Research Methods (MSc Programme), 2018

Introduction to MATLAB 3

Numerical techniques in MATLAB: differential equations and non-linear dynamics

Piotr Z. Jelonek

May 23, 2018

1 Differential Equations

1.1 Ordinary linear equation: An example

Let us start with the following simple example:

$$\frac{dx}{dt} = ax + b. \tag{1}$$

In order to solve eq. (1) on a computer we will need to transit from *continuous* time to *discrete* time. This is because computers can not faithfully represent real numbers. Start by rewriting the initial formula as:

$$dx = (ax + b)\,dt.$$

Next assume that the time is a discrete index with: n ascending values, initial value equal to zero, and terminal value equal to T:

$$0 = t_1 < t_2 < .... < t_{n-1} < t_n = T.$$

Discrete time counterparts of continuous time differential operators dx, dt are difference operators:

$$\Delta x(t_{k+1}) = x(t_{k+1}) - x(t_k), \qquad \Delta t_{k+1} = t_{k+1} - t_k.$$

If we replace differentials with differences at time t_{k+1} and assume that x's on the right hand side are *delayed* in time:

$$\Delta x(t_{k+1}) = (ax(t_k) + b)\,\Delta t_{k+1} \quad \equiv \quad x(t_{k+1}) - x(t_k) = (ax(t_k) + b)(t_{k+1} - t_k) \quad \equiv$$
$$\equiv \quad x(t_{k+1}) = x(t_k) + (ax(t_k) + b)(t_{k+1} - t_k).$$

This calculation motivates the following iterative formula:

$$x(t_k) = \begin{cases} x(0) & \text{for } k = 1 \\ x(t_{k-1}) + (ax(t_{k-1}) + b)(t_k - t_{k-1}) & \text{for } k \in \{2, ..., T\}. \end{cases} \tag{2}$$

In the equation above the increment of time is called a **time step**, it is often convenient to assume it is constant and thus set $t_{k+1} - t_k := d$. The process of switching from the differential equation in continuous time to difference equation in discrete time (which can be effectively solved by substitution) is called the **discretization**. If the right hand side of eq. (1) is continuous, then the iterative formula (2) may provide a good approximation of the solution to the original differential equation, if only the time steps are small enough.

In order to check whether this approximation is *accurate*, we will need to compare it with the exact analytical solution. To solve eq. (1) first rewrite it, and next take *antiderivatives* of both sides:

$$\frac{1}{ax+b}\,dx = dt \quad \equiv \quad \int \frac{1}{ax+b}\,dx = \int 1\,dt \quad \equiv \quad \int \frac{d}{dx}\left(\frac{1}{a}\ln(ax+b)\right)dx = \int \frac{d}{dt}t\,dt \quad \equiv$$
$$\equiv \quad \frac{1}{a}\ln(ax+b) + c = t \quad \equiv \quad \ln(ax+b) = a(t-c) \quad \equiv \quad ax + b = e^{a(t-c)} \quad \equiv$$
$$\equiv \quad x(t) = \frac{1}{a}\left(e^{a(t-c)} - b\right).$$

This is a general solution (x as a function of t). Since antiderivative is unambiguous up to a constant term, constant c can by anything. We may want to check if this solutions is correct, indeed:

$$\frac{d}{dt}\frac{1}{a}\left(e^{\,a(t-c)}-b\right)=e^{\,a(t-c)}=a\cdot\frac{1}{a}\left(e^{\,a(t-c)}-b+b\right)=a\cdot\frac{1}{a}\left(e^{\,a(t-c)}-b\right)+b=ax(t)+b.$$

It is often convenient to express c in terms of x at $t=0$ (initial value of x):

$$x(0)=\frac{1}{a}\left(e^{\,a(0-c)}-b\right)\quad\equiv\quad ax(0)+b=e^{\,-ac}\quad\equiv\quad\ln\left(ax(0)+b\right)=-ac\quad\equiv$$

$$\equiv\quad c=-\frac{1}{a}\ln\left(ax(0)+b\right).$$

If we plug the formula for c back into the general solution, we obtain:

$$x(t)=\frac{1}{a}\left(e^{\,a(t+\frac{1}{a}\ln\left(ax(0)+b\right))}-b\right)=\frac{1}{a}\left((ax(0)+b)e^{\,at}-b\right)=\left(x(0)+\frac{b}{a}\right)e^{\,at}-\frac{b}{a}. \qquad (3)$$

Now the solution depends on: parameters a and b and the initial value $x(0)$.

We will solve our initial problem on a computer for the time step $d=0.01$, this example is provided as *'ode1.m'* script:

```
clear; clc;

% equation parameters
a=1/pi
b=(exp(1))/2
x0=1.234

% trajectory parameters
T=3;           % <- terminal time
d=0.01;        % <- grain of time (quantum)

% number of periods
n=1+T/d;

% vector of time
t=linspace(0,T,n)'; % <- same as: t=d*(cumsum(ones(n,1))-1);

% solution
x=zeros(n,1); % <- here we will save the output
x(1)=x0;      % <- the initial value

for i=2:n
    x(i)=x(i-1)+(a*x(i-1)+b)*d;
end

plot(t,x)
axis([0 T 0 12])
```

Now we will repeat this exercise with the following modification. First we will solve the eq. (1) for $d = 0.25$ (for $n = 12 + 1$ point in time), then for $d = 0.001$ ($3000 + 1$ points), next we will compute analytical (exact) solutions for 3001 points in time. Finally, we will plot all the solutions on one graph. This example is provided as *'ode2.m'* script:

```
clear; clc;

% parameters
a=1/pi
b=(exp(1))/2
x0=1.234

%%%%%%%%%%%%%%%%%%%% Time step d = 0.25 %%%%%%%%%%%%%%%%%%%
d=0.25;             % <- grain of time (quantum)

% trajectory parameters
T=3;                % <- terminal time

% number of periods
n=1+T/d;

% vector of time
t=linspace(0,T,n)'; % <- same as: t=d*(cumsum(ones(n,1))-1);

% solution
x=zeros(n,1);       % <- here we will save the output
x(1)=x0;            % <- the initial value
for i=2:n
    x(i)=x(i-1)+(a*x(i-1)+b)*d;
end

%%%%%%%%%%%%%%%%%%% Time step d = 0.001 %%%%%%%%%%%%%%%%%%
d=0.001;            % <- grain of time (quantum)

% number of periods
n=1+T/d;

% vector of time
s=linspace(0,T,n)'; % <- same as: t=d*(cumsum(ones(n,1))-1);

% solution
y=zeros(n,1);       % <- here we will save the output
y(1)=x0;            % <- the initial value
for i=2:n
    y(i)=y(i-1)+(a*y(i-1)+b)*d;
end
```

```
%%%%%%%%%%%%%%%%%%%  Analytic solution %%%%%%%%%%%%%%%%%%%
z=(x0+(b/a))*exp(a*s)-(b/a);

%%%%%%%%%%%%%%%%%%%%% All one one plot %%%%%%%%%%%%%%%%%%%
plot(s,z,'b--',s,y,'k-',t,x,'r+')
legend('Solution','Approx. (d=0.001)','Approx. (d=0.25)','Location','NorthWest')
axis([0 T 0 12])
```

The output chart is provided below:

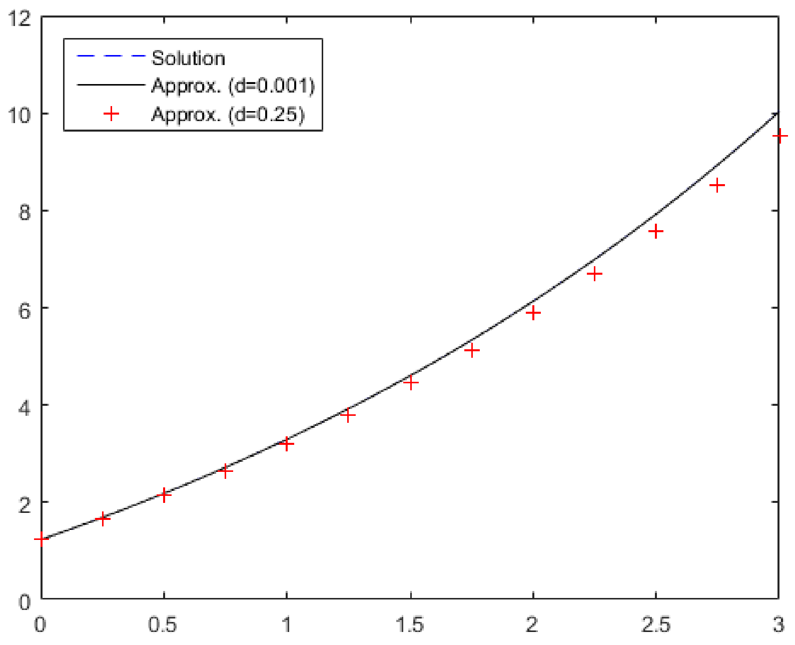


Figure 1: Analytical solution is represented by a dashed blue line, numerical approximation (with $d = 0.001$) – by a solid black line, another numerical approximation ($d = 0.25$) – by red '+' symbols.

When I produced this graph, I have first plotted the blue dashed line (representing analytical solution), next the black solid line (approximation, $d = 0.001$), and finally the red '+' symbols (approximation, $d = 0.25$).

Question: what happened to the blue dashed line?

Note: we have not used at any point the fact that equation (1) is linear, or that its trajectories are *smooth* (differentiable an arbitrary number of times). Hence a similar approach could be used to approximate solutions to non-linear, non-ordinary differential equations which, in general, remain unknown.

Why is it useful: This example illustrates how easy it may be to solve differential equations on a computer. The most difficult thing we did here was to write a *for* loop, in multivariate case it might also be convenient to store the approximated solutions in a matrix (instead of multiple vectors). This approach typically works fine, but for larger systems it is tedious – so the examples that follow will be solved differently. A situation when we would need to take this pedestrian approach: input variables are bounded (e.g. the population ratios, contained in $[0, 1]$ interval), and in result trajectories of solutions are non-differentiable (kinked).

1.2 Equations with delays: logistic equation

We could also solve equations with dependence delayed in time. A **logistic equation** is a model of population dynamics, useful in environmental economics. It depicts the growth of a population when the maximal population size is limited by *environmental capacity* C.

$$\frac{dp}{dt} = r \cdot p(t) \left(1 - \frac{p(t)}{C}\right), \qquad \text{with } r, C > 0. \tag{4}$$

Motivation: In equation (4) for a very small population size the term $(1 - p(t)/C) \approx 1$ and population growth is (approximately) exponential. When the population size approaches the value of C, then $(1 - p(t)/C) \approx 0$ and the growth is halted. This is because the environment has no more resources to accommodate a larger population.

Logistic equation has a known analytic solution, which takes the form:

$$p(t) = \frac{C}{1 + \left(\frac{C}{p(0)} - 1\right) e^{-rt}}.$$

Since we do not derive this formula, let us at least check if it is correct:

$$\frac{d}{dt}p(t) = (-1)\frac{C}{\left[1 + \left(\frac{C}{p(0)} - 1\right) e^{-rt}\right]^2}(-r)\left(\frac{C}{p(0)} - 1\right) e^{-rt} =$$

$$= r \cdot \frac{C}{\left[1 + \left(\frac{C}{p(0)} - 1\right) e^{-rt}\right]^2}\left[1 + \left(\frac{C}{p(0)} - 1\right) e^{-rt} - \frac{1}{C}C\right] = r \cdot p(t)\left(1 - \frac{p(t)}{C}\right).$$

We have shown by substitution that the formula above is *some solution* to the equation (4), but we have not shown it is a *general* solutions.

Now we will approximate the solutions to the logistic equation using a build-in MATLAB ordinary differential equations **solver**. What solvers are doing is exactly what we did in the previous subsection, but automatically (so you do not need to write a loop) and in general *better*. Solvers may use different *discretization schemes* and varying time lengths more suitable for a given problem, they also allow to control a magnitude of the approximation errors. We will be using the **ode45()** solver, which is appropriate for the systems of *non-stiff* ordinary differential equations. An equation is non-stiff if it can be solved with a non-stiff approximation method[1]. To learn how to use this solver type:

```
help ode45
```

in the MATLAB command window. The basic syntax of **ode45()** solver is as follows:

```
[t,x]=ode45(@system_dx,tspan,x0);
```

[1]If it sounds like a 'butterly butter' it is because it is a 'butterly butter'.

Input vector **tspan** is required to have the initial time as it s first, and the terminal time as its second entry. Input vector **x0** represents the initial point of a trajectory, it contains the starting values of all the variables. Both vectors can be either row vectors, or column vectors (whichever you find more convenient). The monkey *'@'* symbol tells MATLAB that whatever follows is **a name of a function**. Hence *'system_dx'* is a name of a function. This function is required to have *tspan* and a vector x, representing the current point of the output trajectory, as its first and second (exactly in that order) input variables. Function **system_dx()** is required to output a column vector of derivatives of our system of equations, evaluated in point x. Output vector **t** represents time. Since these points were selected by a solver, the values of t do not need to be equally spaced. Output matrix x contains the values of all the points of the trajectory, evaluated for the times provided by t. The first column of x is your first variable, the second column is your second variable etc.

The following code solves the logistic equation and compares this approximation to the analytic solution. It is is provided as *'logistic.m'* script:

```
clear; clc;

% system parameters
C=500;       % <- environmental capacity
r=0.1;       % <- growth rate

% trajectory parameters
x0=1;        % <- initial population size
T=100;       % <- terminal time
tspan=[0 T]; % <- time span

% time scale
s=0:0.1:T;   % <- same as: s=linspace(0,T,1+T/0.1);
rowss=length(s);

% analytic solution to logistic differential equaiton
x=zeros(rowss,1);
for k=1:rowss
    x(k)=C/(1+(C/x0-1)*exp(-r*s(k)));
end

% numerical solution using ode45 solver, 'logistic_dx' is a function
% which outputs a vector of derivatives
[t,y]=ode45(@logistic_dx,tspan,x0);

% compare the solutions: theoretic and numeric
plot(s,x,t,y,'or')
xlabel('TIME (IN YEARS)')
ylabel('POPULATION SIZE')
```

The **logistic_dx()** function is defined as:

```
function dx=logistic_dx(tspan,x)
    r=0.1; % <- defining parameters here is not a good practice
    C=500;
    dx=r*x*(1-x/C);
end
```

The chart produced by the script is presented below:

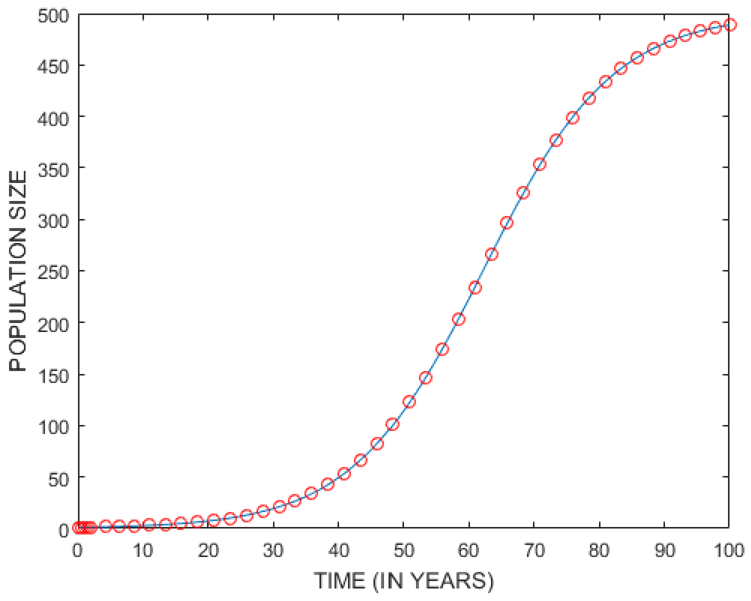


Figure 2: Analytical solution to *logistic* equation is denoted with a blue solid line, numerical approximation is marked by red circles.

For the sake of simplicity of the syntax I have defined parameters of this system *inside* the function that outputs the derivative. That is not a good practice, in the next example we will do it in a much neater way. We will also control the quality of the generated approximation.

A more realistic extension of this mode is logistic equation **with a delay**:

$$\frac{dp}{dt} = r \cdot p(t-s)\left(1 - \frac{p(t)}{C}\right), \qquad \text{with } r, C, s > 0. \tag{5}$$

Motivation: Only *adult* members of a population can produce offsprings. Hence population growth rate should only depend on a number of adult members (who were born at least s units of time ago). Since some of them might have died before reaching the adulthood, r should be corrected for an attrition rate. As children also need to eat, environmental capacity is being exhausted by the current population size.

Equation (5) is a simple example of a **Delayed Differential Equation (DDE)**. These type of equations requires different solving methods than the Ordinary Differential Equations and are not covered in basic differential equations courses.

To solve this example we will use the **dde23()** solver, dedicated to solving the delayed differential equations. To learn how to use this solver type:

```
help dde23
```

in the MATLAB command window. The basic syntax of this solver is as follows:

```
sol=dde23(@system_dx,lags,hist,tspan);
```

but now what we are going to use is its more advanced version:

```
sol=dde23(@(t,x,z) system_dx(t,x,z,params),lags,hist,tspan,options);
```

Input vector **lags** indicates the required lags of all the variables. Variable **hist** codes the history of our variable(s) prior the initial time. It may be equal either to *a scalar* value, or to *a vector* of values, or be given by *a function.* Input vector **tspan** is required to have the initial time as its first, and the terminal time as its second entry. The **options** variable denotes a MATLAB **structure**, that is – a type of data that consists of a number of variables of different types, bundled together for the sake of convenience. Different fields (elements) of this variable might be conveniently set by the **odeset()** function (explained later). The *'system_dx'* above is again a name of a function, which is signalized by the monkey (*'@'*) sign. This function has four parameters: current time **t**, vector of current values of all variables in the system **x**, matrix of lagged values **z** (lags of each variable are provided in a separate column), and a vector of parameters **params**. Out of these four parameters the first three are the required by the solver, we tell this to MATLAB explicitly using the *'@(t,x,z)'* syntax. The output **sol** structure characterizes the solution. Its two most important fields are the vector of times and the matrix of approximated solutions, which may be conveniently acessed through the following assignments:

```
t=sol.x;     x=sol.y;
```

The code displayed below solves the *delayed* logistic equation, it is provided as *'dlogistic.m'* script:

```
clear; clc;

% system parameters
C=500;        % <- environmental capacity
r=0.1;        % <- growth rate
lags = 16;    % <- delay in time (could be a vector)

% trajectory parameters
T=100;        % <- terminal time
tspan=[0 T]; % <- time span
hist=1;       % <- population size up to initial time

% solver parameters
small=0.000001;   % <- definition of a small number

% approximation
options = odeset('RelTol',small,'AbsTol',small);
```

```
% sol=dde23(@dlogistic_dx,lags,hist,tspan);
sol=dde23(@(t,x,z) dlogistic_dx(t,x,z,r,C),lags,hist,tspan,options);

% acessing elements of the output structure
t=sol.x';
x=sol.y';

plot(t,x,'ro')
xlabel('TIME (IN YEARS)')
ylabel('POPULATION SIZE')

n=size(t,1)
```

The **dlogistic_dx()** function is defined as:

```
function dx=dlogistic_dx(t,x,z,r,C)
    dx=r*z(:,1).*(1-x(:,1)/C);
end
```

The **odeset()** function modifies the **options** structure and simultaneously makes it accessible in the *Workspace* window. It is often used to modify the required *accuracy* of the solutions. That is, we may require approximation errors of variable x_i at each time t to be small enough to fulfil

$$\varepsilon(i) \leq \max(\mathrm{RelTol} \cdot x_i, \mathrm{AbsTol}).$$

Parameter *'RelTol'* represents the **relative error tolerance**. In example, RelTol=0.01 means that we never make a (relative) error larger than 1%. Parameter *'AbsTol'* represents the **absolute error tolerance**. Hence AbsTol = 0.001 means that we never make an (absolute) error larger than 0.001. Both tolerance parameters may be set either globally, for all the variables, or we may use a different tolerance for each variable. We may also use the odeset() function with the **ode45()** solver, in such an instance we would call it like in the line(s) below (**options** becomes the last parameter of the **ode45()** function):

```
% approximation
options = odeset('RelTol',small,'AbsTol',small);

% numerical solution using ode45 solver
[t,y]=ode45(@logistic_dx,tspan,x0,options);
```

The chart produced by the script is presented on the next page.

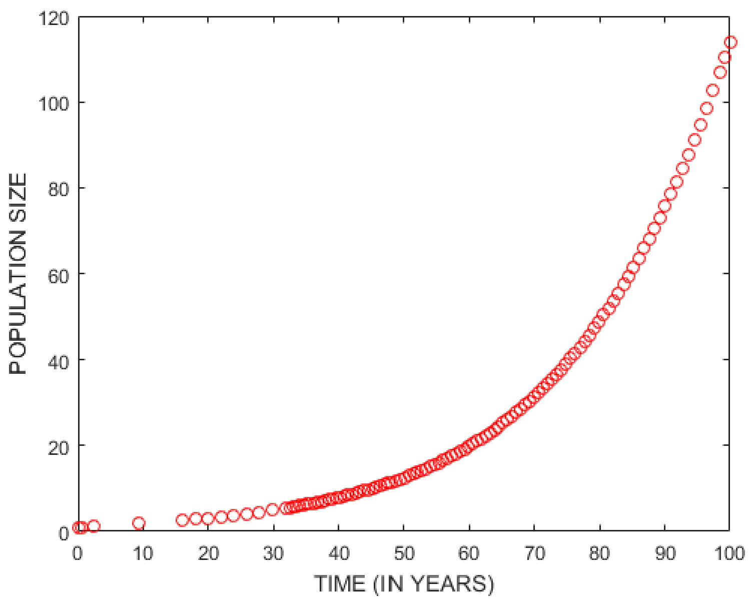


Figure 3: Numerical approximation of the solution is marked by red circles.

