits
P[i,t]=p1[i,t]*Y1[i,t] + p2[i,t]*Y2[i,t] - p1[i,t]*R[i,t] - p1[i,t]*K[i,t]

```

```
#(14) Capital accumulation
K[i,t]= K[i,t-1] + beta*(P[i,t-1]/p1[i,t-1])

#(8) Employment/population dynamics
N[i,t] = N[i,t-1] + gamma*(w[i,t-1] - wS[i,t-1])

} # close iterations loop
} # close time loop
} # close scenarios loop
```

 Python code

```

import numpy as np

# Set number of periods
Q = 600

# Set number of scenarios (including baseline)
S = 5

# Set period in which shock/shift will occur
s = 15

# Create (S x Q)-matrices that will contain the simulated data
Y1 = np.ones((S, Q))
Y2 = np.ones((S, Q))
R = np.ones((S, Q))
P = np.ones((S, Q))
P1 = np.ones((S, Q))
P2 = np.ones((S, Q))
N = np.ones((S, Q))
N1 = np.ones((S, Q))
N2 = np.ones((S, Q))
w = np.ones((S, Q))
wn = np.ones((S, Q))
K = np.ones((S, Q))
MPL = np.ones((S, Q))
r = np.ones((S, Q))
p1 = np.ones((S, Q))
p2 = np.ones((S, Q))
N_eq = np.zeros(S)
K_eq = np.zeros(S)
test = np.ones((S, Q))

# Set baseline parameter values
gamma = 5
beta = 1
A = np.full((S, Q), 2.0)
a1 = np.full((S, Q), 0.7)
a2 = np.full((S, Q), 0.5)
wS = np.full((S, Q), 0.5)

# Set parameter values for different scenarios
A[1, s:] = 3      # scenario 2: productivity boost I
a1[2, s:] = 0.75 # scenario 3: productivity boost II
a2[3, s:] = 0.55 # scenario 4: productivity boost III
wS[4, s:] = 0.6  # scenario 5: higher subsistence wage

```

```

# Initialize variables
N1[:, 0] = 1
N2[:, 0] = 1
N[:, 0] = N1[:, 0] + N2[:, 0]
K[:, 0] = 1
w[:, 0] = wS[:, 0]

```

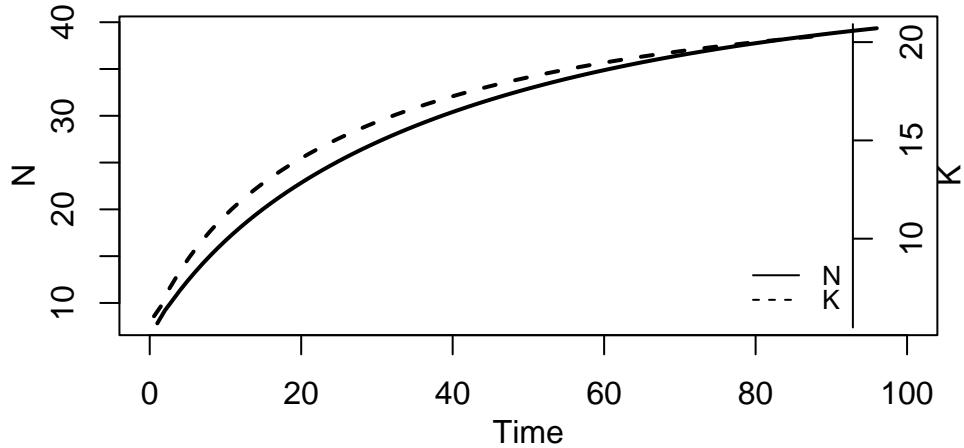
14.2.3 Plots

Figures 1-4 illustrate the model's dynamics under the baseline parameterisation. Starting from below-equilibrium levels, the economy grows in terms of output, capital, and employment but then approaches what Ricardo famously called a 'stationary state'. Figure 3 shows that during the adjustment phase, the MPL declines, reflecting diminishing marginal returns in agriculture. This captures the idea that a growing economy will have to utilise less fertile lands. The real wage is driven up until it is equal to the MPL. Figure 4 shows that total profits initially increase but are then squeezed to zero as differential rents increase.

```
# Set start and end periods for plots
Tmax=100
Tmin=6

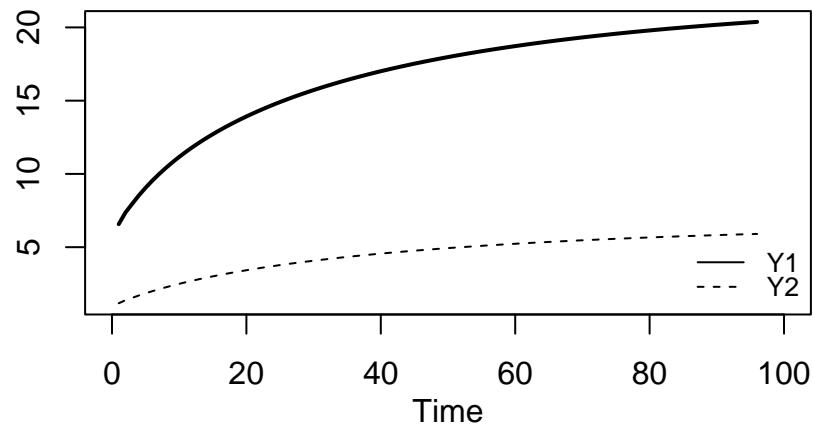
## Baseline
#Employment and capital accumulation
plot(N[1, Tmin:(Tmax+1)], type="l", lwd=2, lty=1, xlim=range(0:(Tmax)), ylab = '',
      title(main="Figure 1: Employment and capital accumulation", ylab = 'N', xlab = 'Time', cex=1,
            par(mar = c(5, 4, 4, 4) + 0.3)
            par(new = TRUE)
            plot(K[1, Tmin:(Tmax+1)], type="l", col=1, lwd=2, lty=2, font.main=1, cex.main=1, ylab = '',
                  xlab = '', ylim = range(K[1, Tmin:(Tmax+1)]), cex=0.8)
                  axis(side = 4, at = pretty(range(K[1, 2:(Tmax+1)])))
                  mtext("K", side = 4, line = 2)
                  legend("bottomright", legend=c("N", "K"),
                        lty=1:2, cex=0.8, bty = "n", y.intersp=0.8)
```

Figure 1: Employment and capital accumulation



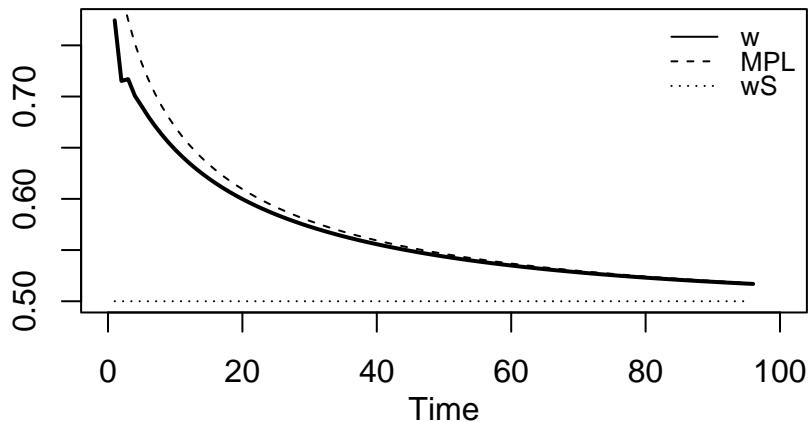
```
#Output in the two sectors
plot(Y1[1, Tmin:(Tmax+1)],type="l", col=1, lwd=2, lty=1, xlim=range(0:(Tmax)), xlab="", yl
title(main="Figure 2: Output in agriculture and luxuries", xlab = 'Time',cex=0.8 ,line=2)
lines(Y2[1, Tmin:(Tmax+1)],lty=2)
legend("bottomright", legend=c("Y1", "Y2"),
lty=1:2, cex=0.8, bty = "n", y.intersp=0.8)
```

Figure 2: Output in agriculture and luxuries



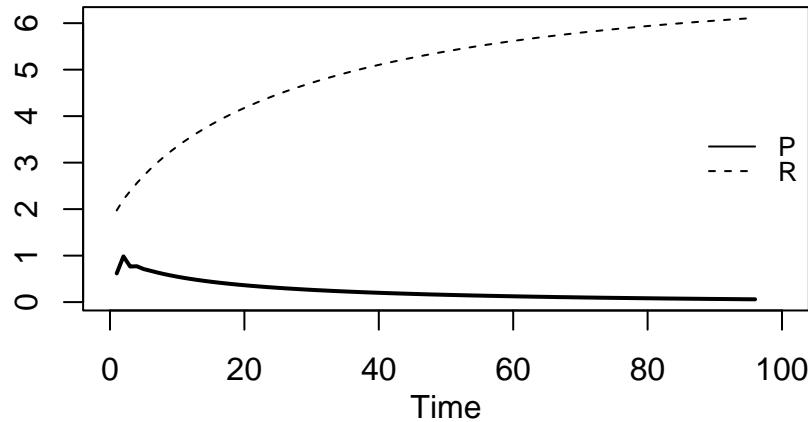
```
# Real wage, MPL, and subsistence wage
plot(w[1, Tmin:(Tmax+1)], type="l", col=1, lwd=2, lty=1, xlim=range(0:(Tmax)), xlab="", ylab="",
      title(main="Figure 3: Real wage, marginal product of labour, and subsistence wage", xlab = "Time"),
      lines(MPL[1, Tmin:Tmax], lty=2)
      lines(wS[1, Tmin:Tmax], lty=3)
      legend("topright", legend=c("w", "MPL", "wS"),
             lty=1:3, cex=0.8, bty = "n", y.intersp=0.8)
```

3: Real wage, marginal product of labour, and subsistence



