Notes in Computational Economics and Applications

Pedro Brinca

Department of Economics,
Stockholm University
E-mail address: pedro.brinca@ne.su.se
URL: http://people.su.se/pedrobrinca

Abstract. These notes are aimed at advanced undergraduate and graduate students in economics. The goal is to expose the students to elementary concepts of numerical analysis that, nonetheless, will allow the students to model and solve a large class of economic problems. The first part consists of an introduction to the numerical analysis methods and their implementation. The second parts concerns applications of these tools to many problems in economics. Accompanying each technique there is a simple example of an economic problem for which the technique is relevant. The solution and sample code are provided and explained. The software used is MatLab.

Contents

Preface

These notes were written both for my own use and my students. On one hand, they are intended to introduce advanced undergraduate and graduate students to elementary concepts of numerical analysis that, nonetheless, will allow the students to model and solve a large class of economic problems. On the other hand, by disciplining myself in writing these notes, I am making sure that I keep a systematic and organized portfolio of these techniques. Naturally, the topics covered will mainly reflect my own research interests, though I try to give examples of applications of the very same topics to other fields of economics. Very few of the material is original. The sources are many and range from lecture notes, course materials from other professors, books, articles, software documentation and so forth. To the extent of my possibility (and knowledge), I will include references to the material used. If you find any reference missing, please contact me so I can add it. This being said, there are indeed excellent resources already regarding the topics here covered. **?**, **?** and **?** are but a few classic books in the field from where these notes draw heavily. Finally, this is still very preliminary work. These notes will most surely have many typos and imprecisions for which your comments, suggestions and corrections are highly appreciated.

Part 1

Numerical Methods

CHAPTER 1

Numerical differentiation and Integration

This chapter introduces basic techniques to compute numerical derivatives and integrals. The need to for such tools comes from the fact that very often we do not have closed form expressions for the functions we want to compute the derivatives or it is too costly to compute them.

1. Numerical derivatives

The formulas to compute derivatives come from Taylor approximations.

Let f be a continuous and n-differentiable function in the neighborhood of x_0. The first order Taylor approximation to f in the neighborhood of x_0 is given by:

$$f(x) \approx f(x_0) + f'x_0)(x - x_0) \tag{1.1}$$

If we solve w.r.t. $f'(x_0)$ we get:

$$f'(x_0) \approx \frac{f(x) - f(x_0)}{x - x_0} = \frac{f(x_0 + h) - f(x_0)}{h}, h = x - x_0 \tag{1.2}$$

This gives a straightforward formula to compute a numerical derivative. To see how we can implement it, let

$$f(x) = 2 - \frac{1}{2}x^{-0.5} - \frac{1}{2}x^{-0.2} \tag{1.3}$$

The analytical derivative of f is given by

$$f'(x) = \frac{1}{4}x^{-\frac{3}{2}} + \frac{1}{10}x^{-\frac{6}{5}} \tag{1.4}$$

Let $x_0 = 2$. Then $f'(2) = 0.131915875813125$. Let us use the formula in 1.2 to compute the numerical approximation:

```
h = 10^-10;
f = @(x) 2-0.5*x^-0.5-0.5*x^-0.2; %defines the f function
df = @(x) (f(x+h)-f(x))/h; %forward difference approximation
df(2)

ans =

   0.131916699785961
```

In this example, the formula derived from the Taylor approximation is correct up to the 5^{th} decimal. It may be so that we need higher precision. One way of doing it is to compute the optimal size of h. For a discussion on this, see **?**. An alternative is to use a better approximation. Consider two second order Taylor

approximations to f around $x+h$ and $x-h$, subtract the second from the first and get the below approximation to the derivative:

$$f'(x_0) \approx \frac{f(x+h) - f(x-h)}{2h} \tag{1.5}$$

Approximations for higher order derivatives can also be found from higher order Taylor approximations. For the second derivative we have

$$f''(x_0) \approx \frac{f(x_0+h) - 2f(x_0) + f(x_0-h)}{h^2} \tag{1.6}$$

which can also be easily implemented.

EXERCISE 1. *Let f be a continuous and twice differentiable function of x in the interval $[0.05, 2]$, such that*

$$f(x) = 2 - 0.5x^{-0.5} - 0.5x^{-0.2} \tag{1.7}$$

Compute the analytical derivative and numerical derivatives with both forward and two-sided approximation. Evaluate the three functions in the $[0.05, 2]$ interval and plot the absolute values of the difference between the analytical derivative and each numerical approximation. What do you conclude?

Solution:

```
%analytical f
syms x
f  = 2-0.5*x^-0.5-0.5*x^-0.2;

%analytical derivative
df = diff(f,1);

%numerical f
fn = @(x) 2-0.5*x.^-0.5-0.5*x.^-0.2;

%numerical derivative using forward difference approximation
h = 10^-10; %
df_fa = @(x) (fn(x+h)-fn(x))/h;

%numerical derivative using two-sided difference approximation
df_2s = @(x) (fn(x+h)-fn(x-h))/(2*h);

%define the domain
x  = 0.5:0.01:2;

%evaluate the three functions for x
ay = eval(subs(df,x)); fy = df_fa(x); ty = df_2s(x);

%plot the absolute value of the differences
plot(x,abs(ay-fy),x,abs(ay-ty));
```

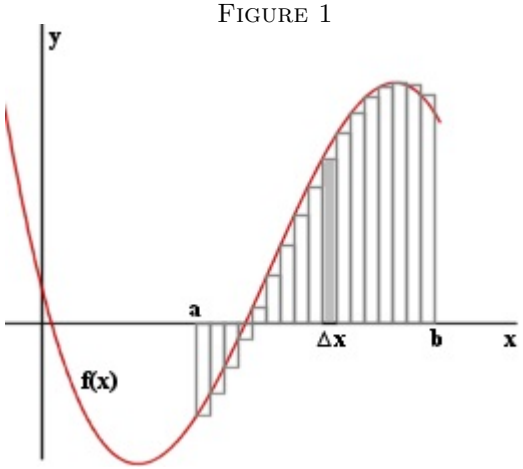
2. Numerical Integrals

The need to use numerical methods to compute integrals arises from the same issues we mentioned for derivatives. The methods mentioned below rely in discretizing the domain and approximate the function with piece-wise polynomials. More generally, all methods described here make use of the below approximation:

$$\int_I f(x)w(x)dx \approx \sum_{i=0}^{n} w_i f(x_i) \tag{2.1}$$

Below an example where the function f is approximated in the interval $[a, b]$ by a stepwise function, for which computing the integral becomes a trivial problem i.e. just add up the area of the rectangles.

FIGURE 1



The method exemplified graphically in 1 is called the left Riemann sum, because the height of the rectangle that approximates the integral of the function in each subinterval is given by the value of the function at the beggining of the interval. One could alternatively compute the right Riemann sum, or even both and take the average as the approximation. Let's look at the below exercise.

EXERCISE 2. *Let $f(x) = x^2$. Compute $\int_1^4 x^2 \, \mathrm{d}x$ by making use of the left and right Riemann sums and the average of both. Compare the numerical approximations with the analytical solution for $n = 50, 500, 5000$.*

The analytical solution is given by:

$$\int_1^4 x^2 \, \mathrm{d}x = \left[\frac{x^3}{3}\right]_1^4 = \frac{64}{3} - \frac{1}{3} = 21 \tag{2.2}$$

In order to compute the numerical approximation to $\int_1^4 x^2 \, \mathrm{d}x$, we can use the code below:

```
for n = [50 500 5000]
% analytical f, the integral of f and numerical f and integration interval
```

```
syms x; f = x.^2; intf = int(f); fn = @(x) x.^2; a = 1; b = 4;

% get the lenght of the base of each rectangle
dx = (b-a)/n;

% create a vector of all nodes evaluate them and compute the integral
lnodes = a:dx:4-dx; lfvalues = fn(lnodes);lnsol =sum(dx.*lfvalues);
rnodes = a+dx:dx:4; rfvalues = fn(rnodes);rnsol =sum(dx.*rfvalues);

% compare with analytical solution
asol = subs(intf,4)-subs(intf,1); [eval(asol) lnsol rnsol (lnsol+rnsol)/2]
end

ans =

   21.0000   20.5518   21.4518   21.0018
   21.0000   20.9550   21.0450   21.0000
   21.0000   20.9955   21.0045   21.0000
```

As we can see, in this case, the approximation increases in accuracy as n increases. We can also observe that the average between the right and left Riemann sums converges much faster to the analytical solution. This last method is said to be first order exact, because it will give the exact solution when computing the integral of any first order polynomial.

Sometimes, we need higher order accuracy. The Simpson's rule, instead of approximating the f with a pice-wise linear polynomial as in the previous cases, approximates f with a second-order pice-wise polynomial. Gaussian quadrature methods, in particular Gauss-Legendre quadrature, provide even higher accuracy.

2.1. Monte-Carlo integration. Suppose you follow the below exercise:

EXERCISE 3. *The manager of a supermarket has to decide each day how much bread to buy. For every bread that it is sold, there is a profit of 60 cents. However, for every one not sold, there is a loss of 40 cents. The demand is uniformily distributed in the interval* $[80, 140]$. *How much bread should the manager order, in order to maximize the profit? What if the demand follows the Gamma distribution, with parameters* $k = 2$ *and* $\theta = 55$*?*

We start by formalizing the problem. Let P denote the profit. Then:

$$P = \begin{cases} 0.6Q & \text{if } D \geq Q \\ 0.6D - 0.4\,(Q - D) & \text{if } D < Q \end{cases} \tag{2.3}$$

Let's first address the case when $D \sim U(80, 140)$. Solving this problem analytically is straitforward. This is because D's probability density function is just given by:

$$\begin{cases} \frac{1}{b-a}, & \text{if } x \in [a, b] \\ 0, & \text{elsewhere} \end{cases} \tag{2.4}$$

Hence, the problem reduces to finding the quantity Q that maximizes the expected profit:

$$\max_Q \mathrm{E}[P] = \max_Q \int_{80}^{Q} [0.6x - 0.4(Q-x)]\frac{1}{60}\mathrm{d}x + \int_Q^{140} 0.6Q\frac{1}{60}\mathrm{d}x \tag{2.5}$$

In this case, the problem has a closed form solution. Just integrate w.r.t. to x and solve the quadratic maximization problem in Q. How about in the case when the demand follows the Gamma distribution? The expression for the expected profit given quantity Q is now:

$$\mathrm{E}[P(Q)] = \int_0^{Q} [0.6x - 0.4(Q-x)]\frac{1}{\Gamma(k)\theta^k}x^{k-1}\mathrm{d}x + \int_Q^{+\infty} 0.6Q\frac{1}{\Gamma(k)\theta^k}x^{k-1}\mathrm{d}x \tag{2.6}$$

We now have two issues. First, we do not have a closed form solution to the Γ function, that we need in order to evaluate the probability density function. So we can rule out solving this analytically. Most importantly even, we cannot use our integration techniques from before, because the outcome of the integration process depends on a parameter Q. One way would be to evaluate the integral for each Q over a grid and pick the maximum, but since Γ has strictly positive density over $\mathbb{R}^+$ that would imply a grid with an infinite number of nodes. This is a case where Monte Carlo integration can help us.