Why is it useful: in economics we often use simultaneous differential equations. However, most of policies require transmission time, while some other variables (like: expectation) are forward looking – the time lag/forward structure matters. Investigating how time structure might affect policy transmission in classical models might be an interesting path to pursue in a dissertation.

Fun fact: in an (otherwise interesting) maths project available here you may breed straight after you were born, but you do not need to eat until you are 18.

1.3 Deterministic chaos: Lorenz attractor

From Lorenz system entry on Wikipedia: a non-linear, deterministic, three-dimensional system of differential equations, developed in 1963 by Edward Lorenz as a mathematical model of *atmospheric convection*. Also useful in modelling: laser, electric circuits, chemical reactions and forward osmosis. Its most notable interesting feature is a set of chaotic solutions, known as the Lorenz attractor, obtained for certain: parameter values and initial conditions.

The three Lorenz equations are presented below.

$$\frac{d}{dt}x(t) = -\sigma\left(x(t) - y(t)\right),$$
$$\frac{d}{dt}y(t) = x(t)\left(\rho - z(t)\right) - y(t),$$
$$\frac{d}{dt}z(t) = x(t)y(t) - \beta z(t).$$

Question: Can you guess how solutions of this systems will look on a graph?

Numerical approximations of the solutions to the Lorenz system are generate by *'lorenz.m'* script, provided below.

```
clear; clc;

% system parameters
sigma=10;
rho=28;
beta=8/3;

% trajectory parameters
x0=[1 10 5];      % <- starting point
tspan=[0 40];     % <- time span (from t=0 up to t=40)

% solver parameters
small=0.000001;   % <- definition of a small number

% approximation
options = odeset('RelTol',small,'AbsTol',small);
[t,x] = ode45(@(tspan,x) lorenz_dx(tspan, x, sigma, rho, beta), tspan, x0, options);

plot3(x(:,1),x(:,2),x(:,3));
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Lorenz attractor');

n=size(t,1)       % <- number of points in time (optimized by solver)
```

The **lorenz_dx()** function is defined as:

```
function dx = dx_lorenz(tspan, x, sigma, rho, beta)
% tspan is a parameter in general required by the solver,
% here it remains unused
    dx = zeros(3,1);
    dx(1) = -sigma*(x(1)-x(2));
    dx(2) = x(1)*(rho - x(3)) - x(2);
    dx(3) = x(1)*x(2) - beta*x(3);
end
```

Why is it useful: Lorenz attractor is a simple example of a system which displays a property, known as *deterministic chaos*. The chaotic jumps between the two adjoint discs (the shape of this chart is sometimes called a *butterfly*) are also susceptible to the *Butterfly effect*, which describes high sensitivity of solutions to the systems of **Partial Differential Equations** to their initial conditions (this is why forecasting weather is hard). Could we face (somehow) similar problems in the systems of differential equations, describing an economy?

The chart produced by the script is presented on the next page.

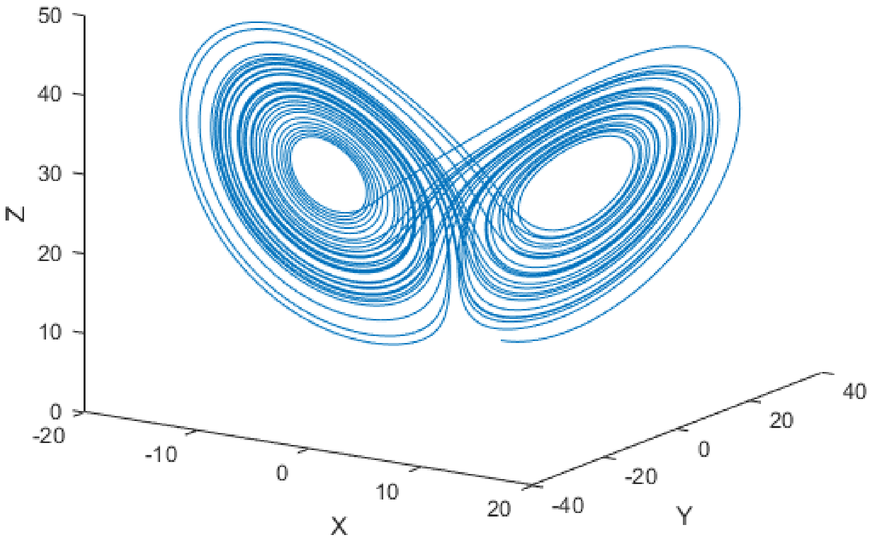


Figure 4: Lorenz attractor.

1.4 Instability in economic systems: Minsky model(s)

The Goodwin model (for the reference see p. 224 in Keen (2013a), or Steven Keen's lecture) is able to replicate **GDP fluctuations** and **employment cycles**.

Values of parameters:

$$\alpha = 0.02, \quad \beta = 0.01, \quad c = 4.8, \quad d = 5, \quad \gamma = 0.01, \quad \nu = 3,$$

system of equations (driving the entire dynamics):

$$\begin{aligned}
\frac{d}{dt}L(t) &= \left(\frac{1}{\nu}\left(1 - \frac{w(t)}{a(t)}\right) - \gamma - \alpha\right) \cdot L(t),\\
\frac{d}{dt}w(t) &= \left(d \cdot \frac{L(t)}{N(t)} - c\right) \cdot w(t),\\
\frac{d}{dt}a(t) &= \alpha \cdot a(t),\\
\frac{d}{dt}N(t) &= \beta \cdot N(t),
\end{aligned}$$

initial values:

$$L(0) = 300, \quad w(0) = 0.95, \quad a(0) = 1, \quad N(0) = 300,$$

definitions and identities:

$$Y(t) = a(t) \cdot L(t), \quad K(t) = \frac{1}{\nu}Y(t), \quad \Pi(t) = Y(t) - w(t) \cdot L(t), \quad I(t) = \Pi(t).$$

The code approximating the solutions to Godwin model is provided as *'minsky_I.m'* script:

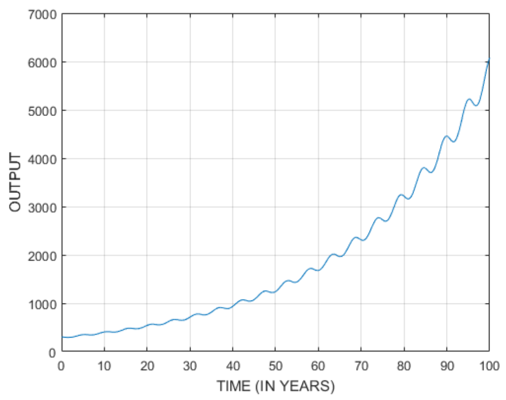
```
clear; clc;

% system parameters
alpha=0.02;      % <- per annum
beta=0.01;       % <- per annum
c=4.8;
d=5;
gamma=0.01;      % <- per annum
nu=3;

% initial values
L0=300;          % <- initial labour force
w0=0.95;         % <- initial wages
a0=1;            % <- initial technology
N0=300;          % <- initial population

% trajectory parameters
x0=[L0 w0 a0 N0];
T=100;           % <- terminal time
tspan=[0 T];     % <- time span (from t=0 up to t=T)

% solver parameters
small=0.00001;   % <- definition of a small number

% approximation
options = odeset('RelTol',small,'AbsTol',small);
[t,x] = ode45(@(tspan,x) minsky_I_dx(tspan,x,alpha,beta,c,d,gamma,nu), tspan, x0);

% mapping solutions of ode to economy
L=x(:,1);
w=x(:,2);
a=x(:,3);
N=x(:,4);

Y=a.*L;
K=Y/nu;
P=Y-w.*L;
I=P;

employment_rate=100*L./N; % <- in % points
wage_share=100*(w.*L)./Y; % <- in % points
```

```
% here choose your favourite chart
chart=1;

if chart==1
    % output (cyclical)
    plot(t,Y)
    xlabel('TIME (IN YEARS)')
    ylabel('OUTPUT');
    grid on;
elseif chart==2
    % employment cycles
    plot(employment_rate,wage_share)
    xlabel('EMPLOYMENT RATE')
    ylabel('WAGES TO OUTPUT')
    axis([90 105 60 120])
    grid on;
end
```

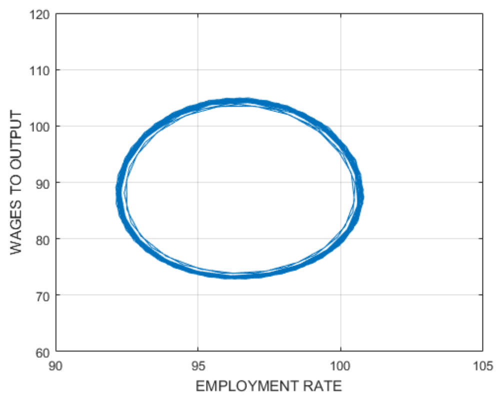
The **minsky_I_dx()** function is defined as:

```
function dx = minsky_I_dx(tspan,x,alpha,beta,c,d,gamma,nu)
    dx=zeros(4,1);
    dx(1)=( (1/nu)*(1-x(2)/x(3))- gamma - alpha)*x(1);
    dx(2)=(d*(x(1)/x(4))-c)*x(2);
    dx(3)=alpha*x(3);
    dx(4)=beta*x(4);
end
```

The charts produced by the script are presented below:



(a) Output fluctuates.



(b) Employment is cyclic.

Figure 5: The results following from the approximated solutions to the Godwin model.

Minsky II model (for the reference see pp. 225-226 in Keen (2013a), or Steven Keen's lecture) has two regimes: stable, where the economy converges to an equilibrium, and unstable, where the crisis occurs.

Non-linear functions:

$$\begin{aligned}
\mathrm{GenExp}(v; x, y, s, m) &:= (y-m)\cdot e^{\,s\cdot(v-x)/(y-m)} + m,\\
W_f(v) &:= \mathrm{GenExp}(v; x_w, y_w, s_w, m_w),\\
I_f(v) &:= \mathrm{GenExp}(v; x_i, y_i, s_i, m_i),
\end{aligned}$$

values of parameters:

$$\alpha = 0.02, \quad \beta = 0.01, \quad \gamma = 0.01, \quad \nu = 3, \quad r = 0.05,$$

$$x_p = 0.05, \quad y_p = 0.05, \quad s_p = 1.75, \quad m_p = 0,$$

$$x_l = 0.95, \quad y_l = 0, \quad s_l = 0.5, \quad m_l = -0.01,$$

definitions and identities:

$$L(t) = Y(t)/N(t), \quad \Pi(t) = Y(t) - w(t)\cdot L(t) - r\cdot D(t), \quad \lambda(t) = \frac{Y(t)}{a(t)\cdot N(t)}$$

system of equations (driving the entire dynamics):

$$\begin{aligned}
\frac{d}{dt}Y(t) &= \left[\frac{1}{\nu} I_f\left(\frac{\Pi(t)}{\nu\cdot Y(t)}\right) - \gamma\right]\cdot Y(t),\\
\frac{d}{dt}w(t) &= P_f\left(\lambda(t)\right)\cdot w(t),\\
\frac{d}{dt}D(t) &= I_f\left(\frac{\Pi(t)}{\nu\cdot Y(t)}\right)\cdot Y(t) - \Pi(t),\\
\frac{d}{dt}a(t) &= \alpha\cdot a(t),\\
\frac{d}{dt}N(t) &= \beta\cdot N(t),
\end{aligned}$$

initial values of variables:

$$Y(0) = 300, \quad w(0) = 0.95, \quad D(0) = 0, \quad a(0) = 1, \quad N(0) = 300,$$

Note: Steven Keen says that there are some stock-flow problems with this model, and that **Minsky III** model that follows is better. The provided parameter values correspond to the stable regime.

Note to myself: tell them about rabbits, hats and why this equilibrium is an interesting property.

The code approximating the solutions to Minsky II model is provided as the *'minsky_II.m'* script:

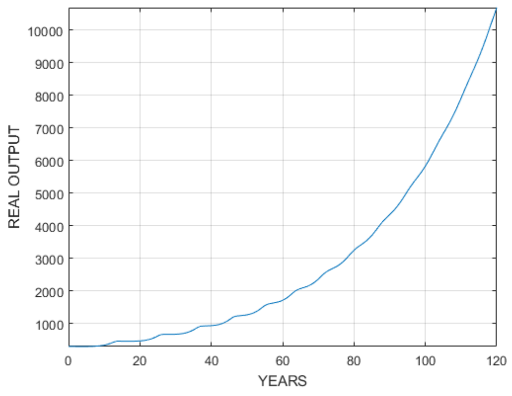
```
clear; clc;

% system parameters
alpha=0.02;      % <- per annum
beta=0.01;       % <- per annum
gamma=0.01;      % <- per annum
nu=3;
r=0.05;          % <- interest rate, exogeneous

% parameters of a non-linear function, capturing the dependence
% of investment on ratio of profit to capital
x_p=0.05;
y_p=0.05;
s_p=1.75;
m_p=0;

% parameters of a non-linear function, capturing the dependence
% of wage growth rate on employment (aka Phillips curve)
x_l=0.95;
y_l=0;
s_l=0.5;
m_l=-0.01;

% all parameters in one vector
params=[alpha beta gamma nu r x_p y_p s_p m_p x_l y_l s_l m_l];

% initial values
Y0=300;          % <- initial output
w0=0.95;         % <- initial wages
D0=0;            % <- initial debt
a0=1;            % <- initial technology
N0=300;          % <- initial population

% trajectory parameters
x0=[Y0 w0 D0 a0 N0];
T=120;           % <- terminal time
tspan=[0 T];     % <- time span (from t=0 up to t=T)

% solver parameters
small=0.000001;   % <- definition of a small number

% approximation
options = odeset('RelTol',small,'AbsTol',small);
[t,x] = ode45(@(tspan,x) minsky_II_dx(tspan,x,params), tspan, x0,options);
```

```
% mapping solutions of ode to economy
Y=x(:,1);
w=x(:,2);
D=x(:,3);
a=x(:,4);
N=x(:,5);

K=Y/nu;
L=Y./a;
P=Y-w.*L-r*D;
I=P;

% here choose your favourite chart
chart=1;

if chart==1
% PHillips curve
    l=linspace(0.9,1.01,1101);
    H=(y_l-m_l)*exp(s_l*(l-x_l)/(y_l-m_l))+m_l;
    plot(100*l,100*H);
    xlabel('EMPLOYMENT RATE (IN %)')
    ylabel('ANNUAL CHANGE IN REAL WAGE (IN %)');
    axis([90 101 -10 20]); grid on;
elseif chart==2
    % Investment function
    p=linspace(-0.05,0.11,1601);
    I=(y_p-m_p)*exp(s_p*(p-x_p)/(y_p-m_p))+m_p;
    plot(100*nu*p,100*I); % <- x has to be multiplied by nu
    xlabel('PROFIT (IN %)')
    ylabel('INVESTMENT (AS % OF OUTPUT)')
    axis tight; grid on;
elseif chart==3
    % Output
    plot(t,Y);
    xlabel('YEARS'); ylabel('REAL OUTPUT')
    axis tight; grid on;
elseif chart==4
    wage_share=(w.*L)./Y;
    employment_rate=L./N;
    debt_ratio=D./Y;
    x1=wage_share; x2=employment_rate; x3=debt_ratio;
    c = 1:length(t);      % <- number of colors
    h = surface([x1(:), x1(:)], [x2(:), x2(:)], [x3(:), x3(:)], ...
    [c(:), c(:)], 'EdgeColor','flat', 'FaceColor','none'); colormap(jet(numel(t)))
    xlabel('WAGE SHARE OF OUTPUT')
    ylabel('EMPLOYMENT RATE'); zlabel('DEBT RATIO'); grid on
end
```