```
# Total Profits and Rents
plot(P[1, Tmin:(Tmax+1)], type="l", col=1, lwd=2, lty=1, xlim=range(0:(Tmax)), xlab="", yla
title(main="Figure 4: Total profits and rents", xlab = 'Time', cex=0.8, line=2)
lines(R[1, Tmin:(Tmax+1)], lty=2)
legend("right", legend=c("P", "R"),
lty=1:2, cex=0.8, bty = "n", y.intersp=0.8)
```

Figure 4: Total profits and rents

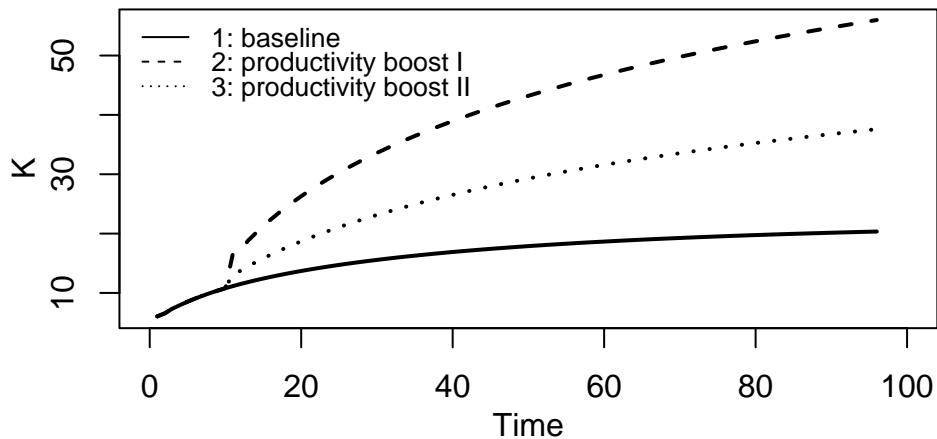


Figures 5 and 6 display capital accumulation under the five different scenarios described in Table 1. Technical change that increases productivity in agriculture (scenarios 2 and 3) raises the speed of capital accumulation and the equilibrium level of capital. By contrast, an increase in productivity in the luxury good sector (scenario 4) has no effect on capital accumulation. This is because productivity in sector 2 has no effects on functional income distribution.⁹ An increase in the initial stock of capital (scenario 5) raises the steady state value. Thus, economies with larger initial endowments will reach a higher level of income in the stationary state.

```
## Scenarios
# Capital accumulation under scenarios 1-3
plot(K[1, Tmin:(Tmax+1)], type="l", lwd=2, lty=1, xlim=range(0:(Tmax)), ylim=range(K[1, Tmin:(Tmax+1)]))
title(main="Figure 5: Capital accumulation under different scenarios (pt 1)", ylab = 'K', xlab = 'Time')
lines(K[2, Tmin:(Tmax+1)], lty=2, lwd=2)
lines(K[3, Tmin:(Tmax+1)], lty=3, lwd=2)
legend("topleft", legend=c("1: baseline", "2: productivity boost I", "3: productivity boost II"))
```

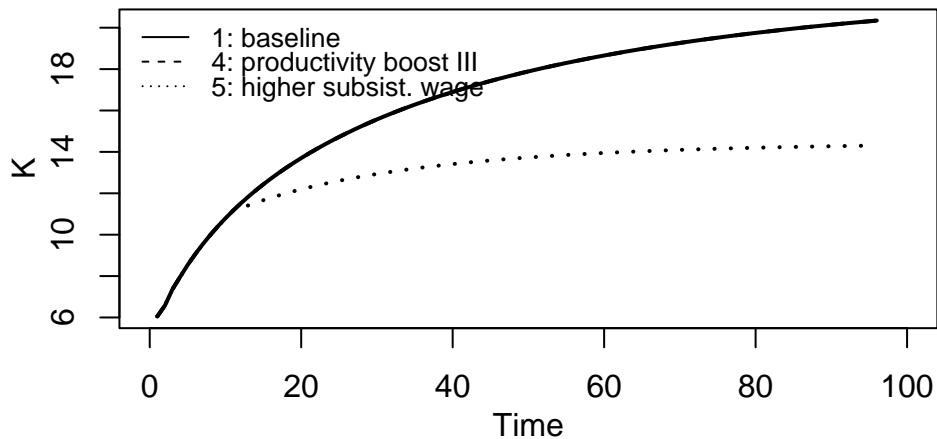
⁹The increase in a_2 does raise real output and profits in sector 2 but it leaves total profits unchanged.

Figure 5: Capital accumulation under different scenarios (p



```
# Capital accumulation under scenarios 1, 4+5
plot(K[1, Tmin:(Tmax+1)], type="l", lwd=2, lty=1, xlim=range(0:(Tmax)), ylim=range(K[1, Tmin:(Tmax+1)]))
title(main="Figure 6: Capital accumulation under different scenarios (pt 2)", ylab = 'K', xlab = 'Time')
lines(K[4, Tmin:(Tmax+1)], lty=2, lwd=2)
lines(K[5, Tmin:(Tmax+1)], lty=3, lwd=2)
legend("topleft", legend=c("1: baseline", "4: productivity boost III", "5: higher subsist.
```

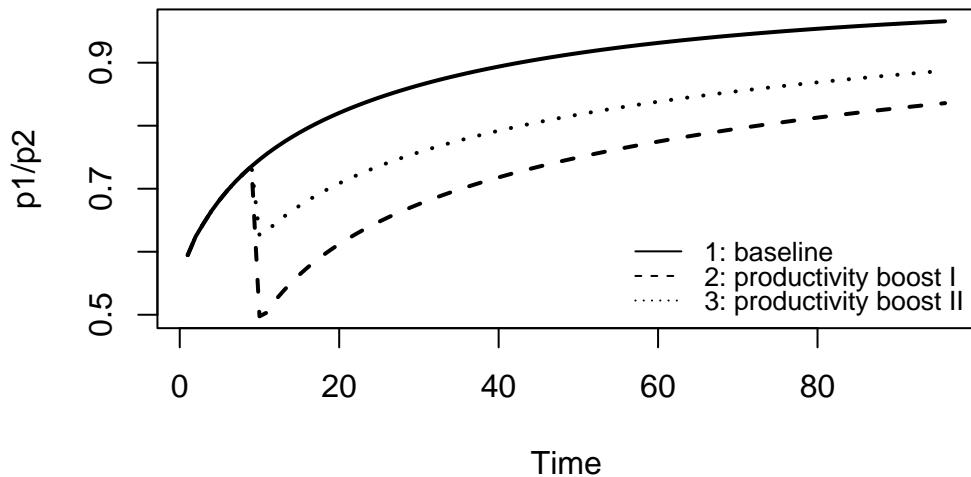
Figure 6: Capital accumulation under different scenarios (p



Figures 7 and 8 show the dynamics of relative prices (corn price relative to luxury good price) for the different scenarios. Over time, corn becomes more expensive in relative terms due to diminishing marginal returns. Improvements in labour productivity reduce the relative price of the respective sector in line with the labour theory of value.

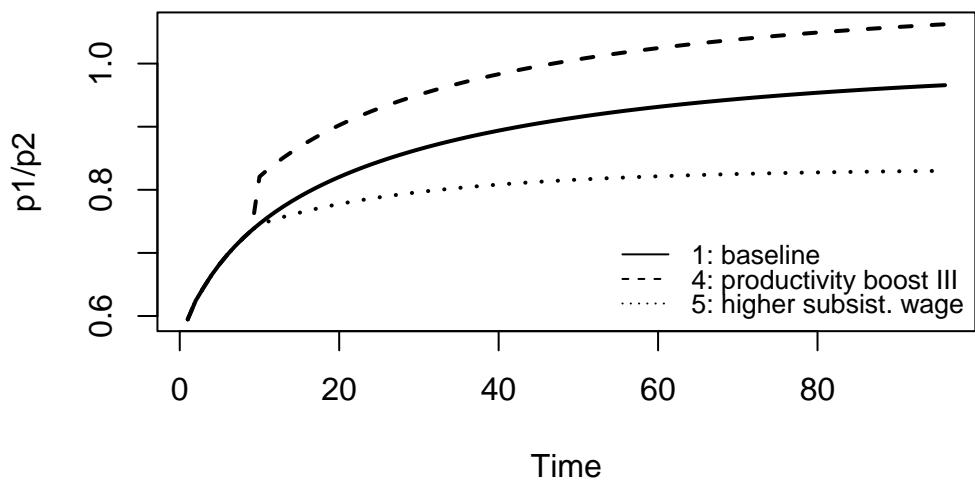
```
# Relative prices under scenarios 1 - 3
relpr=p1/p2
plot(relpr[1, Tmin:(Tmax+1)], type="l", col=1, lwd=2, lty=1, font.main=1, cex.main=1,
     main="Figure 7: Relative prices under different scenarios (pt 1)", ylab = 'p1/p2', xlab=
lines(relpr[2, Tmin:(Tmax+1)], lty=2, lwd=2)
lines(relpr[3, Tmin:(Tmax+1)], lty=3, lwd=2)
legend("bottomright", legend=c("1: baseline", "2: productivity boost I", "3: productivity
```

Figure 7: Relative prices under different scenarios (pt 1)