The idea behind Monte-Carlo integration is to choose the nodes randomly in the interval of integration. At face value this may seem a bit strange. And in fact, in many occasions, choosing regularly spaced points will yield a better approximation. The convenience of Monte-Carlo integration lies elsewhere though. In economics we are often faced with the problem of computing expected values, which is an integration problem.

Let's take a step back. Suppose that we want to compute the below integral:

$$\omega = \int_0^1 f(x)\mathrm{d}x \tag{2.7}$$

If it is too cumbersome, or even impossible to compute it analytically, note that:

$$\omega = \mathrm{E}[f(X)],\ X \sim U(0,1) \tag{2.8}$$

This means that we can compute the integral by generating random realizations from the uniform distribution over the interval $[0,1]$, evaluate each realization by the f function and compute the average. This is exactly what we did before for the left and right Riemann sums, but now instead of defining equally spaced nodes, we draw the nodes randomly from the uniform distribution over the integration interval. From the Strong Law of the Large numbers, we know that the estimator of ω:

$$\hat{\omega}_n = \frac{1}{n}\sum_1^n f(X_i) \tag{2.9}$$

is unbiased, i.e., $\mathrm{E}[\hat{\omega}_n] = \omega$ and is consistent, i.e., $\hat{\theta}_n \xrightarrow{p} \omega$. The example below approximates $\int_0^1 x^2\mathrm{d}x$ which we know to be $\frac{1}{3}$:

```
% generate n realizations of x~U(0,1)
n=500; x = rand(n,1);

%evaluate the function at all x's and compute the mean
```

```
f = x.^2; omega = mean(f)

omega =

   0.312664416159528
```

There is a subtlety though. This simple example works because:

$$\mathrm{E}\left[f(X)\right] = \int_a^b f(x)g(x)\mathrm{d}x = \int_0^1 f(x)\mathrm{d}x =,\ g(x) = \frac{1}{b-a} = 1 \tag{2.10}$$

precisely for the case of $a = 0$ and $b = 1$, because $X \sim U(0,1)$ and $g(x)$ is the p.d.f. of X.

Let's go back to our exercise 2. Note that

$$\int_1^4 x^2\mathrm{d}x = (4-1)\int_1^4 x^2\frac{1}{4-1}\mathrm{d}x \tag{2.11}$$

The integral in the left hand side of 2.11 has now the form of an expected value, as in expression 2.10. We can now easily compute our approximation to the integral using the same Monte Carlo procedure from above.

```
% generate n realizations of x~U(0,1)
omegas = zeros(1,3)*NaN;i=1;
for n = [50 500 5000]
%draw from U(1,3)
x = 1+(4-1)*rand(n,1);

%evaluate the function at all x's and compute the mean
f = x.^2; omegas(i) = 3*mean(f);i=i+1;
end
omegas

omegas =

   22.8723   21.1865   20.9019
```

We are now able to go back and solve the problem in equation 2.6.We just need to draw from the Gamma distribution instead. Just before we solve it, notice that we have two improper integrals i.e. the integrals are not bounded from above. In this case, the integrals are only properly defined if

$$\lim_{b\to+\infty}\int_0^b f(x)\mathrm{d}x = c, c \in \mathbb{R} \tag{2.12}$$

Or, in other words, if the sum converges. However, we know that Γ is a probability density function and therefore the integral over any subset of the support converges to a constant equal or lesser than one. This means that for sufficiently high Q, the density is so low that we can disregard it in our approximation. Therefore, implementing the solution to the problem above becomes straightforward.

```
% Define the parameters of the gamma distribution
k = 2; theta=55;
for n = [5 500 5000];

    % compute inv gamma of U(0,1) to spread nodes over all the support
    Q  = gaminv(rand(n,1),k,theta);

    gQ = gampdf(Q,k,theta);% compute the density g(Q) at each Q
    P  = zeros(n,1)*NaN; % stores profits(Q(i),D)
    D  = (random('gamma',k,theta,n,1)); % compute realization of n demands
    mP = zeros(size(Q,1),1)*NaN; % stores the expected profit (Q(i))

    %compute the expected profit for each pair of Q and D
    for i=1:size(Q,1);
        for j=1:n
            if D(j) >= Q(i)
                P(j) = 0.6*Q(i);
            else
                P(j) = 0.6*D(j)-0.4*(Q(i)-D(j));
            end
        end
        mP(i) = mean(P);
    end

    Pfunc = [Q mP]; %create "Profit funtion"
    [maxx iq] = max(Pfunc(:,2),[],1); %gets max of mP and respective Q


disp(['Optimal Q = ' num2str(Pfunc(iq,1)) ' and Expected profit = '...
    num2str(maxx) ' for n = ' num2str(n) ' draws.'])
end
```

```
Optimal Q = 43.417 and Expected profit = 17.0865 for n = 5 draws.
Optimal Q = 112.2306 and Expected profit = 38.1249 for n = 500 draws.
Optimal Q = 111.5217 and Expected profit = 36.7342 for n = 5000 draws.
```

CHAPTER 2

Curve fitting and function approximation

Very often we need to work with approximations to the actual functions we are interested in. One reason may be that we do not know the actual function but have access to a limited sample of data from which we try to infer the functional relationship. That is the idea behing interpolation and projection methods. Other times we might have full knowledge about the functional form, but for tractability purposes, we need to use a linear or low order approximation. This is generally the case for Taylor approximations.

1. Interpolation

Assume the supply of a good is constant and that the demand for the same good is equal to 5 if the price is 4 and 10 if the price is equal to 2. We have our prior that demand is strictly decreasing as a function of the price and we would like to know much would the demand be if the price were set to 3.

Notice that our prior is not enough to answer the question. There is an infinite set of strictly decreasing functions for which $D(4) = 5, D(10) = 2$ and $D'(p) < 0$. Without further information, we need to assume some functional form for the demand function in this interval.

The easiest way is to assume the demand is linear. If it is the case, then it follows that $D(3) = 7.5$. To see this just notice that under the assumption of linearity, if to a price increase of 2, corresponds a demand decrease of 5, then to a price increase of 1.5, corresponds a decrease demanded quantity of 2.5.

However, as said before, this is not the only possibility. A higher order interpolation would result in a demand of 5.5 for the same price. Only economic theory can tell which interpolation procedure is more appropriate.

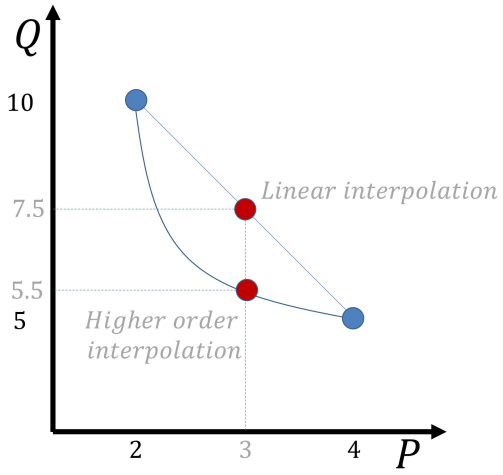
This is shown in Figure 1 below.

A separate issue concerns extrapolation. In the example in 1, we want to find the image of a point in the convex set of X. We could instead want to find the demand when the price equals 5. Again, we need further assumptions in order to provide our guess. In the linear extrapolation example, we can for instance assume that the function is also linear outside X's convex set. In this case the demanded quantity for a price of 5 is equal to 2.5.

Naturally, interpolation methods are generalized for n data points. In particular we will look into methods where a picewise polynomial is fitted to the interval between two consecutive data points. Let's look more closely at both picewise linear interpolation and a particular case of a higher order curve fitting procedure.

1.1. Linear interpolation. In picewise linear interpolation, we require that the value of each polynomial coincides with each adjacent data point value i.e. two conditions for each linear picewise polynomial. Since each line needs two parameter

FIGURE 1. Linear and higher order interpolation



values per interval (intercept and slope) the interpolation problem is just to solve a system of n linear equations that is exactly identified. The number of equations is equal to the number of intervals i.e. the number of data points minus one. More formally:

DEFINITION 1. *Given n data points* $(x_1, y_1), ..., (x_n, y_n)$*, a linear piecewise polynomial is a function defined of the form:*

$$\begin{aligned} P_1(x) &= a_1 + b_1 x, x \in [x_1, x_2] \\ P_2(x) &= a_2 + b_2 x, x \in [x_2, x_3] \\ \vdots &= \vdots \\ P_{n-1}(x) &= a_{n-1} + b_{n-1} x, x \in [x_{n-1}, x_n] \end{aligned}$$

and we require that:

$$\begin{aligned} P_i(x_i) &= y_i \\ P_i(x_{i+1}) &= y_{i+1} \end{aligned}$$

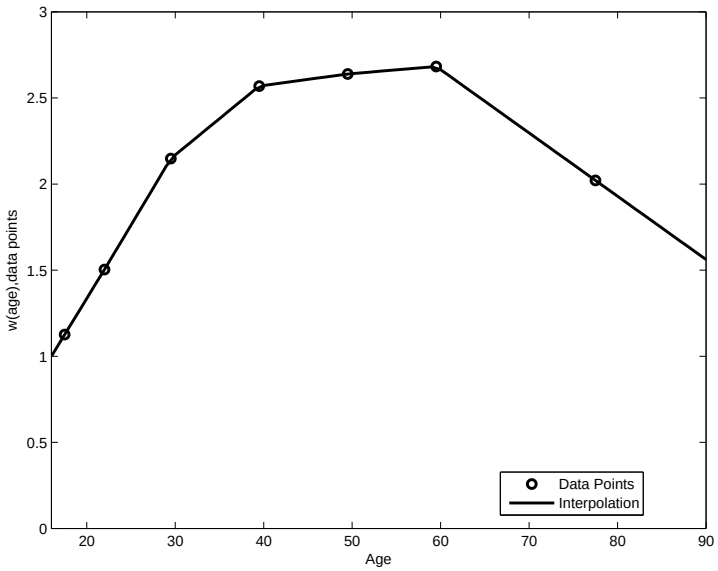
for $i \in 1, 2, ...n$.