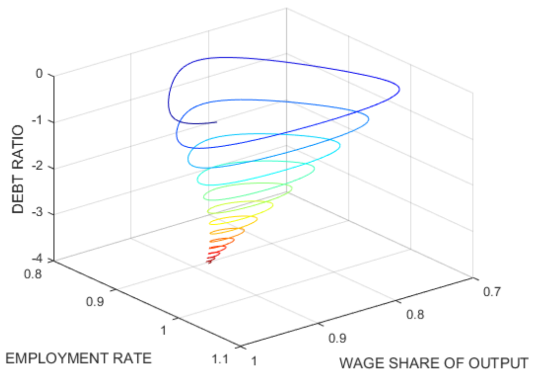
The **minsky_II_dx()** function is defined as:

```
function dx = minsky_II_dx(tspan,x,params)
    % reading paramaters
    alpha=params(1); beta=params(2); gamma=params(3); nu=params(4); r=params(5);
    x_p=params(6); y_p=params(7); s_p=params(8); m_p=params(9);
    x_l=params(10); y_l=params(11); s_l=params(12); m_l=params(13);

    % auxilaries
    L=x(1)/x(4);                                    % <- labour
    P=x(1)-x(2)*L - r*x(3);                         % <- profit
    p=P/(nu*x(1));                                  % <- profit to capital
    I=(y_p-m_p)*exp(s_p*(p-x_p)/(y_p-m_p))+m_p;     % <- investment as a function of profit
    l=x(1)/(x(4)*x(5));                             % <- employment rate
    H=(y_l-m_l)*exp(s_l*(l-x_l)/(y_l-m_l))+m_l;     % <- growth rate of wages as a fctn. of emp

    % derivative
    dx=zeros(5,1);
    dx(1)=x(1)*(I/nu - gamma );
    dx(2)=H*x(2);
    dx(3)=I*x(1)-P;
    dx(4)=alpha*x(4);
    dx(5)=beta*x(5);
end
```

The charts produced by the script are presented below:

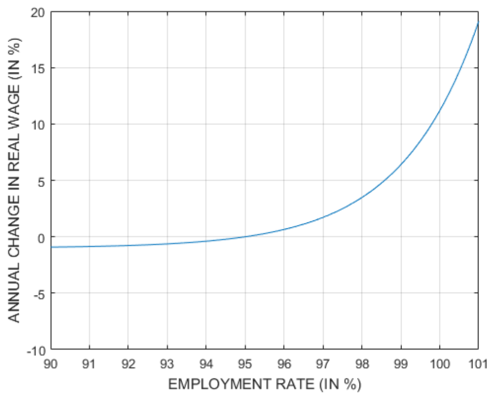


(a) Now output smoothes.

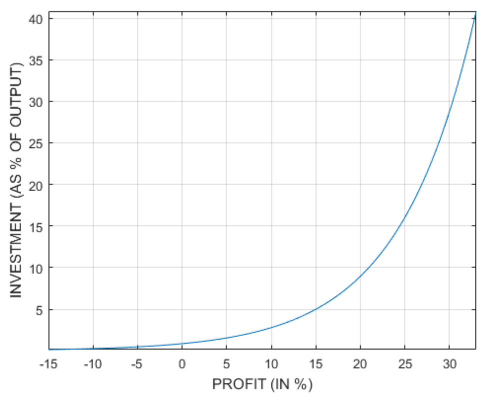


(b) Convergence to equilibrium.

Figure 6: The results following from the approximated solutions to the Minsky II model.



(a) Reaction of real wages to employment.



(b) Reaction of investment to profit.

Figure 7: The results following from the approximated solutions to the Minsky II model.

Minsky III model (straight from Steven Keen), this model replicates a deceptive **economic moderation**, followed by a vehement **structural crisis**, generated by the accumulation of private debt (consistent with the Minsky *financial instability* hypothesis).

Non-linear functions:

$$\begin{aligned}
\mathrm{GenExp}(v; x, y, s, m) &:= (y - m) \cdot e^{\,s \cdot (v-x)/(y-m)} + m, \\
W_f(v) &:= \mathrm{GenExp}(v; x_w, y_w, s_w, m_w), \\
I_f(v) &:= \mathrm{GenExp}(v; x_i, y_i, s_i, m_i),
\end{aligned}$$

values of parameters:

$$\alpha = 0.025, \quad \beta = 0.015, \quad \delta = 0.07, \quad \nu = 3, \quad r_b = 0.04, \quad s = 0.3, \quad \tau_p = 1, \quad \tau_i = 0.5,$$

$$x_w = 0.6, \quad y_w = 0.0, \quad s_w = 1, \quad m_w = -0.04,$$
$$x_i = 0.03, \quad y_i = 0.03, \quad s_i = 2.25, \quad m_i = 0,$$

definitions and identities:

$$\begin{aligned}
r(t) &:= r_b + \max\left(i(t), 0\right), \\
\pi(t) &:= 1 - \omega(t) - r(t) \cdot d(t), \\
f(t) &:= -\frac{1}{\tau_p}\left(1 - \frac{1}{1-s}\omega(t)\right), \\
g(t) &:= \frac{1}{\nu} I_f\left(\frac{\pi(t)}{\nu}\right) - \delta, \\
Y(t) &:= Y(0) \cdot \left(1 + \int_0^t g(s)\, ds\right),
\end{aligned}$$

system of equations (driving the entire dynamics):

$$\begin{aligned}
\frac{d}{dt}\lambda(t) &= \left[\frac{1}{\nu}I_f\left(\frac{\pi(t)}{s}\right) - (\alpha+\beta)\right]\cdot\lambda(t),\\
\frac{d}{dt}\omega(t) &= \left[W_f\left(\lambda(t)\right) - (\alpha + f(t))\right]\cdot\omega(t),\\
\frac{d}{dt}d(t) &= \left[I_f\left(\frac{\pi(t)}{s}\right) - \pi(t)\right] - \left[\frac{1}{\nu}I_f\left(\frac{\pi(t)}{s}\right) - \delta + f(t)\right]\cdot d(t),\\
\frac{d}{dt}i(t) &= -\frac{1}{\tau_i}\left(i(t) - f(t)\right).
\end{aligned}$$

initial values of variables:

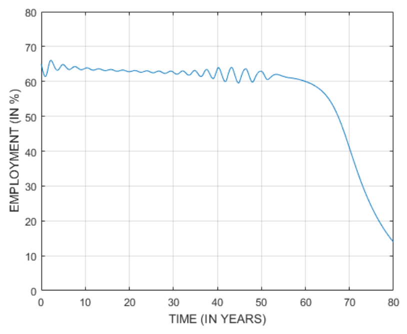
$$\lambda(0) = 0.65, \quad \omega(0) = 0.82, \quad d(0) = 0.5, \quad i(0) = 0.1,$$

The code which approximates the dynamics of this model is provided as *'minsky_III.m'* script, vector of derivatives for this system is calculated by the *'minsky_III_dx.m'* function.

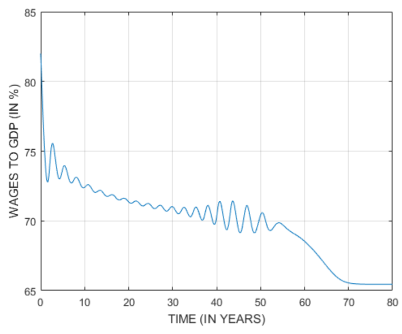
Why is it useful: this model does produce an interesting dynamics (so this is a good ODE exercise). It also comes from one of a very few economists who *predicted* the last major crisis (see: Keen (2013b)) and who expects further problems in China (check the hyperlink to his lecture).

Further extensions: endogenous money + banking system (relies on multiagent system, can not be conveniently simulated in MATLAB).

The graphs, produced by the script, are available on the next two pages.

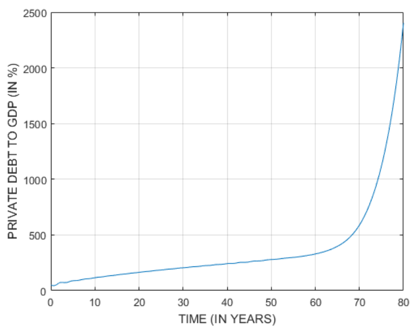


(a) Employment fluctuates and collapses.

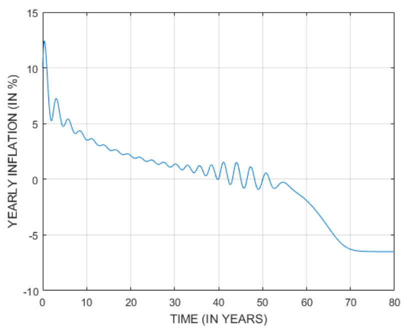


(b) Wages fluctuate and decline.

Figure 8: The results following from the approximated solutions to the Minsky III model.

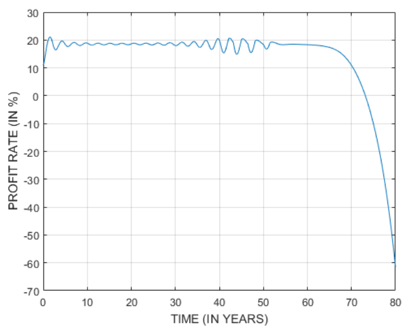


(a) Private debt explodes.

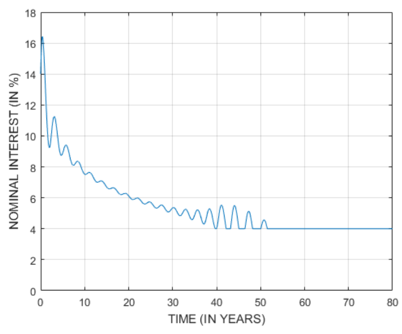


(b) Long-run deflation.

Figure 9: The results following from the approximated solutions to the Minsky III model.

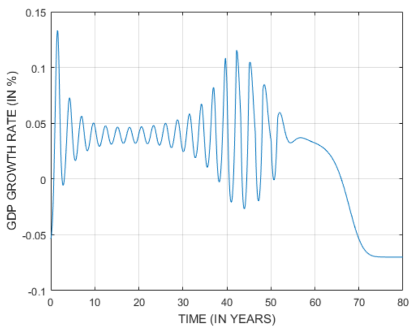


(a) Profit collapses.

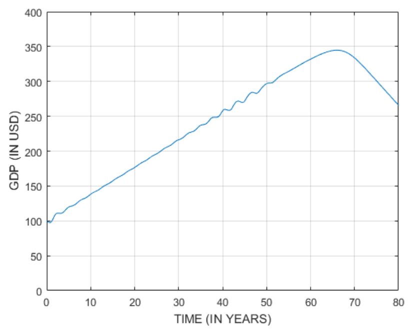


(b) Interest rates freeze.

Figure 10: The results following from the approximated solutions to the Minsky III model.

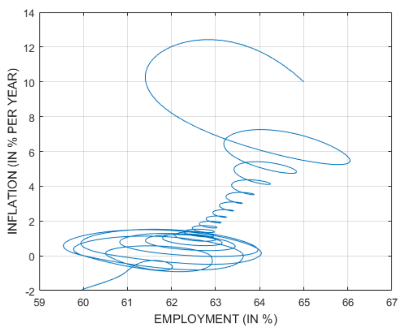


(a) Long-run negative growth.

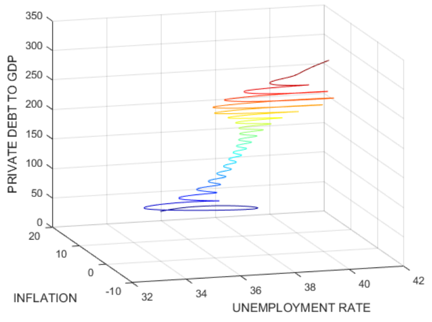


(b) GDP bends over.

Figure 11: The results following from the approximated solutions to the Minsky III model.

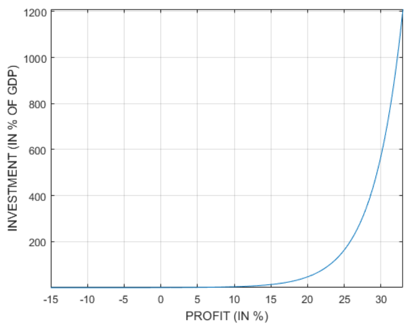


(a) This thing goes cinammon swirl.

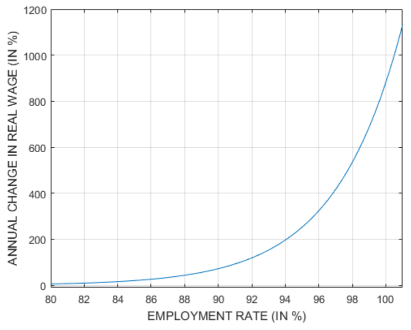


(b) From moderation to vehement crisis.