```
# Relative prices under scenarios 1, 4-5
plot(relpr[1, Tmin:(Tmax+1)], type="l", col=1, lwd=2, lty=1, font.main=1, cex.main=1,
     main="Figure 8: Relative prices under different scenarios (pt 2)", ylab = 'p1/p2', xlab=
lines(relpr[4, Tmin:(Tmax+1)], lty=2, lwd=2)
lines(relpr[5, Tmin:(Tmax+1)], lty=3, lwd=2)
legend("bottomright", legend=c("1: baseline", "4: productivity boost III", "5: higher subs"))
```

Figure 8: Relative prices under different scenarios (pt 2)



Python code

```
# Plots (here for employment and capital accumulation only)

import matplotlib.pyplot as plt

# Set start and end periods for plots
Tmax = 100
Tmin = 6

# Baseline
# Employment and capital accumulation
fig, ax1 = plt.subplots()
ax1.plot(N[0, 2:(Tmax+1)], linestyle='solid', label='N', linewidth=0.8, color="black")

ax1.set_xlabel('Time')
ax1.set_ylabel('N', rotation=0)
ax2 = ax1.twinx()
ax2.plot(K[0, 2:Tmax], linestyle='dashed', label='K', linewidth=0.8, color="black")
ax2.set_ylabel('K', rotation=0)
lines, labels = ax1.get_legend_handles_labels() #collect legend in one box
lines2, labels2 = ax2.get_legend_handles_labels()
ax2.legend(lines + lines2, labels + labels2, loc=5)
plt.title("Figure 1: Employment and capital accumulation")
plt.show()
```

14.3 Directed graph

Another perspective on the model's properties is provided by its directed graph. A directed graph consists of a set of nodes that represent the variables of the model. Nodes are connected by directed edges. An edge directed from a node x_1 to node x_2 indicates a causal impact of x_1 on x_2 .¹⁰

```
## Create directed graph
# Construct auxiliary Jacobian matrix for 18 variables:
#(1)Y1 (2)N1 (3)MPL (4)R (5)P1 (6)p1 (7)Y2 (8)w (9)N2 (10)p2 (11)P2
#(12)P (13)K (14)N (15)A (16)a1 (17)a2 (18)wS,
```

¹⁰Valuation effects from changes in relative prices were omitted for simplicity.

```

# where non-zero elements in regular Jacobian are set to 1 and zero elements are unchanged

#
#          1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
M_mat=matrix(c(0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0, #1
              0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0, #2
              0,1,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0, #3
              1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #4
              1,1,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0, #5
              0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #6
              0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #7
              0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0, #8
              0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0, #9
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0, #10
              0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0, #11
              0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0, #12
              0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0, #13
              0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1, #14
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #15
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #16
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #17
              0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, #18
            ), 18,18, byrow=TRUE)

# Create adjacency matrix from transpose of auxiliary Jacobian
A_mat=t(M_mat)

# Create and plot directed graph from adjacency matrix
library(igraph)
dg= graph_from_adjacency_matrix(A_mat, mode="directed", weighted= NULL)

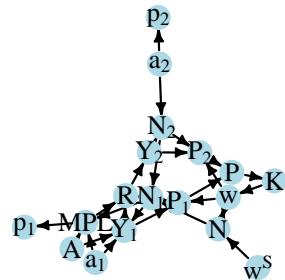
# Define node labels
V(dg)$name=c(expression(Y[1]), expression(N[1]), "MPL", "R", expression(P[1]),
              expression(p[1]), expression(Y[2]), "w", expression(N[2]),
              expression(p[2]), expression(P[2]), "P", "K", "N", "A",
              expression(a[1]), expression(a[2]), expression(w^S))

# Plot directed graph
# Plot directed graph
plot(dg, main="Figure 9: Directed graph", vertex.size=20, vertex.color="lightblue",
      vertex.label.color="black", edge.arrow.size=0.3, edge.width=1.1, edge.size=1.2,
      edge.arrow.width=1.2, edge.color="black", vertex.label.cex=0.8,

```

```
vertex.frame.color="NA", margin=0.08)
```

Figure 9: Directed graph



 Python code

```

# Create directed graph

import networkx as nx

# Construct auxiliary Jacobian matrix for 18 variables:
#(1)Y1 (2)N1 (3)MPL (4)R (5)P1 (6)p1 (7)Y2 (8)w (9)N2 (10)p2 (11)P2
#(12)P (13)K (14)N (15)A (16)a1 (17)a2 (18)wS,
# where non-zero elements in regular Jacobian are set to 1 and zero elements are unchanged

#
#          1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
M_mat = np.array([
    [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0], #1
    [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0], #2
    [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0], #3
    [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], #4
    [1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], #5
    [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], #6
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], #7
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0], #8
    [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], #9
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], #10
    [0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0], #11
    [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0], #12
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0], #13
    [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], #14
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], #15
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], #16
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], #17
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]] #18

# Create adjacency matrix from transpose of auxiliary Jacobian and add column names
A_mat = M_mat.transpose()

# Create the graph from the adjacency matrix
G = nx.DiGraph(A_mat)

# Define node labels
nodelabs = {
    0: r'$Y_1$', 1: r'$N_1$', 2: 'MPL', 3: 'R', 4: r'$P_1$',
    5: r'$p_1$', 6: r'$Y_2$', 7: 'w', 8: r'$N_2$', 9: r'$p_2$',
    10: r'$P_2$', 11: 'P', 12: 'K', 13: 'N', 14: 'A',
    15: r'$a_1$', 16: r'$a_2$', 17: r'$w^S$'
}