To better exemplify the interpolation and extrapolation procedures, consider the following exercise.

EXERCISE 4. *Access Table 1 from http://www.bls.gov/cps/cpswom2007.pdf where you can find weekly median wages by age group. Plot the data points and compute the linear interpolation and extrapolation for ages 16 to 90 years old. For extrapolation, use the linear interpolation of adjacent interval. Normalize the wages to the first interpolated value.*

The Figure 2 below shows the outcome of the linear interpolation procedure:

FIGURE 2. Linear interpolated CPS wages as a function of age



The code makes use of the `interp1.m`function. It takes as inputs two vectors (X and Y) that consist precisely of the ordered pairs referred above. The third argument concerns the domain over which the function is supposed to produce the interpolation / extrapolation. The fourth argument defines the interpolation procedure (linear interpolation in the below example). The last command informs the function to produce extrapolations if the domain contains elements outside the convex set of the elements in X.

```
%Table 1 from http://www.bls.gov/cps/cpswom2007.pdf

bracket_16t19  = 337*52; % multiply by 52 to get yearly wages
bracket_20t24  = 450*52;
bracket_25t34  = 643*52;
bracket_35t44  = 769*52;
bracket_45t54  = 790*52;
bracket_55t64  = 803*52;
bracket_65plus = 605*52;

wageprofile    = [bracket_16t19;bracket_20t24;bracket_25t34;bracket_35t44;...
    bracket_45t54;bracket_55t64;bracket_65plus];

% compute the middle point for each age bracket
agebrackets    = [16+19;20+24;25+34;35+44;45+54;55+64;65+90]./2;

xi = agebrackets;
yi = wageprofile;
```

```
% creates the domain of points for which we want to interpolate/extrapolate
xx = 16:1:90;

% produces the interpolations/extrapolations
yy = interp1(xi,yi,xx,'linear','extrap');

figure
plot(xi,yi/yy(1),'ko',xx,yy/yy(1),'k','LineWidth',2)
xlabel('Age')
ylabel('w(age),data points')
legend('Data Points','Interpolation','Location','Best')
axis([16 90 0 3])
```

1.2. Cubic splines interpolation. As mentioned before, we can perform higher order approximations. In particular we may want to use an interpolation method that produces a function that is twice continuous differentiable, given that in most of our models both first and second derivatives matter. One of the methods that is standard and produces an interpolation function that is twice continuous differentiable is to fit a third order polynomial to each interval. This method is called interpolation by cubic splines.

Each third order polynomial is defined by four parameteres. Two conditions are similar to the ones used in fitting linear picewise polynomials, or to any interpolation method for that matter. Each polynomial must have the same value at adjacent data points. The two extra conditions come from requiring that both the first and the second derivatives are also the same at adjacent data points. More formally:

DEFINITION 2. *Given n data points $(x_1, y_1), ..., (x_n, y_n)$, a cubic spline is a piecewise defined function of the form:*

$$\begin{aligned}
S_1(x) &= y_1 + b_1(x - x_1) + c_1(x - x_1)^2 + d_1(x - x_1)^3, x \in [x_1, x_2] \\
S_2(x) &= y_2 + b_2(x - x_2) + c_2(x - x_2)^2 + d_2(x - x_2)^3, x \in [x_2, x_3] \\
\vdots &\quad = \quad \vdots \\
S_{n-1}(x) &= y_{n-1} + b_{n-1}(x - x_{n-1}) + c_{n-1}(x - x_{n-1})^2 + d_{n-1}(x - x_{n-1})^3, x \in [x_{n-1}, x_n]
\end{aligned}$$

Notice that we have $3n - 3$ unknowns. From the first equation, notice that we already have that $S_1(x_1) = y_1$. We add $S_{n-1}(x_n) = y_n$. This ensures that the function intersects all data points. We further require the function to be continous i.e.:

$$S_i(x_{i+1}) = S_{i+1}(x_{i+1}) = y_{i+1} \tag{1.1}$$

So far we have $n - 1$ constraints. We then require that the function is first order differentiable at each interior point:

$$S_i^{'}(x_{i+1}) = S_{i+1}^{'}(x_{i+1}) \tag{1.2}$$

and also second order differentiable at the same interior points:

$$S_i^{''}(x_{i+1}) = S_{i+1}^{''}(x_{i+1}) \tag{1.3}$$

which adds another $2n-4$ equations. In total so far we have $3n-5$ constraints, but $3n-3$ coefficients to compute. We need two extra constraints so that the system of equations is uniquely identified.

There are three common assumptions that are used to solve the problem:

- The natural spline: $S_1^{''}(x_1) = 0 = S_{n-1}^{''}(x_n)$
- The clamped spline: $S_1^{'} = a$ and $S_{n-1}^{'}(x_n) = b$, for arbitrary $a, b \in \mathbb{R}$
- Not-a-knot: $S_1^{'''}(x_2) = S_2^{'''}(x_2)$ and $S_{n-2}^{'''}(x_{n-2}) = S_{n-1}^{'''}(x_{n-2})$

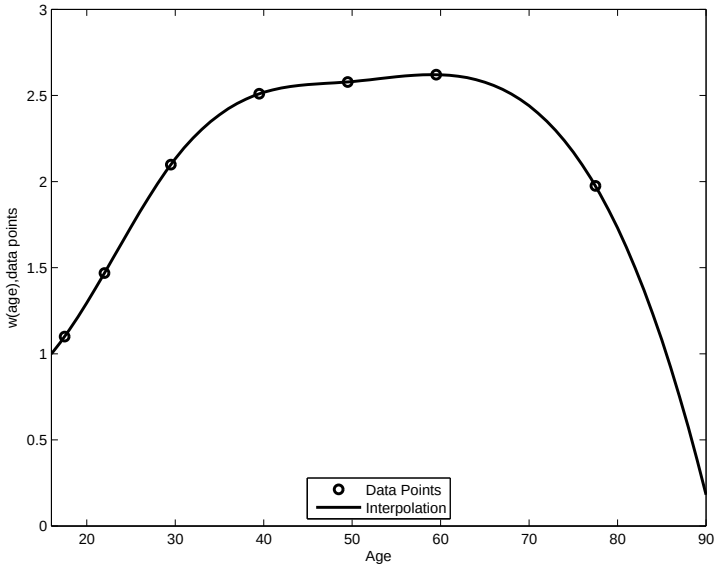
The last assumption is the one used by default by MatLab in the spline.m function (called by `interp1.m`when given the option `'spline'`).

To solve Exercise 4 now with cubic splines, just use the interp1.m function with the 'spline' command instead:

```
yy = interp1(xi,yi,xx,'spline','extrap');
```

and the cubic splines function now come as in Figure 3:

FIGURE 3. Cubic splines interpolated CPS wages as a function of age



2. Taylor approximation

There is often the need to approximate non-linear functions with lower order polynomials. These approximations can be very usefull, particularly if we are interested in studying the behavior of a function in a small interval. The Taylor theorem below shows how a n-differentiable function can be approximated to arbitrary precision by a n^{th} order Taylor approximation:

THEOREM 1. *Let $n \in \mathbb{N}$ and $f : \mathbb{R} \to \mathbb{R}$ be n times differentiable at $x_0 \in \mathbb{R}$. Then there exists a function $h_n : \mathbb{R} \to \mathbb{R}$:*

$$f(x) = \frac{f(x_0)}{0!}(x-x_0)^0 + \frac{f'(x_0)}{1!}(x-x_0)^1 + \frac{f''(x_0)}{2!}(x-x_0)^2 + \ldots + \frac{f^n(x_0)}{n!}(x-x_0)^n + h_n(x)(x-x_0)^n$$

and

$$\lim_{x \to x_0} h_n(x) = 0 \tag{2.1}$$

In practice we ignore the $h_n(x)(x - x_0)$ and say that the Taylor polynomial approximates f in the neighbourhood of x_0. Let's look at the below exercise to visualize some Taylor approximations:

EXERCISE 5. *Let the demand for a good be defined as* $Q(p) = 0.5p^{-0.5} + 0.3p^{-0.2}$. *Plot the original demand curve and Taylor approximations of degree one to three around* $x_0 = 0.5$. *Make four plots with distances of* $0.1, 0.2, 0.5$ *and* 1.5 *around* x_0.

In the first panel of Figure 4 we can see that in the neighbourhood of $x_0 = 0.5$, all the approximations are reasonably close (off course the definition of reasonable is based on economic relevance and is contingent on the problem being solved. When we move to the second panel and we inspect the behavior of the approximations in a wider neighbourhood, we notice how high order polynomials manage to significantly outperform a simple linear approximation. This is even more obvious in the third panel.

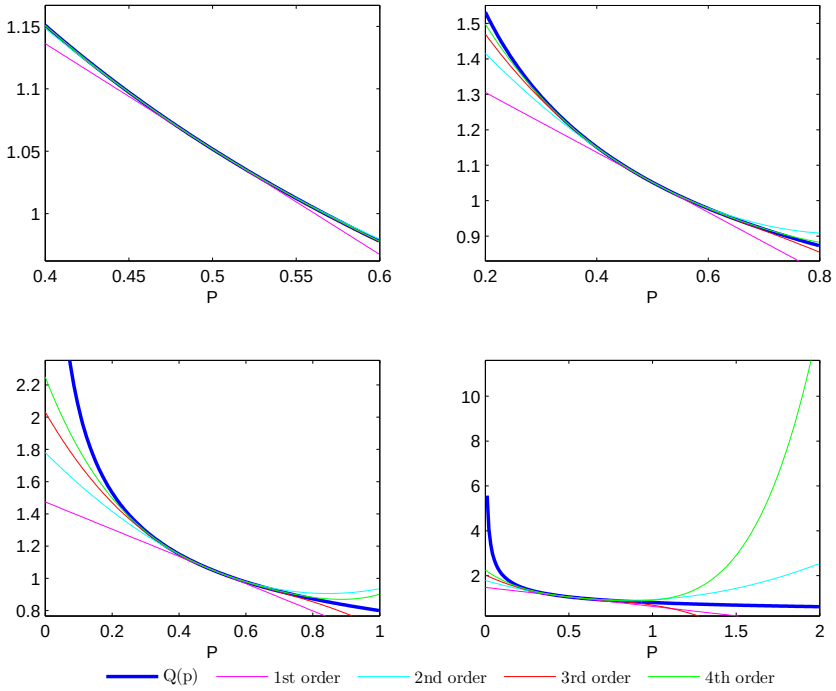
However, as we go further away from x_0, notice that higher order polynomials will diverge away from the true function at a much faster rate than lower order approximations.