Figure 12: The results following from the approximated solutions to the Minsky III model.



(a) Investment as a function of profit.



(b) Phillips curve.

Figure 13: The results following from the approximated solutions to the Minsky III model.

The code approximating the solutions to Minsky II model is provided as the *'minsky_III.m'* script:

```
clear; clc;

%%%%%%%%%%%%%%% INITIAL VALUES
lambda0=0.65;
omega0=0.82;
d0=0.5;
i0=0.1;

%%%%%%%%%%%%%%% SYSTEM PARAMETERS
alpha=0.025;       % <- per annum
beta=0.015;        % <- per annum
delta=0.07;        % <- per annum
nu=3;
r_b=0.04;          % <- base interest rate, exogeneous
s=0.3;
tau_p=1;
tau_i=0.5;

% parameters of a non-linear function, capturing the dependence
% of
x_i=0.03;
y_i=0.03;
s_i=2.25;
m_i=0;

% parameters of a non-linear function, capturing the dependence
% of
x_w=0.6;
y_w=0;
s_w=1;
m_w=-0.04;

% all parameters in one vector
params=[alpha beta delta nu r_b s tau_p tau_i x_i y_i s_i m_i x_w y_w s_w m_w];

%%%%%%%%%%%%%%% TRAJECTORY PARAMETERS
x0=[lambda0 omega0 d0 i0];
T=80;              % <- terminal time
tspan=[0 T];       % <- time span (from t=0 up to t=T)

% solver parameters
small=0.000001;    % <- definition of a small number

% approximation
options = odeset('RelTol',small,'AbsTol',small);
```

```
[t,x] = ode45(@(tspan,x) minsky_III_dx(tspan,x,params), tspan, x0,options);

% mapping solutions of ode to economy
lambda=x(:,1);
omega=x(:,2);
d=x(:,3);
i=x(:,4);

f=-(1/tau_p)*(1-omega/(1-s));
r=r_b+i.*(i>0); % <- this syntax is useful
p=1-omega-r.*d;
I=(y_i-m_i)*exp(s_i*((p/nu)-x_i)/(y_i-m_i))+m_i;
g=(I/nu-delta);
% there is a prize for explaining why the line below works!
Y=100*(1+[0; cumsum(0.5*(t(2:end)-t(1:end-1)).*(g(2:end)+g(1:end-1)))]);

n=size(t,1)       % <- number of points in time (optimized by solver)

%%%%%%%%%%%%%% CHARTS %%%%%%%%%%%%%%%%
chart=1; % <- here choose your chart

if chart==1
    plot(t,100*lambda)
    xlabel('TIME (IN YEARS)'); ylabel('EMPLOYMENT (IN %)')
    axis([0 T 0 80]); grid on
elseif chart==2
    plot(t,100*omega)
    xlabel('TIME (IN YEARS)'); ylabel('WAGES TO GDP (IN %)')
    axis([0 T 65 85]); grid on;
elseif chart==3
    plot(t,100*d)
    xlabel('TIME (IN YEARS)'); ylabel('PRIVATE DEBT TO GDP (IN %)'); grid on
elseif chart==4
    plot(t,100*i)
    xlabel('TIME (IN YEARS)'); ylabel('YEARLY INFLATION (IN %)'); grid on
elseif chart==5
    plot(t,100*p)
    xlabel('TIME (IN YEARS)'); ylabel('PROFIT RATE (IN %)'); grid on
elseif chart==6
    plot(t,100*r)
    xlabel('TIME (IN YEARS)'); ylabel('NOMINAL INTEREST (IN %)')
    axis([0 T 0 18]); grid on
elseif chart==7
    plot(t,g)
    xlabel('TIME (IN YEARS)'); ylabel('GDP GROWTH RATE (IN %)'); grid on
elseif chart==8
    plot(t,Y)
    xlabel('TIME (IN YEARS)'); ylabel('GDP (IN USD)')
    axis([0 T 0 400]); grid on
elseif chart==9
    last=902; % <- only first 60 years
    plot(100*lambda(1:last,:),100*i(1:last,:))
    xlabel('EMPLOYMENT (IN %)'); ylabel('INFLATION (IN % PER YEAR)'); grid on
elseif chart==10
   last=902; % <- only first 60 years
   x1=100*(1-lambda(1:last));
   x2=100*i(1:last);
   x3=100*d(1:last);
   c = 1:length(t(1:last));      % <- number of colors
   h = surface([x1(:), x1(:)], [x2(:), x2(:)], [x3(:), x3(:)], ...
   [c(:), c(:)], 'EdgeColor','flat', 'FaceColor','none'); colormap(jet(numel(t)))
   xlabel('UNEMPLOYMENT RATE'); ylabel('INFLATION')
   zlabel('PRIVATE DEBT TO GDP'); grid on
elseif chart==11
    % Investment function
    pt=linspace(-0.05,0.11,1601);
    It=(y_i-m_i)*exp(s_i*(pt-x_i)/(y_i-m_i))+m_i;
    plot(100*nu*pt,100*It); % <- x has to be multiplied by nu
    xlabel('PROFIT (IN %)'); ylabel('INVESTMENT (IN % OF GDP)')
    axis tight; grid on;
elseif chart==12
    % Phillips curve
    lt=linspace(0.8,1.01,2101);
    Wt=(y_w-m_w)*exp(s_w*(lt-x_w)/(y_w-m_w))+m_w;
    plot(100*lt,Wt);
    xlabel('EMPLOYMENT RATE (IN %)'); ylabel('ANNUAL CHANGE IN REAL WAGE (IN %)');
    axis([80 101 -10 1200]); grid on;
end
```

The **minsky_III_dx()** function is defined as:

```
function dx = minsky_III_dx(tspan,x,params)
    alpha=params(1); beta=params(2); delta=params(3); nu=params(4);
    r_b=params(5); s=params(6); tau_p=params(7); tau_i=params(8);
    x_i=params(9); y_i=params(10); s_i=params(11); m_i=params(12);
    x_w=params(13); y_w=params(14); s_w=params(15); m_w=params(16);

    r=r_b; % <- interest rate
    if x(4)>0
        r=r+x(4);
    end
    p=1-x(2)-r*x(3);
    f=-(1/tau_p)*(1-x(2)/(1-s));
    I=(y_i-m_i)*exp(s_i*((p/nu)-x_i)/(y_i-m_i))+m_i;
    W=(y_w-m_w)*exp(s_w*(x(1)-x_w)/(y_w-m_w))+m_w;
    dx=zeros(4,1);
    dx(1)=( ((1/nu)*I-delta) -(alpha + beta) )*x(1);
    dx(2)=( W - (alpha+f) )*x(2);
    dx(3)=( I-p ) -( (1/nu)*I - delta + f )*x(3);
    dx(4)=-(1/tau_i)*(x(4)-f);
end
```

That is all for today. Thank you!

References

Keen, S., 2013a. A monetary Minsky model of the Great Moderation and the Great Recession. Journal of Economic Behavior & Organization 86, 221–235.

Keen, S., 2013b. Predicting the 'Global Financial Crisis': Post-Keynesian Macroeconomics. Economic Record 89, 228–254.