}

# Plot the graph
pos = nx.spring_layout(G, k=0.9)
nx.draw_networkx(G, pos, node_size=200, node_color="lightblue",
                 edge_color="black", width=1.2, arrowsize=10,
                 arrowstyle='->', font_size=8, font_color="black",
                 with_labels=True, labels=nodelabs)

plt.axis("off")
plt.title("Figure: Directed graph of Ricardian Two-Sector Model")
plt.show()

```

In Figure 9, it can be seen that productivity in agriculture (A and a_1) are key exogenous variables that impact income in sector 1 and the marginal product of labour. The subsistence wage (w^S) is another exogenous variable that impacts the system through its effect on population dynamics. Productivity in the luxuries sector (a_2) feeds into the system via employment. Most other variables are endogenous and form a closed loop (or cycle) within the system. Profits are a residual. The directed graph illustrates the supply-driven nature of the agricultural sector, where (marginal) productivity determine employment and distribution. By contrast, the luxury goods sector is demand-determined with employment being the residual. Profits determine capital accumulation, which in turn provides funds that can be used to hire more agricultural workers. A higher subsistence wage reduces capital accumulation as it leaves fewer profits to be reinvested.

14.4 Analytical discussion

To analyse the dynamics, combine Equation 14.1 to Equation 14.13 and substitute into Equation 14.14. Further use Equation 14.4 and Equation 14.5 in Equation 14.15. This yields the two-dimensional dynamic system in K_t and N_t :

$$\begin{aligned} K_t &= (1 - \beta)K_{t-1} + \beta(a_1^{a_1} AN_{t-1}^{a_1}) \\ N_t &= N_{t-1} + \gamma \left(\frac{K_{t-1}}{N_{t-1}} - w^S \right) \end{aligned}$$

The Jacobian matrix is given by:

$$J(K, N) = \begin{bmatrix} 1 - \beta & \beta a_1^{1+a_1} A N^{a_1-1} \\ \frac{\gamma}{N} & 1 - \frac{\gamma K}{N^2} \end{bmatrix}.$$

From equations Equation 14.14 and Equation 14.15, it can readily be seen that an equilibrium is reached when

$$P^* = 0$$

and

$$w^* = w^S.$$

Using $P^* = 0$ with Equation 14.6 and Equation 14.13, yields $w^* = w^S = MPL$. Thus, in equilibrium, profits are zero, and the real wage is equal to the MPL and the subsistence wage. Setting $K_t = K_{t-1}$ and $N_t = N_{t-1}$, we can further derive:

$$K^* = a_1^{a_1} A \left(\frac{w^S}{a_1^{a_1} A} \right)^{-\frac{a_1}{1-a_1}}$$

and

$$N^* = \left(\frac{w^S}{a_1^{a_1} A} \right)^{-\frac{1}{1-a_1}}$$

With this, we can evaluate the Jacobian at the steady state:

$$J(K^*, N^*) = \begin{bmatrix} 1 - \beta & \beta a_1 w^S \\ \gamma \left(\frac{w^S}{a_1^{a_1} A} \right)^{\frac{1}{1-a_1}} & 1 - \gamma a_1^{a_1} A \left(\frac{w^S}{a_1^{a_1} A} \right)^{\frac{2-a_1}{1-a_1}} \end{bmatrix}.$$

For the system to be stable, both eigenvalues of the Jacobian need to be inside the unit circle. This requires the following three conditions to hold:

$$\begin{aligned} 1 + \text{tr}(J) + \det(J) &> 0 \\ 1 + \text{tr}(J) - \det(J) &> 0 \\ 1 - \det(J) &> 0, \end{aligned}$$

where $\text{tr}(J)$ is the trace and $\det(J)$ is the determinant of the Jacobian.

Let us consider the Classical case where $\beta = 1$, i.e. all profits are reinvested. Then we have

$$\det(J) = -a_1 w^S \gamma \left(\frac{w^S}{a_1^{a_1} A} \right)^{\frac{1}{1-a_1}} < 0,$$

so that the third condition is always satisfied and it is the first one that is binding. The first condition then becomes

$$2 - \gamma \left[a_1^{a_1} A \left(\frac{w^S}{a_1^{a_1} A} \right)^{\frac{2-a_1}{1-a_1}} + a_1 w^S \left(\frac{w^S}{a_1^{a_1} A} \right)^{\frac{1}{1-a_1}} \right] > 0$$

We can check the analytical solutions and stability conditions numerically:

```
# Calculate equilibrium solutions
for (i in 1:S){
  N_eq[i]=(wS[i,Q]/((a1[i,Q]^a1[i,Q])*A[i,Q]))^(-1/(1-a1[i,Q]))
  K_eq[i]=(a1[i,Q]^a1[i,Q])*A[i,Q]*(wS[i,Q]/((a1[i,Q]^a1[i,Q])*A[i,Q]))^(-a1[i,Q]/(1-a1[i,Q]))
}

# Compare with numerical solutions (here for the example of Y, baseline)
N_eq[1]
```

[1] 44.20066

```
N[1,Q]
```

```
[1] 44.19942
```

```
K_eq[1]
```

```
[1] 22.10033
```

```
K[1,Q]
```

```
[1] 22.09989
```

```
### Examine model properties (here for the baseline scenario only)
# Construct Jacobian matrix at the equilibrium
J=matrix(c(1-beta,
           beta*a1[1,Q]*wS[1,Q],
           gamma*(wS[1,Q]/((a1[i,Q]^a1[i,Q])*A[1,Q]))^(1/(1-a1[1,Q])),
           1-gamma*(a1[i,Q]^a1[i,Q])*A[1,Q]*(wS[1,Q]/((a1[i,Q]^a1[i,Q])*A[1,Q]))^((2-a1[1,Q])/((1-a1[1,Q]))))
```

```
# Obtain eigenvalues
```

```
ev=eigen(J)
(values = ev$values)
```

```
[1] 0.98368845 -0.04024869
```

```
# Obtain determinant and trace
det=det(J)      # determinant
tr=sum(diag(J)) # trace
```

```
#Check stability conditions
print(1+tr+det>0)
```

```
[1] TRUE
```

```
print(1-tr-det>0)
```

```
[1] TRUE
```

```
print(1-det>0)

[1] TRUE

# Check specific stability condition for the case beta=1
for (i in 1:S){
  print(paste0("Scenario ", i, ":"))
  print(2-gamma*((a1[i,Q]^a1[i,Q])*A[i,Q]*(wS[i,Q]/(((a1[i,Q]^a1[i,Q])*A[i,Q])))^((2-a1[i,Q])
}

[1] "Scenario 1:"
[1] TRUE
[1] "Scenario 2:"
[1] TRUE
[1] "Scenario 3:"
[1] TRUE
[1] "Scenario 4:"
[1] TRUE
[1] "Scenario 5:"
[1] TRUE
```

 Python code

```

# Initialize arrays for equilibrium solutions
N_eq = np.zeros(S)
K_eq = np.zeros(S)

# Calculate equilibrium solutions
for i in range(S):
    N_eq[i] = (wS[i, Q-1] / ((a1[i, Q-1] ** a1[i, Q-1]) * A[i, Q-1])) ** (-1 / (1 - a1[i, Q-1]))
    K_eq[i] = (a1[i, Q-1] ** a1[i, Q-1]) * A[i, Q-1] * (wS[i, Q-1] / ((a1[i, Q-1] ** a1[i, Q-1]) * A[i, Q-1])) ** (-1 / (1 - a1[i, Q-1]))

# Compare with numerical solutions (example for N, baseline)
N_eq[0]
N[0,Q-1]

# Construct Jacobian matrix at the equilibrium
J = np.array([
    [1 - beta, beta * a1[0, Q-1] * wS[0, Q-1]],
    [gamma * (wS[0, Q-1] / ((a1[0, Q-1] ** a1[0, Q-1]) * A[0, Q-1])) ** (1 / (1 - a1[0, Q-1])),
     1 - gamma * (a1[0, Q-1] ** a1[0, Q-1]) * A[0, Q-1] * (wS[0, Q-1] / ((a1[0, Q-1] ** a1[0, Q-1]) * A[0, Q-1])) ** (-1 / (1 - a1[0, Q-1]))]
])