The lesson here is that low order Taylor approximations can be usefull and more precision can be achieved in the vicinity of the approximation point by increasing the order of the polynomial, but one should be carefull about interpreting results implied by approximations when far away from the approximation point.

We used the `taylor.m`function that is part of the Symbolic toolbox in MatLab to perform the approximations and plot the results. The code below produces the plots shown in Figure 4. Notice that the objects being plotted are symbolic functions for which we use the function `ezplot.m`.

```
syms P % declare P as symbolic
Q = 0.5*P^-0.5+0.3*P^-0.2; % original function
Q1 = taylor(Q,P,0.5,'order',2); % The taylor approximations
Q2 = taylor(Q,P,0.5,'order',3); % Notice that the order option is defined
Q3 = taylor(Q,P,0.5,'order',4); % as 1 + the approximation order. This is
Q4 = taylor(Q,P,0.5,'order',5); % actually the order of the remainder.

xint = [0.4,0.6;0.2,0.8;0,1;0,2]; %different intervals for the plots

for i=1:4
subplot(2,2,i)
pQ = ezplot(Q,xint(i,:));
set(pQ,'Color','blue','LineWidth', 2)
hold on;
pQ1 = ezplot(Q1,xint(i,:));
set(pQ1,'Color','magenta')
hold on;
pQ2 = ezplot(Q2,xint(i,:));
set(pQ2,'Color','cyan')
```

FIGURE 4. Taylor approximations to $Q(p)$ around 0.5




```
hold on;
pQ3 = ezplot(Q3,xint(i,:));
set(pQ3,'Color','red')
hold on;
pQ4 = ezplot(Q4,xint(i,:));
set(pQ4,'Color','green')
hold off;
title('');
end

h1 = legend('Q(p)','1st order','2nd order',...
'3rd order','4th order','Location', 'NorthOutside','Orientation',...
'horizontal'); legend('boxoff');

p = get(h1,'Position');p(1)=0.08;p(2)=-0.00;set(h1,'Position',p);
 set(h1,'Interpreter','latex'); %manually center the legend
```

CHAPTER 3

Root finding algorithms

This chapter introduces classic techniques to solve equations and systems of equations. I start by showing how a very simple economic question can be posed for which there is the need to use numerical methods.

EXAMPLE 1. *Suppose the demand for a good is given by* $Q(p) = 0.5p^{-0.5} + 0.3p^{-0.2}$. *For a given demanded quantity of* $Q = 1$, *what is the price* p *that clears the market?* [1]

The solution to Example 1 consists in solving the equation below:

$$1 - 0.5p^{-0.5} - 0.3p^{-0.2} = 0 \tag{0.1}$$

There is no closed form solution to the above problem i.e. we cannot substitute Q for 1, solve the demand function w.r.t. p and do the math. Note however, that there exists a solution to this problem. The code below plots the demand function from Example 1 and equation 0.1, also called excess demand function.

```
figure
subplot(2,1,1)
ezplot(@(p) 0.5*p^-0.5+0.3*p^-0.2);
subplot(2,1,2)
ezplot(@(p) 1-0.5*p^-0.5-0.3*p^-0.2);
```

As it can be seen, the demand equation (first plot in Figure 1) looks like a standard demand function i.e. it is downward sloping. For $Q = 1$, we can see that p should lie somewhere around 0.5. Let's substitute $p = 0.5$ in the excess demand function:

```
>> subs(@(p) 1-0.5*p^-0.5-0.3*p^-0.2,0.5)

ans =

   -0.0517
```

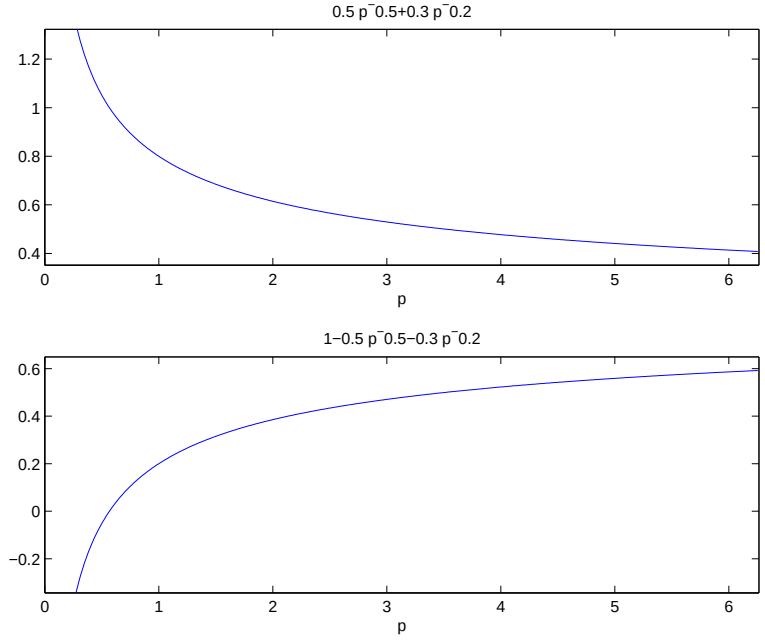
The price $p = 0.5$ does not clear the market at $Q = 1$, because as we can see, the actual demanded quantity is

```
>> subs(@(p) 0.5*p^-0.5+0.3*p^-0.2,0.5)

ans =
```

[1] ?

```
    1.0517
```

FIGURE 1



What if we raise the price, in order to reduce the demanded quantity? Let $p = 0.6$. Then:

```
>> subs(@(p) 1-0.5*p^-0.5-0.3*p^-0.2,0.6)

ans =

    0.0222
```

The price is now too high, as for $p = 6$ the demand is just:

```
>> subs(@(p) 0.5*p^-0.5+0.3*p^-0.2,0.6)

ans =

    0.9778
```

Given the continuity and monotonicity of Equation 0.1, the unique solution lies in the interval $[0.5, 0.6]$. We could continue by trying to further narrow this interval until it is arbitrarily small. This procedure to find the solution of an equation is called the bisection method.

1. Bisection

To ensure the existence of a solution to the problem, the following theorem is stated:

THEOREM 2. *Let f be a real valued function, defined and continuous on a bounded closed interval $[a, b]$ of the real line. Assume further that $f(a)f(b) < 0$. Then, there exists at least a ρ in $[a, b]$, such that $f(\rho) = 0$*

Notice how the theorem doesn't say anything about uniqueness. A sufficient condition for the solution to be unique is to further assume that the function is strictly monotonous in the interval $[a, b]$:

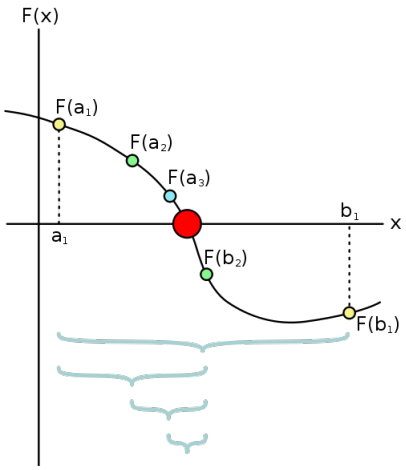
THEOREM 3. *Let f be a real valued function, defined, continuous and strictly monotone on a bounded closed interval $[a, b]$ of the real line. Assume further that $f(a)f(b) < 0$. Then, there exists a unique ρ in $[a, b]$, such that $f(\rho) = 0$*

The bisection algorithm works as follows:

(1) Start with the interval $[a_1, b_1]$
(2) Compute $F(\frac{a_1+b_1}{2})$
(3) If the sign of $F(\frac{a_1+b_1}{2})$ is the same as of $F(a_1)$, the root must be to the right of $F(\frac{a_1+b_1}{2})$ and now we set $a_2 = \frac{a_1+b_1}{2}$.
(4) If instead, the sign of $F(\frac{a_1+b_1}{2})$ is the same as of $F(b_1)$, the root must be to the right of $F(\frac{a_1+b_1}{2})$ and now we set $b_2 = \frac{a_1+b_1}{2}$.
(5) Go back to step 2 and repeat to arbitrary precision.

Below a graphical description[2] of the algorithm:

FIGURE 2



In order to implement this algorithm, we need to define what we mean by arbitrary precision. Basically we need to tell the computer when to stop. Many

[2]picture available in: `http://en.wikipedia.org/wiki/File:Bisection_method.svg`

criteria can be used. We could tell the computer to stop iterating once $[a_j, b_k]$[3] is bounded by an interval of length equal or smaller than δ. This ensures that $\frac{a_j+b_k}{2}$ is at most within δ distance from the solution. If the function is very steep in the neighborhood of the root, this criteria may yield a solution such that $|F(\frac{a_j+b_k}{2})|$ may still be significantly larger than zero. An alternative is to stop iterating once $|F(\frac{a_j+b_k}{2})|$ is smaller than δ. Or just stop once both criteria are met.

Below the code for a function (`bisect.m`) that implements the bisection algorithm:

```
function y = bisect(f,a,b,tol)

% this functions uses the bisection algorithm to find a root of f in the
% [a,b] interval up to tol precision. it outputs the value of x such that
% |x-x0|<tol, where x0 is the solution.

while b-a > tol
   if sign(f((a+b)/2)) == sign(f(a));a=(a+b)/2;
   elseif sign(f((a+b)/2)) == sign(f(b));b=(a+b)/2;end
end

y=(a+b)/2;
```

Let's go back to our initial problem. We want to find p such that $1-0.5p^{-0.5}-0.3p^{-0.2}=0$ where $p \in [0.5, 0.6]$, say to 10^{-15} precision. By making use of the `bisect.m` function, we find the solution.

```
>> x0 = bisect(@(p) 1-0.5*p^-0.5-0.3*p^-0.2,0.5,0.6,10^-15)

x0 =

    0.5671
```

which is shown to be root of the excess demand equation:

```
>> subs(@(p) 1-0.5*p^-0.5-0.3*p^-0.2,x0)

ans =

   1.1102e-16
```

and the price for which the quantity demanded of the good is 1:

```
>> subs(@(p) 0.5*p^-0.5+0.3*p^-0.2,x0)

ans =

    1.0000
```

[3]where j and k are the number of times the bounds a and b were updated,

MatLab has its own function that uses the bisection algorithm, namely `fzero.m` . Though under the conditions stated in 3 convergence is assured, there is no trivial way to generalize this method to higher dimensions and we have no assurance of convergence in such circumstances.

2. Fixed Point Function Iteration

The idea behind fixed point iteration as an algorithm to find the solution to $f(x) = 0$ is to rewrite it as a fixed point problem:

$$f(x) = 0 \Longleftrightarrow x = x - f(x) \Longleftrightarrow x = g(x),\ g(x) = x - f(x) \tag{2.1}$$

The algorithm involves starting with a guess x_0 and then iterating following the below rule:

$$x^{i+1} = g(x^i) \tag{2.2}$$

The code below creates the function `fixedproot.m` that implements the algorithm.

```
function root = fixproot(f,x0,tol,maxiter)

% Takes as inputs a function, a guess for the root x0 and searches for
% the root of f with tol precision and at most in maxiter iterations.

gn = @(x) x-f(x);
iter = 0; d=1;
while abs(d) > tol && iter < maxiter
    x = gn(x0); d=abs(x-x0);
    x0=x;
    iter = iter+1;
end
if iter == maxiter
    disp('maximum number of iterations reached, convergenge not assured')
end
root = x;
```

Applying it to solve $1 - 0.5p^{-0.5} - 0.3p^{-0.2} = 0$ and again we get:

```
>> fixproot(@(p) 1-0.5*p^-0.5-0.3*p^-0.2,5,10^-10,1000)

ans =

    0.5671
```

The trick is to use the transformation in 3 to transform the root finding problem into a fixed point problem. Next show graphically how the fixed point iteration works.

EXERCISE 6. *Let $f(x) = \cos(x)$. Find x such that $f(x) = x$ up to 10^{-10} precision.*

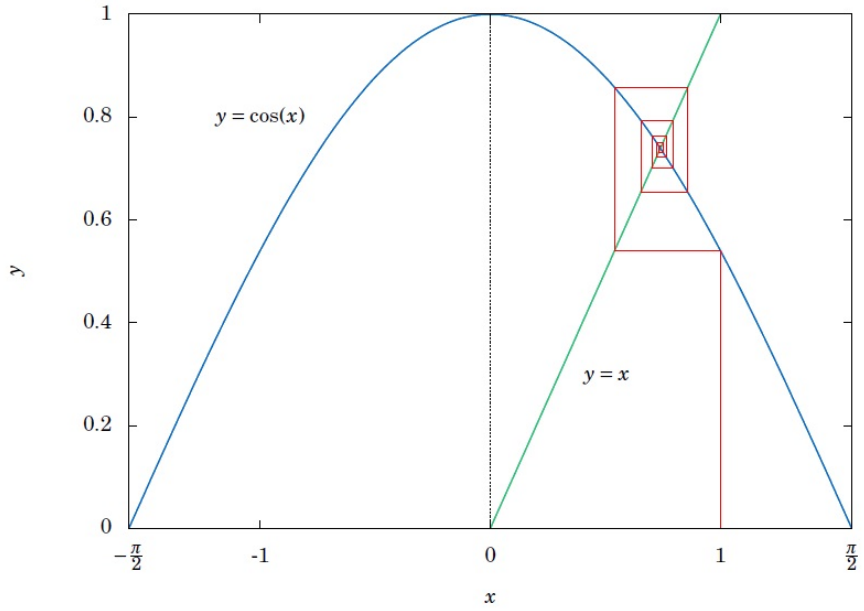
Let $x_0 = 1$. A simple code to find x can also be written:

```
x0=1;
for i=1:100
    x0 = cos(x0);
```

```
end
x0
x0 =

    0.7391
```

Graphically, we reach the solution as in Figure

FIGURE 3



The properties of the fixed point iteration algorithm are stated in the below theorem:

THEOREM 4. *Let $f \in C^1[a,b] : f(x) \in [a,b], \forall x \in [a,b]$. Furthermore, suppose $|f'(x)| \in (0,1), \forall x \in [a,b]$. Then, for any $x_0 \in [a,b]$, the sequence $x^{k+1} = f(x^k)$ converges to the unique fixed point in $[a,b]$*

Basically the theorem claims that any function f that is continuous and differentiable in the interval $[a,b]$ and whose derivative is always smaller than one in the same interval, will have a unique fixed point that can be reached through the fixed point iteration algorithm.

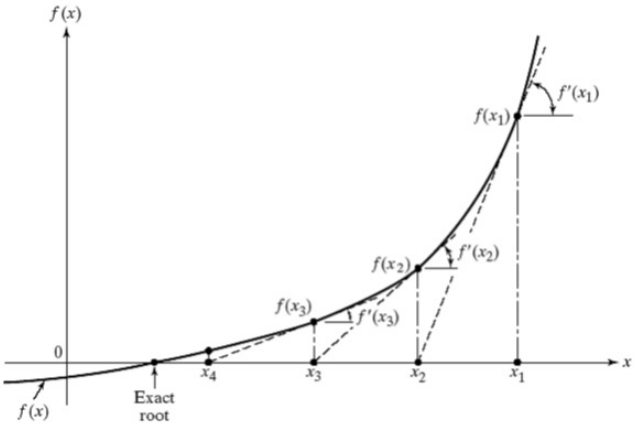
This method is can easily be generalized for higher dimensional problems, and the same principles apply. Though the requirement of the derivative to be lesser than one may seem restrictive, these are sufficient conditions and very often the algorithm converges even when the conditions are not satisfied.

These results are of paramount importance for economics and are the basis for many of the numerical solution approaches in dynamic programming.

3. Newton-Rhapson

The Newton-Rhapson method to find the root of an equation (or a system of equations) is one of the most used in computational economics. The best way to describe the algorithm is to analyze Figure 4:

FIGURE 4



We start with a guess x_0. We compute the equation of the line that is tangent to f at x_0 and find the point where it intersects the x-axis. That will be our next point in the iteration. Deriving the general rule for the iteration is straighforward. The general equation of a line is given by:

$$y = a + bx \tag{3.1}$$

Note that $b = f'(x_1)$. Substituting in 3.2 and solving w.r.t. a yields:

$$a = f(x_1) - f'(x_1)x_1 \tag{3.2}$$

Now that we have expressions for a and b, we have the equation of the tangent:

$$y = f(x_1) - f'(x_1)x_1 + f'(x_1)x \tag{3.3}$$

The next point in the iteration, i.e., x_1, has coordinates $(x_2, 0)$. All we need is to substitute this in 3.3 and solve w.r.t. x_2:

$$x_2 = x_1 - \frac{f(x_1)}{f'x_0} \tag{3.4}$$

As it becomes obvious, it is fundamental that the function is differentiable in the search space. The conditions for convergence in the Newton-Rhapson algorithm are laid down in Theorem **??**:

THEOREM 5. *Let $f \in C^1[a,b] : f'(x) \neq 0, \forall x \in [a,b]$. Suppose that $\exists x_n : f(x_n) = 0$. Then, for any $x_k \in [a,b]$, the sequence $x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}$ converges to x_n.*

The code below implements the algorithm above. Since we need to compute the derivative, I start by creating a function that uses the aproximation of the derivative given in 1.5:

```
function d=deriv(f,x0,h)

% This function takes as arguments the C2 function f, a point x0 in the
% domain of f and a perturbation scalar h
% the output d is the numerical approximation of the derivative of f at the
% point x0

d = (f(x0+h)-f(x-h))/(2*h);
```

We can now write our routine for the Newton-Rhapson algorithm. Let's write it in the context of the following exercise:

EXERCISE 7. *Let* $f(x) = 1 - 0.5x^{-0.5} - 0.3x^{-0.2}$. *Find* $x : f(x) = 0$.

The below function implements the Newton-rhapson algoritm, iterates less than 1000 times and yields a message with the root and the number of iterations or that it couldn't find the root in the iteration limit.

```
function root = nwt(f,x0,h)

% This function takes as inputs a function f, scalar guess for the root of
% f, x0 and scalar perturbation h to compute the derivative.
% The output is the numerical approximation to the root of f, following the
% Newton-Rhapson method.

i   = 1;

while i <= 999 && abs(f(x0))>h
    x0 = x0 - f(x0)/deriv(f,x0,h);
    i = i + 1;
end

if f(x0) > h || isnan(x0) == 1
    display('Root not found. Maximum number of iterations reached!')
elseif f(x0)<=h
    root = x0;
    display(['Root found in ' num2str(i) ' iterations'])
end
```

As before, to compute the root of the equation above, just type

```
>> nwt(@(x) 1-0.5*x^-0.5-0.3*x^-0.2,1,10^-6)
Root found in 6 iterations

ans =

    0.5671
```

Notice however that the algorithm is sensitive to initial conditions. By changing our initial guess to $x_1 = 2$ for example, the algorithm does not converge:

```
>> nwt(@(x) 1-0.5*x^-0.5-0.3*x^-0.2,2,10^-6)
Root not found. Maximum number of iterations reached!
```

This algorithm is generalizable for real-value functions of many variables. Instead of the derivative, we need to compute its multidimensional equivalent, the gradient. There is a function in MatLab that implements the Newton-Rhapson, `fsolve.m`.

CHAPTER 4

Optimization

In this chapter we will focus on methods designed to find extreme values of real valued functions with respect to a finite set of variables. This class of problems is arguably the most common in economics. We will start by introducing derivative-free algorithms in a one-dimensional context: the golden search and explore in which circumstances it makes sense to use grid-search methods. We will then move to derivative based approaches and generalize to multidimensional problems.

An inital note of caution though. Apart from the grid search method, all other methods are designed to find local extreme values. In economics, we often want to maximize continuous concave functions over compact and convex domains, which ensures existence and uniqueness. In these cases, we are sure that an optimum found is indeed global. One should be well aware of the structure of the problem, to make sure that each method is appropriate and that adequate conclusions can be drawn from the results.