# Obtain eigenvalues
eigenvalues, eigenvectors = np.linalg.eig(J)
print(eigenvalues)

# Obtain determinant and trace
det = np.linalg.det(J)
tr = np.trace(J)

# Check general stability conditions
print(1+tr+det>0)
print(1-tr+det>0)
print(1-det>0)

# Check specific stability condition for the case beta=1
for i in range(S):
    print(f"Scenario {i + 1}:")
    print(2 - gamma * (
        (a1[i, Q-1] ** a1[i, Q-1]) * A[i, Q-1] *
        (wS[i, Q-1] / ((a1[i, Q-1] ** a1[i, Q-1]) * A[i, Q-1])) ** ((2 - a1[i, Q-1]) / (a1[i, Q-1] * wS[i, Q-1] * (wS[i, Q-1] / ((a1[i, Q-1] ** a1[i, Q-1]) * A[i, Q-1])) ** (-1 / (1 - a1[i, Q-1])))) > 0)

```

14.5 References

15 A Lewis Model of Economic Development

15.1 Overview

This model captures some key features of W. Arthur Lewis (1954)' two-sector model of the growth process of a developing country. The model describes the development process as one where a modern urban sector draws labour inputs from a traditional subsistence sector that is characterised by surplus labour, i.e. labour that does not produce any additional output (reflected in a zero marginal product). Workers in the traditional sector are paid a fixed subsistence wage. The presence of surplus labour allows the modern sector to attract workers from the traditional sector for a small wage premium above the subsistence wage. This enables the modern sector to make profits that are reinvested and thus drive capital accumulation in the modern sector. As a result, the economy undergoes industrialisation. This process of rapid structural change comes to an end when the surplus labour in the traditional sector is depleted. Labour supply then becomes sensitive to the real wage, leading to a fall in profits and a slowdown of capital accumulation.

We present a formal version of the model that is based on the graphical representation in Todaro and Smith (2015), pp. 124-127.

15.2 The Model

$$Y_{1t} = \begin{cases} L_{1t}^\alpha, & \text{if } L_{1t} \leq \lambda \\ \lambda^\alpha, & \text{if } L_{1t} > \lambda \end{cases} \quad (15.1)$$

$$L_{1t} = L - L_{2t} \quad (15.2)$$

$$w_{1t} = w^S \quad (15.3)$$

$$Y_{2t} = L_{2t}^\beta K_{2t}^{1-\beta} \quad (15.4)$$

$$w_{2t} = \begin{cases} \gamma L_{2t}, & \text{if } L_{1t} \leq \lambda \\ w^S + \rho, & \text{if } L_{1t} > \lambda \end{cases} \quad (15.5)$$

$$L_{2t} = \frac{\beta Y_{2t}}{w_{2t}} \quad (15.6)$$

$$P_{2t} = Y_{2t} - w_{2t} L_{2t} \quad (15.7)$$

$$K_{2t} = K_{2t-1} + P_{2t-1} \quad (15.8)$$

where the subscripts 1 and 2 refer to the traditional and modern sector, respectively, and Y_t , L_t , w_t , K_t , and P_t represent output, employment, the real wage rate, the capital stock, and profits, respectively.

Equation 15.1 describes output determination in the traditional sector. The variable λ is the employment level beyond which the marginal product of labour (MPL) in sector 1 becomes zero. If actual employment is below this level, the sector faces a production function that is increasing in employment but with diminishing marginal returns ($\alpha \in (0, 1)$). If employment is above this threshold, there is surplus labour and output is fixed at the level implied by the level of employment for which the MPL becomes zero. Equation 15.2 says that the level of employment in the traditional sector is residually determined by the total labour supply, which we take to be exogenous, net of employment in the modern sector. By Equation 15.3, the real wage in the traditional sector is an exogenously given subsistence wage w^S . Equation 15.4 is the production function of the modern sector, which exhibits constant returns to scale and diminishing marginal returns to the factors ($\beta \in (0, 1)$). Equation 15.5 specifies wage determination in the modern sector. If there is surplus labour in the traditional sector, the modern sector pays a wage premium ρ on the subsistence wage. As soon as surplus labour is depleted, the modern sector faces an upward-sloping labour supply curve, which we model for simplicity as a linear function with slope coefficient γ . Equation 15.6 is the labour demand curve of the modern sector, which is derived from optimisation. To maximise profits, the firm must equalise the real wage and the MPL: $w_{2t} = MPL_{2t} = \beta L_{2t}^{\beta-1} K_{2t}^{1-\beta} = \beta \frac{Y_{2t}}{L_{2t}}$. Solving this condition for L_{2t} yields $L_{2t} = K_{2t} \left(\frac{w_{2t}}{\beta} \right)^{-\frac{1}{1-\beta}} = \frac{\beta Y_{2t}}{w_{2t}}$. Finally, Equation 15.7 defines profits in the modern sector, and Equation 15.8 describes capital accumulation in that sector, where it is assumed that all profits are invested ($P_{2t} = I_{2t}$).

15.3 Simulation

15.3.1 Parameterisation

Table reports the parameterisation and initial values used in the simulation. All scenarios are initialised such that there is surplus labour ($L_{10} > \lambda$).

Table 1: Parameterisation

Scenario	α	ρ	L	β	γ	λ	w^S
1: baseline	0.7	1	20	0.7	0.55	0.05	1
2: rise subsistence wage (w^S)	0.7	1	20	0.7	0.55	0.05	1.1

15.3.2 Simulation code

```
# Clear the environment
rm(list=ls(all=TRUE))

# Set number of periods
T=250

# Set number of scenarios (including baseline)
S=2

# Set period in which shock/shift will occur
s=15

# Create (S x T)-matrices that will contain the simulated data
Y1=matrix(data=1,nrow=S,ncol=T) # Output in sector 1 (traditional)
Y2=matrix(data=1,nrow=S,ncol=T) # Output in sector 2 (modern)
L1=matrix(data=1,nrow=S,ncol=T) # employment in sector 1
L2=matrix(data=1,nrow=S,ncol=T) # employment in sector 2
w2=matrix(data=1,nrow=S,ncol=T) # real wage sector 2
K=matrix(data=1,nrow=S,ncol=T) # capital stock (only in sector 2)
P2=matrix(data=1,nrow=S,ncol=T) # profits in sector 2

# Set fixed parameter values
alpha=0.7 # labour elasticity of output, sector 1
rho=1      # wage premium
L=20       # total labour supply (exogenous)
```

```

gamma=0.2 # labour supply coefficient, sector 2
lambda=10 # employment at which MPL in sector 1 becomes zero
beta=0.7 # labour elasticity of output, sector 2

# Set baseline parameter values
w1=matrix(data=1,nrow=S,ncol=T) # subsistence real wage sector 1

# Set parameter values for different scenarios
w1[2,s:T]=0.9 # scenario 2: fall in subsistence wage

# Initialise such that there is surplus labour (L1 > lambda)
L1[,1]= 0.9*L
L2[,1]= L - L1[,1]

# Simulate the model by looping over T time periods for S different scenarios
for (i in 1:S){

  for (t in 2:T){

    for (iterations in 1:1000){ # note: order matters due to if-else condition

      ## Model equations
      # Output sector1 and wages sector 2
      if(L1[i,t]<lambda){
        Y1[i,t] = (L1[i,t]^alpha)
        w2[i,t]= gamma*L2[i,t]
      }
      else{
        Y1[i,t] = lambda^alpha
        w2[i,t]= w1[i,t] + rho
      }

      # Output sector 2
      Y2[i,t]=(L2[i,t]^beta)*(K[i,t]^(1-beta))

      # Profits sector 2
      P2[i,t]=Y2[i,t] - w2[i,t]*L2[i,t]

      # Capital accumulation sector 2
      K[i,t]= K[i,t-1] + P2[i,t-1]
    }
  }
}

```

```

# Employment sector 2
L2[i,t] =(beta*Y2[i,t])/w2[i,t]

# Employment sector 1
L1[i,t] = L - L2[i,t]

} # close iterations loop
} # close time loop
} # close scenarios loop

#Calculate profit share of sector 2
PS=P2[,]/(Y1[,]+Y2[,])

#Find period in which Lewis turning point occurs
tp=c(min(which(L1[1,< lambda)), min(which(L1[2,< lambda))))

```

 Python code

```

import numpy as np

# Set number of periods
T = 250

# Set number of scenarios (including baseline)
S = 2

# Set period in which shock/shift will occur
s = 15

# Create (S x T)-matrices that will contain the simulated data
Y1 = np.ones((S, T)) # Output in sector 1 (traditional)
Y2 = np.ones((S, T)) # Output in sector 2 (modern)
L1 = np.ones((S, T)) # employment in sector 1
L2 = np.ones((S, T)) # employment in sector 2
w2 = np.ones((S, T)) # real wage sector 2
K = np.ones((S, T)) # capital stock (only in sector 2)
P2 = np.ones((S, T)) # profits in sector 2

# Set fixed parameter values
alpha = 0.7 # labour elasticity of output, sector 1
rho = 1 # wage premium
L = 20 # total labour supply (exogenous)
gamma = 0.2 # labour supply coefficient, sector 2
lambda_val = 10 # employment at which MPL in sector 1 becomes zero
beta = 0.7 # labour elasticity of output, sector 2

# Set baseline parameter values
w1 = np.ones((S, T)) # subsistence real wage sector 1 (baseline)

# Set parameter values for different scenarios
w1[1, s:T] = 0.9 # scenario 2: fall in subsistence wage

# Initialise such that there is surplus labour (L1 > lambda)
L1[:, 0] = 0.9 * L
L2[:, 0] = L - L1[:, 0]

# Simulate the model by looping over T time periods for S different scenarios
for i in range(S):
    for t in range(1, T):
        for iterations in range(1000): # note: order matters due to if-else condition
            # Model equations
            # Output sector1 and wages sector 2
            if L1[i, t] < lambda_val:
                Y1[i, t] = L1[i, t]**alpha
                w2[i, t] = gamma * L2[i, t]
            else:
                Y1[i, t] = lambda_val**alpha
                w2[i, t] = w1[i, t] + rho

            # Output sector 2

```

15.3.3 Plots

Figure 15.1 - Figure 15.4 display the model's dynamics under the baseline scenario. The red vertical line marks the ‘Lewis turning point’, which is the period in which the surplus labour in the traditional sector is depleted ($L_{1t} < \lambda$). It can be seen that before the Lewis turning point is reached, output in the modern sector grows exponentially. After the turning point, it still grows but at a lower rate. This is because as the surplus labour in the traditional sector is removed, the modern sector faces an upward-sloping labour supply curve. As a result, real wages increase and profits are reduced. This slows down capital accumulation and growth. In the traditional sector, output falls somewhat after the turning point as the sector now faces diminishing marginal return to labour.

```
# Set start and end periods for plots
Tmax=240
Tmin=2

#Output in the two sectors, baseline
plot(Y1[1, Tmin:(Tmax+1)], type="l", col=1, lwd=2, lty=1, xlim=range(0:(Tmax)), xlab="", yl
title(main="Output in traditional (Y1) and modern (Y2) sector", xlab = 'Time', cex=0.8, lin
lines(Y2[1, Tmin:(Tmax+1)], lty=2)
abline(v = tp[1], col = "darkred")
legend("topleft", legend=c("Y1", "Y2"),
       lty=1:2, cex=0.8, bty = "n", y.intersp=0.8)
```

Output in traditional (Y1) and modern (Y2) sector

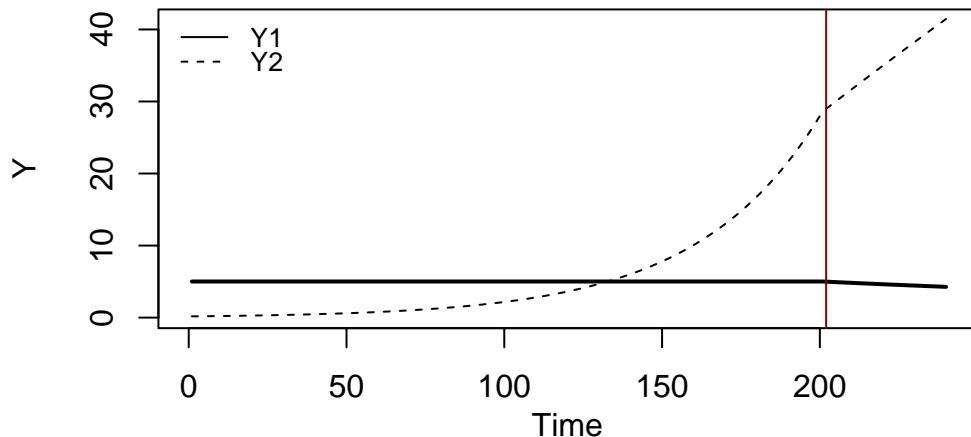


Figure 15.1: Output in traditional (Y1) and modern (Y2) sector

```
#Employment in the two sectors, baseline
plot(L1[1, Tmin:(Tmax+1)],type="l", col=1, lwd=2, lty=1, xlim=range(0:(Tmax)), xlab="", yl
title(main="Employment in traditional (L1) and modern (L2) sector", xlab = 'Time',cex=0.8
lines(L2[1, Tmin:(Tmax+1)],lty=2)
abline(v = tp[1], col = "darkred")
legend("bottomright", legend=c("L1", "L2"),
      lty=1:2, cex=0.8, bty = "n", y.intersp=0.8)
```

Employment in traditional (L1) and modern (L2) sector

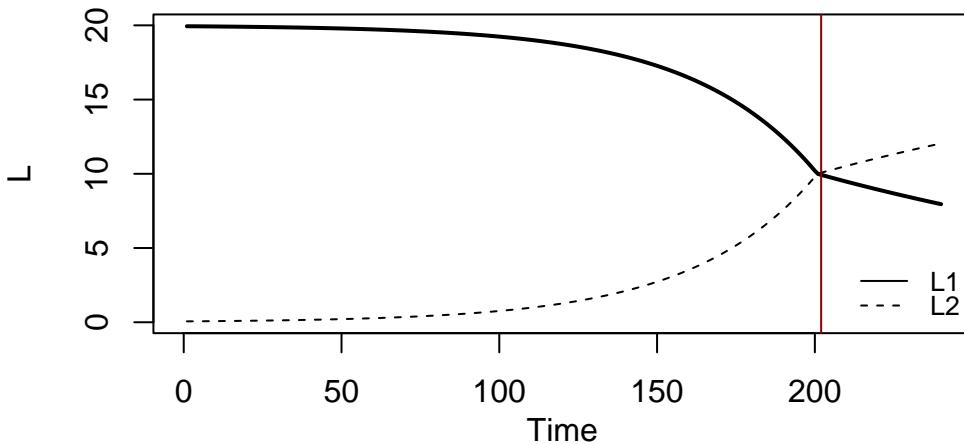


Figure 15.2: Employment in traditional (L1) and modern (L2) sector

```
#Profits and capital accumulation in manufacturing, baseline
plot(P2[1, Tmin:(Tmax+1)],type="l", col=1, lwd=2, lty=1, xlim=range(0:(Tmax)), xlab="", yl
title(main="Capital accumulation and profits in modern sector",ylab = 'P', xlab = 'Time',
abline(v = tp[1], col = "darkred")
par(mar = c(5, 4, 4, 4) + 0.3)
par(new = TRUE)
plot(K[1, Tmin:Tmax],type="l", col=1, lwd=2, lty=2, font.main=1,cex.main=1,ylab = '',
      xlab = '',ylim = range(K[1, 2:(Tmax+1)]),cex.axis=1,cex.lab=0.75)
axis(side = 4, at = pretty(range(K[1, 2:(Tmax+1)])))
mtext("K", side = 4, line = 2)
legend("bottomright", legend=c("P", "K"),
       lty=1:2, cex=0.8, bty = "n", y.intersp=0.8)
```

Capital accumulation and profits in modern sector

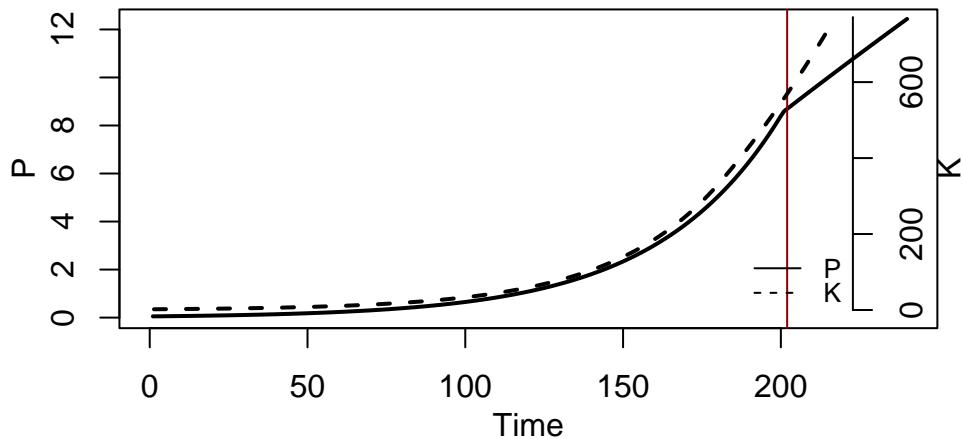


Figure 15.3: Capital accumulation and profits in modern sector