1. Grid Search

The name of the method is self-explanatory. If we want to find a maximum of a function in a given interval, one way of getting an approximate solution is to discretize the domain with a grid, evaluate the function at each gridpoint and then check which gridpoint yielded the maximum. Whether this is an appropriate method or not depends on the structure of the problem and on economic theory. To understand why it is the case, let's look at the following problem:

EXERCISE 8. *Assume that tax revenue for a government is given by* $R(t) = -t^2 + t$*, where* t *is a tax rate. Use the grid search method and find the percentage tax rate that maximizes tax revenue up to* 10^{-2} *precision.*

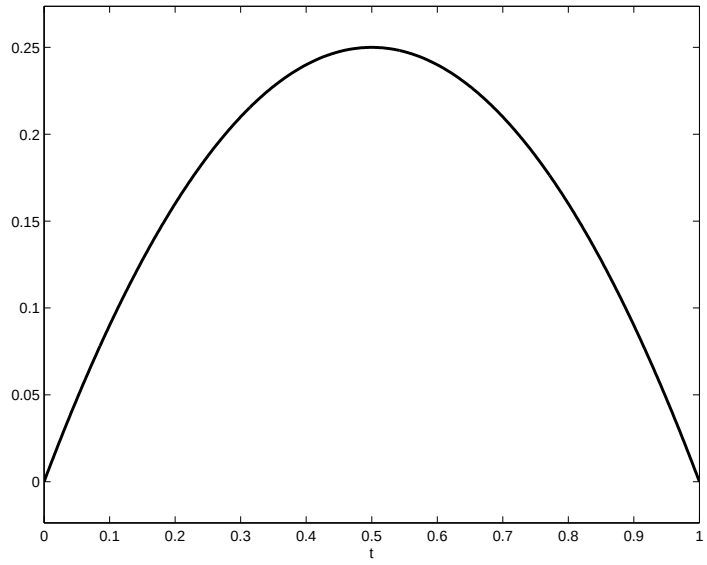
There are a couple of features of the problem that show the convenience of the grid search method in this case. First, the optimal tax rate must be between 0 and 1. This could be argued on economic grounds, that for the problem at hand, only tax rates in the $[0, 1]$ interval are meaningful. The tax revenue function actually gives zero revenue for $t = 0$ and $t = 1$. Also, one might argue that, for reality sakes, we can only set percentage tax rates up to two decimals precision which would set our desired precision level at 10^-4.

Let's look at the plot of the tax revenue function in Figure 1 below:

Implementing a grid search algorithm in this case is pretty straightforward:

```
R = @(t) -t.^2+t; %creates the Revenue function

t = 0:10^-4:1; % discretizes the domain up to 10^-4 precision
y = R(t); % computes the Revenue for each tax rate
```

FIGURE 1. Tax Revenue: $R(t) = -t^2 + t$




```
[maximum, index] = max(y); % finds the maximum revenue and the index
argmax = t(index); % of the element in the t vector that maximizes revenue
```

The solution is to choose a tax rate of $t = 0.5$ which yields a revenue of $R(0.5) = 0.25$.

One advantage of this method is that, for the desired precision level along the argument dimension, it always yields a global maximum and it does not require any particular property of the objective function, such as continuity or differentiability. More often than not though, we need to look for optimal values of a function for which the bounds of the domain are so big relative to the desired precision that grid search is too slow to be feasible.

Though the method is easily generalizable for higher dimensions, the computational requirements of grid search grow exponentially with the dimension of the problem. For a two variable version of our problem, even assuming two tax rates (i.e. two variables bounded on the unit interval) with a desired level of precision of 10^{-4}, this would imply to evaluate a revenue function at $(10^4 + 1)^2$ points i.e. all combinations of both tax rates. For a similar problem with n variables, we would need a grid of $(10^4 + 1)^n$ points, a task which quickly becomes unfeasible.

2. Golden Search

The golden search algorithm doesn't require either the computation of any derivative. For a continuous function over a compact domain, the Weierstrass Theorem assures the existence of at least a maximum. How can we use the golden search algorithm to find it? Let's start by stating the following theorem:

THEOREM 6. *Let $f : X \to Y, X, Y \subset \mathbb{R}$ be a continuous function over the interval $[a, b], a < b$ and $a, b \in X$. Let $x_1^l, x_2^h \in [a, b] : x_1^l < x_2^h$. If $f(x_1^l) < f(x_1^h)$*

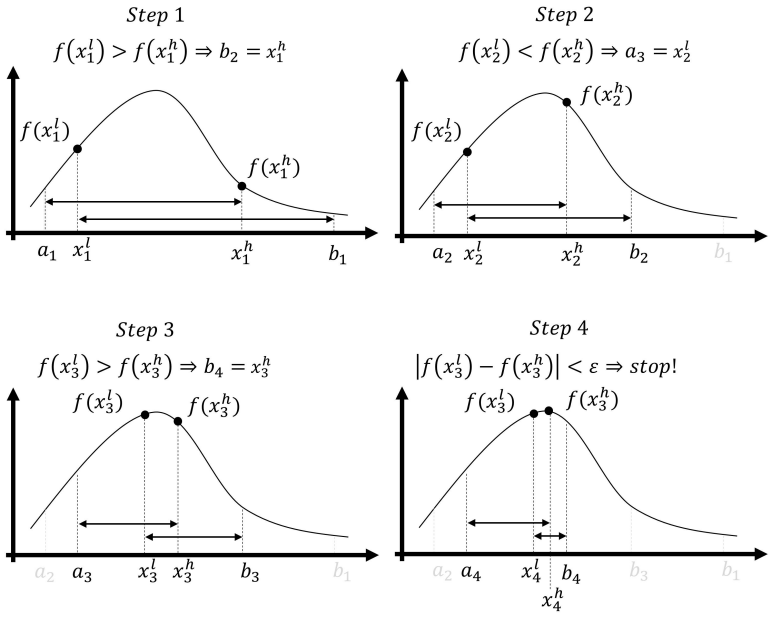
then there is at least a maximum in the interval $[x_1^l, b]$. *Conversely, if* $f(x_1^l) > f(x_1^h)$ *then there exists at least a maximum in the interval* $[a, x_1^h]$.

Once we have our new interval, the lower bound becomes our new a and the upper bound our new b, though only one gets updated every iteration. Now it is time to pick another two points inside the new interval. For computational efficiency, only the point who became the new upper/lower bound gets a new guess.

We now evaluate the function at the two interior points, check which one is greater and define a new (shorter) interval. We proceed until the distance between the two interior points is less than a pre-specified criteria. When that is the case, we simply pick the highest value as the maximum.

The Figure 2 shows how the algorithm works.

FIGURE 2. Golden Search Algorithm



We now improve the algorithm by providing a simple rule regarding the choice of the two interior points. The rule is, at the k^{th} iteration, to choose x_k^l, x_k^h such that:

$$x_k^l = a_k + \frac{3-\sqrt{5}}{2}(b_k - a_k) \tag{2.1}$$

$$x_k^h = a_k + \frac{\sqrt{5}-1}{2}(b_k - a_k) \tag{2.2}$$

The fraction in the second equation is known as the golden ratio. Below the code that implements a function that takes as arguments the function `f`for which we want to find the maximum and scalars `a` and `b` that define the interval in which we search for the maximum. the routine will, up to a precision of 10^{-10}, find a value in the interval $[a, b]$ such that its value is a local maximum.

```
function m = gsearch(f,a,b)

 xl = a+(3-5^.5)/2*(b-a); fl = f(xl);
 xh = a+(5^.5-1)/2*(b-a); fh = f(xh);
  d = abs(xh-xl);
tol = 10^-10;

while d > tol
    d = abs(xl-xh);
    if f(xl)<f(xh)
        a = xl;
        xl = a+(3-5^.5)/2*(b-a); fl = f(xl);
        xh = a+(5^.5-1)/2*(b-a); fh = f(xh);
    else
        b = xh;
        xl = a+(3-5^.5)/2*(b-a); fl = f(xl);
        xh = a+(5^.5-1)/2*(b-a); fh = f(xh);
    end
end

if fh>fl
    m = [fh,xh];
else
    m = [fl,xl];
end
```

Let's put our function to the test by solving the next exercise.

EXERCISE 9. *Let f be a real valued function such that $f(x) = xcos(x^2)$. Find the maximum of the function using the golden search rule in the interval $[0, 3]$, up to 10^-10 precision.*

A direct application of the above function yields the value of $x = 0.8083$ and $f(x) = 0.6418$. Remember however that the golden search algorithm will only find a local maximum, not a global one. Let's plot $f(x)$ in the above interval.

The algorithm successfully found a local maximum, but not the global one. Had we provided the interval $[2, 3]$ then the `gsearch.m`would have found the global maximum of 2.51 at $x = 2.52$. One should always be aware of the assumptions and limitations of each algorithm.

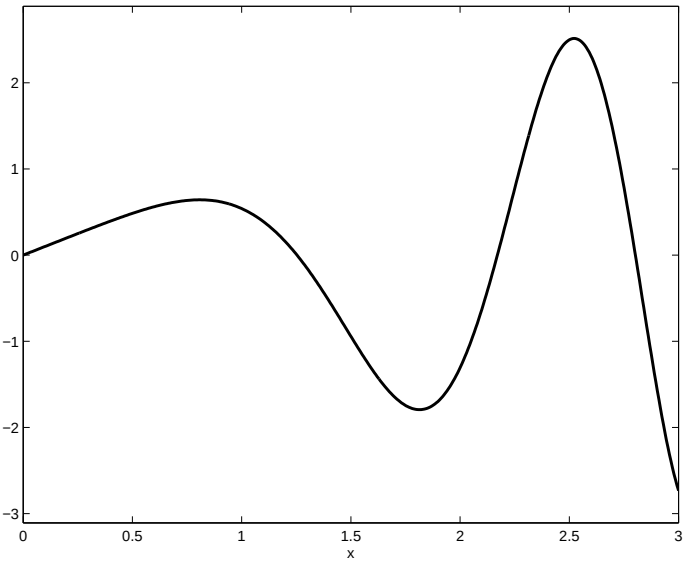
3. Newton-Rhapson

The gradient method is just the generalization the of Newton-Rhapson to real valued functions of n variables. Let's start with our definition of gradient:

DEFINITION 3. *Let $f : \mathbb{R}^n \to \mathbb{R}, f \in \mathbb{C}^2$. Then, $\exists \nabla f : \mathbb{R}^n \to R, \nabla f(x) = \left(\frac{\partial f(\boldsymbol{x})}{\partial x_1}, ..., \frac{\partial f(\boldsymbol{x})}{\partial x_n}\right)$.*

Notice that we do not need twice continuous differentiability to be able to define a gradient function, but we will need it in order to implement our gradient-based optimization method.

FIGURE 3. $xcos(x^2)$



Before using the gradient method directly to search for the maximum of a function, we will start by solving an optimization problem by finding the root of the first order conditions. Let's look at the below exercise:

EXERCISE 10. *Assume an agent lives for two periods, faces the budget constraints and derives utility from consumption as specified below. Find the optimal consumption schedule.*

$$\max_{c_1,c_2} U(c_1,c_2) = \frac{c_1^{1-\mu}}{1-\mu} + \beta\frac{c_2^{1-\gamma}}{1-\gamma} \tag{3.1}$$

$$c_1 = y - s \tag{3.2}$$

$$c_2 = y + Rs \tag{3.3}$$

Consolidate the budget constraints by substituting away s and combine the first order conditions w.r.t. c_1, c_2 to get:

$$c_1^{-\mu} - \beta R c_2^{-\gamma} = 0 \tag{3.4}$$

$$c_1 + \frac{c_2}{R} - y - \frac{y}{R} = 0 \tag{3.5}$$

This is a system of non-linear equations and can be solved by some of the methods introduced earlier. We will use MatLab's implementation of the Newton-Rhapson algorithm introduced earlier, the function `fsolve.m`.

We start by creating an m-file where the equations are written in the form of $f(\mathbf{x}) = 0$. Let's call this file `auxg.m`:

```
function res = auxg(x)

global mu beta gamma R y % the parameters to call from the base environment
```

```
c1 = x(1); c2=x(2); % our guess for the three variables

e =  zeros(2,1)*NaN;

e(1) = c1.^-mu-beta*R*c2.^-gamma;
e(2) = c1+c2./R-y-y./R;

res = e; % if our guess solves our problem, then e = zeros(2,1)
```

Now, all we need to do is to specify in the base environment the parameters, come up with a guess and use `fsolve.m`to solve the problem:

```
global mu beta gamma R y

mu = 1.1; beta = 0.95; gamma = 1.2; R = 1.05; y = 10;

% solve the problem by solving a 2 non-linear equation system
sol = fsolve('auxg',[3 3]);
c1 = sol(1); c2 = sol(2); [c1 c2]

ans =

   10.9784    8.9727
```

We have then that our solution is to choose $c_1 = 10.9784, c_2 = 8.9727$. We do not check for the second order condition, because the objective function is concave in c_1, c_2. Here we used a Newton-Rhapson based method, which involves the computation of a gradient, but to solve the optimization problem by solving a system of non-linear equations, the first order conditions. Let's see how we could have used the same method without computing first order conditions first.

Notice that the problem above can be restated as a one variable optimization problem. Just replace c_1, c_2 in the objective function making use of the budget constraints. We now have a one variable optimization problem i.e. we need to choose how much to save, in order to maximize the value function below:

$$\max_s U(y-s, y+Rs) = \frac{(y-s)^{1-\mu}}{1-\mu} + \beta \frac{(y+Rs)^{1-\gamma}}{1-\gamma} \tag{3.6}$$

This example will make it clear in which sense the Newton-rhapson method of solving an equation is basically the same problem as finding a stationary point of a function. As hinted in the previous example, it so happens that finding the stationary point of a function is the same as finding the root of the first order derivative. We can then write a optimization version of the Newton-Rhapson theorem in 5:

THEOREM 7. *Let $f \in C^2[a,b] : f''(x) \neq 0, \forall x \in [a,b]$. Suppose that $\exists x_n : f'(x_n) = 0$. Then, for any $x_k \in [a,b]$, the sequence $x^{k+1} = x^k - \frac{f'(x^k)}{f''(x^k)}$ converges to x_n.*

Implementing this leads to our new version of `nwt.m`, named `maxnwt.m`, designed to find stationary points and check whether it is a maximum, saddle point or a minimum, based on the sign of the second derivative:

```
function m = maxnwt(f,x0,h)
% This function takes as inputs a function f, scalar guess for the  of
% stationary point of f x0, and scalar perturbation h to compute the
% derivative.
% The output is the numerical approximation to the stationary point of f,
% following the Newton-Rhapson method.

df = @(x) deriv(f,x,h);
i   = 1;

while i <= 999 && abs(df(x0))>h
    x0 = x0 - df(x0)/deriv(df,x0,h);
    i = i + 1;
end

if df(x0) > h || isnan(x0) == 1
    display('Stationary point not found. Maximum number of iterations reached!')
elseif df(x0)<=h
    m = x0;
    display(['stationary point found in ' num2str(i) ' iterations'])
    if deriv(df,x0,h) < -h
        display('Second derivative < 0. Local maximum found!')
    elseif abs(deriv(df,x0,h)) < h
        display('Second derivative = 0. Inflection point!')
    elseif deriv(df,x0,h) > h
        display('Second derivative > 0. Local minimum found!')
    end
end
end
```

To use this function, one needs to setup a value function, which we obtain by substituting the constraints in the objective function, substituting away consumption, and having a one variable maximization problem in s. Solving the problem this way leads to a solution that is identical to the one before, at least to a four digit precision in this case.

Lastly, one could have used MatLab's native function `fminsearch.m`. As the function's name hints, it searches for a minimum. If we want to maximize, all we need to do is to ask it to find the minimum of the symmetric value function i.e.:

```
%solve the problem using fminsearch.m
s = fminsearch(@(s) ...
    -(((y-s).^(1-mu))./(1-mu)+beta*((y+R*s).^(1-gamma))./(1-gamma)),1);
c1d = y-s; c2d = y-R*(c1d-y);
```

4. Constrained optimization

To solve the previous exercises, we substituted the constraints into the objective function or in the first order conditions of the problem. In this way, we avoided dealing with them explicitely. However, we were allowed to do it because the constraints are satisfied with equality. In general that may not be the case. An example often used is when we want to create an environment where agents are not allowed to borrow. In this case we want to add the constraint that asset holdings cannot be negative, an inequality constraint. Remember also that for the budget constraint, we claim it holds with equality based on a non-satiation argument i.e. that the marginal utility is strictly positive. Again, that may not always be the case.

In this section we will introduce two methods of dealing with inequality constraints. The first one is based on the Karush-Kuhn-Tucker theorem. The second will make use of a penalty function.

4.1. Karush-Kuhn-Tucker method. As said before when using the Lagrange formulation, we assume interior solutions in the sense that the constraints are satisfied with equality. Let's restate the previous savings problem in Exercise 10, but now imposing a no-borrowing constraint:

$$\max_{c_1,c_2} U(c_1,c_2) = \frac{c_1^{1-\mu}}{1-\mu} + \beta\frac{c_2^{1-\gamma}}{1-\gamma} \tag{4.1}$$

$$c_1 = y - s \tag{4.2}$$

$$c_2 = y + Rs \tag{4.3}$$

$$s \geq 0 \tag{4.4}$$

We can no longer substitute away s because of the inequality. Let's build the Lagrange function:

$$\mathcal{L} = \frac{c_1^{1-\mu}}{1-\mu} + \beta\frac{c_2^{1-\gamma}}{1-\gamma} + \lambda_1\left(y - s - c_1\right) + \lambda_2\left(y + Rs - c_2\right) + \lambda_3 s \tag{4.5}$$

The first order conditions w.r.t. $c_1, c_2, s, \lambda_1, \lambda_2$ and λ_3 respectively are:

$$c_1^{-\mu} - \lambda_1 = 0 \tag{4.6}$$

$$\beta c_2^{-\gamma} - \lambda_2 = 0 \tag{4.7}$$

$$-\lambda_1 + R\lambda_2 + \lambda_3 = 0 \tag{4.8}$$

$$y - s - c_1 = 0 \tag{4.9}$$

$$y + Rs - c_2 = 0 \tag{4.10}$$

$$\lambda_3 s = 0 \tag{4.11}$$

The system is determinate, we have 6 equations in six unknowns. The issue is to look for the solution such that $\lambda_3 \geq 0$ if $s \leq 0$ so that the no borrowing constraint holds. Notice that now the previous version of the Euler equation will not hold with equality in general. The agent will no longer be able to smooth consumption if he is liquidity constrained. To see this, notice that the system above is equivalent to:

$$c_1^{-\mu} - R\beta c_2^{-\gamma} + \lambda_3 = 0 \tag{4.12}$$

$$y - s - c_1 = 0 \tag{4.13}$$

$$y + Rs - c_2 = 0 \tag{4.14}$$

$$\lambda_3 s = 0 \tag{4.15}$$

Notice how if $\lambda_3 > 0$, the last condition implies that the agent is liquidity constrained i.e. $s = 0$ and that no longer $c_1^{-\mu} = R\beta c_2^{-\gamma}$ as before.

We can use a modified version of the `auxg.m` function, which we will call `auxg2.m`. The code is shown below:

```
function res = auxg2(x)

global mu beta gamma R y % the parameters to call from the base environment

c1 = x(1); c2=x(2); s = x(3);% our guess for the three variables
lambda1 = x(4); lambda2 = x(5); lambda3 = x(6);

e =  zeros(6,1)*NaN;

e(1) = c1.^-mu-lambda1;
e(2) = beta*R*c2.^-gamma-lambda2;
e(3) = -lambda1+R*lambda2+max(0,lambda3);
e(4) = y-s-c1;
e(5) = y+R*s-c2;
e(6) = s - max(0,-lambda3);

res = e; % if our guess solves our problem, then e = zeros(6,1).
```

The way to introduce a multiplier that must be equal or greater than zero is by making use of the $\max(0, \lambda_3)$ term. Notice that if the agent is liquidity constrained, then $c_1^{-\mu} < R\beta c_2^{-\gamma}$ and consequently $\lambda_3 > 0$ and $s = 0$. This means that $\max(0, \lambda_3) > 0$ and that $\max(0, -\lambda_3) = 0$ implies $s = 0$, by the sixth equation in the `auxg2.m` file.

Using `fsolve.m`to solve the problem yields the solution:

```
sol = fsolve('auxg2',ones(6,1))

sol =

   10.0000
   10.0000
         0
    0.0794
    0.0629
    0.0133
```

As we can see, the agent is liquidity constrained, will consume all her income in each period. Savings are zero and the marginal utility of relaxing the no-borrowing constraint by one unit is equal to 0.0133.

4.2. The Penalty Function method. As the name suggests, this method involves the specification of a penalty function. We will use this with hill-climbing algorithms by redefining the objective function in order to penalize solutions that are outside our constrained choice set, such that they will never be optimal. Let's restate the problem in 3.6:

$$\max_s U(y-s, y+Rs) = \frac{(y-s)^{1-\mu}}{1-\mu} + \beta\frac{(y+Rs)^{1-\gamma}}{1-\gamma} - 1000\max(0,-s) \tag{4.16}$$

It is clear how choosing negative savings is never optimal in this context. To solve this we can create a value function file `valfunc.m`:

```
function sol = valfunc(s)

global mu beta gamma R y

sol = ((y-s).^(1-mu))./(1-mu)+beta*((y+R*s).^(1-gamma))./(1-gamma)...
    -1000*max(0,-s);
```

and then just use `fminsearch.m` as before:

```
>> s = fminsearch(@(s) -valfunc(s),1)

s =

  -8.8818e-16
```

5. Value function

Let us consider the following value function

$$\begin{aligned} V(x_t, z_t) = &\max_{u_t}\{R(x_t, z_t, u_t) + \beta E_t V(x_{t+1}, z_{t+1})\} \\ &s.t.\ x_{t+1} = g(x_t, z_t, u_t) \end{aligned} \tag{5.1}$$

where x_t is a vector of deterministic state variables, u_t a vector of control variables, z_t a stochastic state variable and $R(x_t, u_t)$ the return function, and we are assuming that the state vector evolves according to $x_{t+1} = g(x_t, z_t, u_t)$. E_t is the expectation operator over the sequence of possible states of z.