```
# Real wage and profit share in manufacturing
plot(w2[1, Tmin:(Tmax+1)], type="l", col=1, lwd=2, lty=1, xlim=range(0:(Tmax)), xlab="", yl
title(main="Real wage and profit share in manufacturing sector", ylab = 'w', xlab = 'Time',
abline(v = tp[1], col = "darkred")
par(mar = c(5, 4, 4, 4) + 0.3)
par(new = TRUE)
plot(PS[1, Tmin:Tmax], type="l", col=1, lwd=2, lty=2, font.main=1, cex.main=1, ylab = '',
      xlab = '', ylim = range(PS[1, 2:(Tmax+1)]), cex.axis=1, cex.lab=0.75)
axis(side = 4, at = pretty(range(PS[1, 2:(Tmax+1)])))
mtext("pshare", side = 4, line = 2)
legend("topleft", legend=c("w", "pshare"),
      lty=1:2, cex=0.8, bty = "n", y.intersp=0.8)
```

Real wage and profit share in manufacturing sector

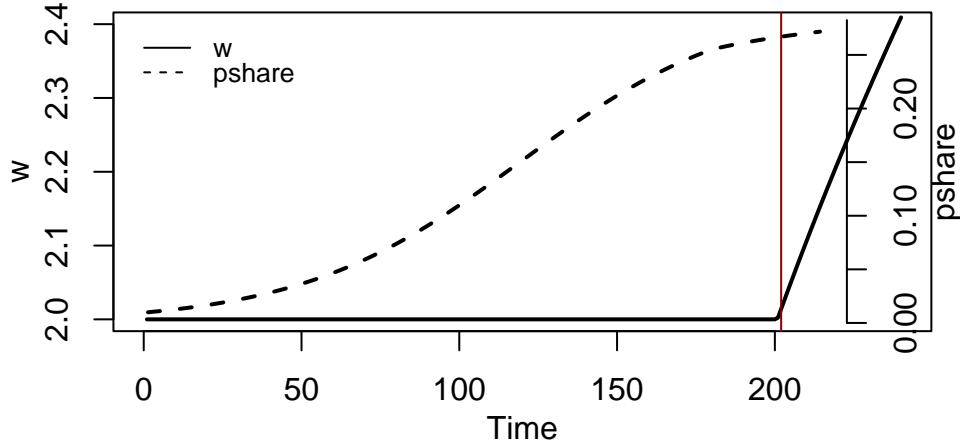


Figure 15.4: Real wage and profit share in manufacturing sector

Figure 15.5 compares output dynamics in the modern sector for the baseline scenario and scenario 2 in which the subsistence wage w^s falls in $t = 15$. It can be seen that for a lower subsistence wage, the modern sector grows faster and the Lewis turning point is reached earlier. This result illustrates the Classical nature of the model, whereby capital accumulation is driven by profits. Thus, any reduction in profits slows down the growth process.

```
plot(Y2[1, Tmin:(Tmax+1)], type="l", col=1, lwd=2, lty=1, xlim=range(0:(Tmax)), xlab="", yl
title(main="Output in modern sector: baseline vs decrease in subsistence wage", xlab = 'T'
lines(Y2[2, Tmin:(Tmax+1)], lty=2)
abline(v = tp[1], col = "darkred")
abline(v = tp[2], col = "red")
legend("topleft", legend=c("1: baseline", "2: decrease in subsistence wage"),
lty=1:2, cex=0.8, bty = "n", y.intersp=0.8)
```

Output in modern sector: baseline vs decrease in subsistence wage

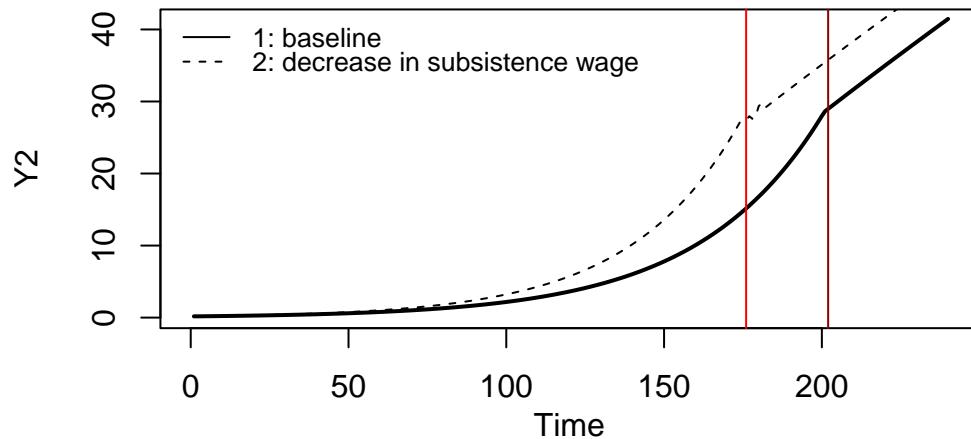


Figure 15.5: Output in modern sector (baseline vs decrease in subsistence wage)

Python code

```
### Plots (here only for output)
import matplotlib.pyplot as plt

# Set start and end periods for plots
Tmax = 240
Tmin = 2

# Output in the two sectors, baseline
plt.plot(range(Tmin, Tmax + 1), Y1[0, Tmin:Tmax + 1], 'k-', linewidth=2, label='Y1')
plt.plot(range(Tmin, Tmax + 1), Y2[0, Tmin:Tmax + 1], 'k--', linewidth=2, label='Y2')

# Highlight Lewis turning point
plt.axvline(x=tp[0], color='darkred', linestyle='--')

# Set plot labels and title
plt.title('Output in traditional (Y1) and modern (Y2) sector')
plt.xlabel('Time')
plt.ylabel('Y')
plt.ylim(min(Y1[0, Tmin:Tmax]), max(Y2[0, Tmin:Tmax]))

# Add legend
plt.legend(loc='upper left')

# Show the plot
plt.show()
```

15.4 Directed graph

Another perspective on the model's properties is provided by its directed graph. A directed graph consists of a set of nodes that represent the variables of the model. Nodes are connected by directed edges. An edge directed from a node x_1 to node x_2 indicates a causal impact of x_1 on x_2 .

```
## Create directed graph
# Construct auxiliary Jacobian matrix for 8 variables:

#           Y1,Y2,L1,L2,w1,w2,P2,K
```

```

M_mat=matrix(c(0, 0, 1, 0, 0, 0, 0, 0,
              0, 0, 0, 1, 0, 0, 0, 1,
              0, 0, 0, 1, 0, 0, 0, 0,
              0, 1, 0, 0, 0, 1, 0, 0,
              0, 0, 0, 0, 0, 0, 0, 0,
              0, 0, 0, 0, 1, 0, 0, 0,
              0, 1, 0, 1, 0, 1, 0, 0,
              0, 0, 0, 0, 0, 1, 0, 0), 8, 8, byrow=TRUE)

# Create adjacency matrix from transpose of auxiliary Jacobian
A_mat=t(M_mat)

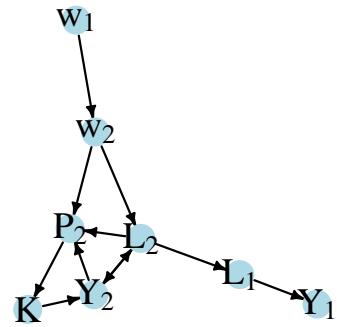
# Create directed graph from adjacency matrix
library(igraph)
dg= graph_from_adjacency_matrix(A_mat, mode="directed", weighted= NULL)

# Define node labels
V(dg)$name=c(expression(Y[1]), expression(Y[2]), expression(L[1]), expression(L[2]),
              expression(w[1]), expression(w[2]), expression(P[2]), "K")

# Plot directed graph
plot(dg, main="Directed Graph of Lewis Model", vertex.size=20, vertex.color="lightblue",
      vertex.label.color="black", edge.arrow.size=0.3, edge.width=1.1, edge.size=1.2,
      edge.arrow.width=1.2, edge.color="black", vertex.label.cex=1.2,
      vertex.frame.color="NA", margin=-0.08)

```

Directed Graph of Lewis Model



 Python code

```
# Create directed graph
import networkx as nx

# Construct auxiliary Jacobian matrix for 8 variables:

#          Y1,Y2,L1,L2,w1,w2,P2,K
M_mat = np.array([
    [0, 0, 1, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 1],
    [0, 0, 0, 1, 0, 0, 0, 0],
    [0, 1, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 1, 0, 0, 0],
    [0, 1, 0, 1, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 1, 0]]))

# Create adjacency matrix from transpose of auxiliary Jacobian and add column names
A_mat = M_mat.transpose()

# Create the graph from the adjacency matrix
G = nx.DiGraph(A_mat)

# Define node labels
nodelabs = {0: r'$Y_1$', 1: r'$Y_2$', 2: r'$L_1$', 3: r'$L_2$',
            4: r'$w_1$', 5: r'$w_2$', 6: r'$P_2$', 7: 'K'}

# Plot the graph
pos = nx.spring_layout(G, k=0.4)
nx.draw_networkx(G, pos, node_size=200, node_color="lightblue",
                 edge_color="black", width=1.2, arrowsize=10,
                 arrowstyle='->', font_size=8, font_color="black",
                 with_labels=True, labels=nodelabs)
plt.axis("off")
plt.title("Directed Graph of Lewis Model")
plt.show()
```

The directed graph illustrates that the subsistence wage in the traditional sector $w_1 = w^s$ is the key exogenous variables that impact the dynamics of growth and distribution via its effect on real wages in the modern sector. Employment, output, profits, and capital accumulation

in the modern sector form a closed loop (or cycle) within the system. The traditional sector plays a residual role.

15.5 Analytical discussion

To analyse the dynamics of capital accumulation before the Lewis turning point, i.e. for the case where ($L_{1t} > \lambda$), combine Equation 15.4, Equation 15.7, and Equation 15.6, and substitute into Equation 15.8. Simplifying yields a first-order difference equation in K_t :

$$K_{2t} = \left[1 + (1 - \beta) \left(\frac{w^s + \rho}{\beta} \right)^{\frac{-\beta}{1-\beta}} \right] K_{2t-1} = \delta K_{2t-1}$$

Since the coefficient on K_{2t-1} is larger than unity ($\delta > 1$), the system is unstable and K_{2t} will grow exponentially. It can further be seen that the subsistence wage w^S is negatively related to the speed of capital accumulation:

$$\frac{\partial \delta}{\partial w^s} = - \left(\frac{w^s + \rho}{\beta} \right)^{\frac{-1}{1-\beta}} < 0.$$

We can calculate the growth rate of the capital stock using the simulation:

```
# Calculate growth rate of capital stock analytically (for baseline)
1+(1-beta)*((w1[1,s]+rho)/beta)^(-beta/(1-beta)) - 1
```

```
[1] 0.02589882
```

```
# Compare with numerical solution (for baseline)
K[1,s+1]/K[1,s] - 1
```

```
[1] 0.02589882
```

i Python code

```
# Calculate growth rate of capital stock analytically (for baseline)
1 + (1 - beta) * ((w1[0, s] + rho) / beta) ** (-beta / (1 - beta)) - 1

# Compare with numerical solution (for baseline)
K[0, s + 1] / K[0, s] - 1
```

15.6 References

Additional Online Resources

Below you can find some related free online resources for macroeconomic model simulation.

Economic Modelling in General

- [Sayama \(2015\)](#) is a free introductory textbook for the modelling and simulation of complex systems in *Python*
- [QuantEcon](#) provides online intermediate and advanced resources for economic modelling and data analysis in *Python* and *Julia*

Stock-Flow Consistent Modelling

- Marco Veronese Passarella's website provides *R* codes for most chapters of [Monetary Economics: An Integrated Approach to Credit, Money, Income, Production and Wealth](#) by Wynne Godley and Marc Lavoie, and more
- [SFC Models](#) provides resources on stock-flow consistent modelling

Agent-Based Modelling

- Alessandro Caiani's website provides *R* codes for simple agent-based models, including those in the introductory book [Economics with Heterogeneous Interacting Agents: A Practical Guide to Agent-Based Modeling](#) by Alessandro Caiani, Alberto Russo, Antonio Palestrini and Mauro Gallegati

Coding

- [Coding for Economists](#) provides a guide for coding in *Python*

- Anthony, Martin, and Michele Harvey. 2012. *Linear Algebra: Concepts and Methods*. Cambridge UK: Cambridge University Press.
- Bhaduri, Amit, and Stephen Marglin. 1990. "Unemployment and the Real Wage: The Economic Basis for Contesting Political Ideologies." *Cambridge Journal of Economics* 14 (4): 375–93.
- Blanchard, Olivier, and David R. Johnson. 2013. *Macroeconomics, 6th Edition*. Pearson.
- Blecker, Robert A., and Mark Setterfield. 2019. *Heterodox Macroeconomics. Models of Demand, Distribution and Growth*. Edward Elgar.
- Carlin, Wendy, and David Soskice. 2014. *Macroeconomics. Institutions, Instability, and the Financial System*. Oxford University Press.
- Chiang, Alpha C, and Kevin Wainwright. 2005. *Fundamental Methods of Mathematical Economics*. 4th ed. New York: McGraw-Hill Education.
- Dutt, Amitava Krishna. 2018. "Some Observations on Models of Growth and Distribution with Autonomous Demand Growth." *Metroeconomica* 70 (2): 288–301. <https://doi.org/10.1111/meca.12234>.
- Fennell, Peter G., David J. P. O'Sullivan, Antoine Godin, and Stephen Kinsella. 2015. "Is It Possible to Visualise Any Stock Flow Consistent Model as a Directed Acyclic Graph?" *Computational Economics* 48 (2): 307–16. <https://doi.org/10.1007/s10614-015-9521-8>.
- Foley, Duncan K. 2006. *Adam's Fallacy. A Guide to Economic Theology*. Cambridge, MA / London: Harvard University Press.
- Fontana, Giuseppe, and Mark Setterfield. 2009. "A Simple (and Teachable) Macreconomic Model with Endogenous Money." In *Macroeconomic Theory and Macroeconomic Pedagogy*, edited by Giuseppe Fontana and Mark Setterfield, 144–68. Basingstoke ; New York: Palgrave Macmillan.
- Froyen, Richard T. 2005. *Macroeconomics. Theories and Policies. 8th Edition*. Pearson Education.
- Galí, Jordi. 2018. "The State of New Keynesian Economics: A Partial Assessment." *Journal of Economic Perspectives* 32 (3): 87–112. <https://doi.org/10.1257/jep.32.3.87>.
- Gandolfo, Giancarlo. 2009. *Economic Dynamics. Study Edition. 4th Edition*. Springer.
- Garín, Julio, Robert Lester, and Eric Sims. 2021. *Intermediate Macroeconomics*. Draft Version 3.0.1. https://juliogarin.com/files/textbook/GLS_Intermediate_Macro.pdf.
- Hein, Eckhard. 2014. *Distribution and Growth After Keynes: A Post-Keynesian Guide*. Cheltenham: Edward Elgar.
- Hicks, A. R. 1937. "Mr. Keynes and the "Classics": A Suggested Interpretation." *Econometrica* 5 (2): 147. <https://doi.org/10.2307/1907242>.
- Lavoie, Marc. 2006. *Introduction to Post-Keynesian Economics*. Palgrave Macmillan.
- . 2014. *Post-Keynesian Economics: New Foundations*. Cheltenham; Northampton, MA: Edward Elgar.
- . 2022. *Post-Keynesian Economics. New Foundations*. 2nd ed. Edward Elgar.
- Lewis, W. A. 1954. "Economic Development with Unlimited Supplies of Labour." *The Manchester School* 22 (2): 139–91. <https://doi.org/10.1111/j.1467-9957.1954.tb00021.x>.
- Pasinetti, Luigi L. 1960. "A Mathematical Formulation of the Ricardian System." *The Review of Economic Studies* 27 (2): 78–98. <https://doi.org/10.2307/2296129>.

- Samuelson, Paul A. 1939. "Interactions between the Multiplier Analysis and the Principle of Acceleration." *The Review of Economics and Statistics* 21 (2): 75–78. <https://doi.org/10.2307/1927758>.
- Sayama, Hiroki. 2015. *Introduction to the Modeling and Analysis of Complex Systems*. Open SUNY Textbooks, Milne Library.
- Serrano, Franklin. 1995. "Long Period Effective Demand and the Sraffian Supermultiplier." *Contributions to Political Economy* 14 (1): 67–90. <https://doi.org/10.1093/oxfordjournals.cpe.a035642>.
- Todaro, Michael P., and Stephen C. Smith. 2015. *Economic Development, 12th Edition*. Pearson.