5.1. Markov Chains. The first thing we would need to do is discretize the state space s=[x,z]. If we assume z follows AR(1) we can discretize the z using a Markov chain. Tauchen (1986) shows that we can approximate an AR(1) process with a Markov chain.

But what is a Markov chain? A Markov chain is a stochastic process that satisfies the Markov property. The Markov property is defined as:

DEFINITION 4. *A stochastic process z_t satisfies the Markov property if for all $i \geq 2$ and all t*

$$Prob(z_{t+1}|z_t, z_{t-1}, ..., z_{t-i}) = Prob(z_{t+1}|z_t)$$

this is, the probability of moving to the next state depends only on the state today and not on previous states.

A Markov chain is characterized by:

- an n-dimensional vector $z \in \mathbb{R}^n$
- a transition matrix Π (n x n)

$$\Pi_{ij} = Prob(z_{t+1} = z_j | z_t = z_i)$$

  with $\sum_{j=1}^{n} \Pi_{ij} = 1$ to be valid.
- a vector ξ (n x 1) with the probabilities of being in state i at time 0

$$\xi_{0i} = Prob(z_0 = z_i)$$

  with $\sum_{i=1}^{n} \xi_{0i} = 1$ to be valid.

Also, to be a valid Markov chain, the probability of being in any state in two period is Π^2

$$Prob(z_{t+2} = z_j | z_t = z_i) =$$
$$\sum_{h=1}^{n} Prob(z_{t+2} = z_j | z_{t+1} = z_h) Prob(z_{t+1} = z_h | z_t = z_i) =$$
$$\sum_{h=1}^{n} \Pi_{ih} \Pi_{hj} = \Pi_{ij}^2$$

5.2. Value function iteration. Now that we know how to discretize the stochastic process z_t we can solve our value function. Assume z follows an auto-regressive process

$$z_{t+1} = \rho z_t + \epsilon_{t+1}$$

Following Tauchen, z can be approximated by a Markov process. Let us assume it can be approximated by a Markov process with two states ($n^z = 2$) and transition matrix

$$\Pi = \begin{bmatrix} \Pi_{11} & \Pi_{12} \\ \Pi_{21} & \Pi_{22} \end{bmatrix}$$

V is going to be an n^z x n^x matrix. The Bellman equation is given by

$$V_{i,r} = \max_{s \in \{1,2,...,n^x\}} \{R(z_i, x_r, x_s) + \beta \sum_{j=1}^{n^z} \Pi_{ij} V_{j,s}\}$$

How do we find the value for V? Under the conditions that $R(x, x', z)$ is concave and $\{(x', x, z) : x' \leq g(x, z, u), u \in \mathbb{R}^{\phi}\}$ is convex and compact the Bellman equation is going to have a fixe point and V is going to be a contraction. So, iterating on

$$V_{i,r}^{(n+1)} = \max_{s\in\{1,2,\ldots,n^x\}} \{R(z_i, x_r, x_s) + \beta \sum_{j=1}^{n^z} \Pi_{ij} V_{j,s}^n\}$$

we will converge to V. So, to find z

I. Guess the initial value function V^0;

II. Find the x_s that maximizes $V_{i,r}^{n+1}$

$$V_{i,r}^{n+1} = max\{V_{i,r,1}^{n+1}, V_{i,r,2}^{n+1}, ...V_{i,r,n^x}^{n+1}\}$$

Repeat this for all combinations of {i,r} to find V^{n+1}. The optimal x_s decision for each combination {i,r} will give us the policy function.

III. Calculate the distance between V^n and V^{n+1}

$$d = |V^{n+1} - V^n|$$

If $d > \epsilon$ go back to point II. for n+2, n+3, n+4,... until $d < \epsilon$.

Part 2

Applications in Macroeconomics

CHAPTER 5

RBC model

1. Closed form solution

We start by considering the model by Brock and Mirmam (1972). The utility of the representative agent is

$$u(c_t) = \ln c_t$$

The output in the economy, y_t, is produced according to the following technology

$$f(k_t, \theta_t) = \theta_t k_t^{\alpha},\ \alpha \in (0, 1)$$

with θ_t and k_t being the level of technology and capital in the economy. Capital law of motion is given by

$$k_{t+1} = (1 - \delta)k_t + i_t$$

where δ is the depreciation and i_t is investment. The resource constraint in this economy is given by

$$y_t = c_t + i_t$$

Assuming $\delta = 100\%$ the model as closed form solution. Then using the capital law of motion and the production technology we can rewrite the resource constraint as

$$k_{t+1} = \theta_t k_t^{\alpha} - c_t$$

If θ_t is deterministic there is no risk in the economy and the problem of the representative agent is to maximize is lifetime utility, being subject to the resource constraint. So it can be written as

$$\sum_{j=t}^{\infty} \beta^{j-t} \ln c_j$$
$$s.t. k_{t+1} = \theta_t k_t^{\alpha} - c_t$$

In this case, replacing c with the resource constraint, the value function would be

$$V(k) = \max_{k'} \{ln(\theta k^{\alpha} - k') + \beta V(k')\} \tag{1.1}$$

Although, if we assume θ_t is stochastic and follows an AR(1) process given by

$$\ln \theta_t = (1 - \rho) \ln \bar{\theta} + \rho \ln \theta_{t-1} + \epsilon_t$$

the representative agent's problem will become

$$E_t \sum_{j=t}^{\infty} \beta^{j-t} \ln c_j$$
$$s.t. k_{t+1} = \theta_t k_t^{\alpha} - c_t$$

and the value function will become

$$V(k,\theta) = \max_{k'} \{ln(\theta k^{\alpha} - k') + \beta E V(k', \theta')\} \tag{1.2}$$

with E denoting expectation. The first order condition of the household's problem is

$$\frac{1}{\theta k^{\alpha} - k'} = \beta E V_k(k', \theta')$$

Using the envelope theorem we get

$$V_k(k,\theta) = \frac{\alpha \theta k^{\alpha-1}}{\theta k^{\alpha} - k'}$$

combining the previous two equations the Euler equation becomes

$$\frac{1}{\theta k^{\alpha} - k'} = \beta E \frac{\alpha \theta k'^{\alpha-1}}{\theta k'^{\alpha} - k''}$$

Assuming the policy function is $k' = \phi(k,\theta)$ if we guess and verify the policy function we would get

$$k' = \beta \alpha \theta k^{\alpha}$$
$$c = (1 - \beta\alpha)\theta k^{\alpha}$$

And the value function would be given by

$$\begin{aligned} V(k,\theta) = &\frac{\alpha}{(1-\beta\alpha)} \ln k + \frac{1}{(1-\beta\alpha)} \frac{1}{(1-\beta\rho)} \ln \theta + \frac{1}{(1-\beta)} \ln(1-\beta\alpha) \\ &+ \frac{\beta}{(1-\beta\rho)} \frac{1-\rho}{(1-\beta\alpha)} \ln \bar{\theta} + \beta \frac{\alpha}{(1-\beta\alpha)} \ln \beta\alpha \end{aligned}$$

2. Numerical methods

If the assumptions of log utility and of full depreciation are not satisfied, the model does not have a closed form solution. In this case we would have to use numerical methods to find the value function and the policy function. Let us consider the following household's problem, where z is a stochastic variable

$$\max_{\{c_t, k_{t+1}\}_{t=0}^{\infty}} E_0 \sum_{t=0}^{\infty} \beta^t \frac{c^{1-\sigma}}{1-\sigma}$$
$$s.t. c_t + k_{t+1} = z_t k_t^{\theta} + (1-\delta) k_t$$
$$z_{t+1} = \rho z_t + \epsilon_{t+1}$$

where ϵ_{t+1} is a shock. We can formulate this problem either using the Lagrangian or the value function. Formulating this problem using the Lagrangian, the Lagrangian and the corresponding first order conditions are

$$E_0\left\{\sum_{t=0}^{\infty}\beta^t\frac{c^{1-\sigma}}{1-\sigma}-\lambda_t[c_t+k_{t+1}-z_tk_t^\theta-(1-\delta)k_t]\right\}$$

$$\partial c_t:\ \beta^t c_t^{-\sigma}=\lambda_t \tag{2.1}$$

$$\partial k_{t+1}:\ \lambda_t=E_t\lambda_{t+1}[\theta z_{t+1}k_{t+1}^{\theta-1}+(1-\delta)] \tag{2.2}$$

$$\partial\lambda_t:\ c_t+k_{t+1}=z_tk_t^\theta+(1-\delta)k_t \tag{2.3}$$

Combining equations 2.1 and 2.2 we get

$$1=\beta E_t\frac{c_{t+1}}{c_t}\theta z_{t+1}k_{t+1}^{\theta-1}+(1-\delta)$$

which evaluated at the steady state, normalizing the steady state of z to one, gives

$$\frac{k}{y}=\frac{\beta\theta}{1-\beta(1-\delta)}$$

$$y=\frac{k}{y}^{\frac{\theta}{1-\theta}}=\frac{\beta\theta}{1-\beta(1-\delta)}^{\frac{\theta}{1-\theta}}$$

$$k=\frac{k}{y}y=\frac{\beta\theta}{1-\beta(1-\delta)}^{\frac{1}{1-\theta}}$$

$$c=y-\delta k=\frac{\beta\theta}{1-\beta(1-\delta)}^{\frac{1}{1-\theta}}\frac{1-\beta(1-\delta)-\beta\theta\delta}{\beta\theta}$$

If we instead formulate the problem using the value function we would get

$$V(z,k)=\max_{k'}\left\{\frac{[zk^\theta+(1-delta)k-k']^{1-\sigma}}{1-\sigma}+\beta EV(z',k')\right\} \tag{2.5}$$

EXERCISE 11. *Assuming a $\sigma=2$, $\theta=0.4$, $\delta=0.1$, $\beta=0.98$, and that the number of possible states for z is three, find the value function that solves the households' problem and simulate the deviations from the steady state values during 1000 periods.*

To solve the previous exercise we first start by discretizing the state space s=[k,z]. We already now that we can approximate the AR(1) process of z by a markov chain using the Tauchen matlab code. When discretizing k we can either choose to have nk (size of k grid) large and evaluate the value function at each point in the k grid and find the value of k' that maximizes the value function, or we can have nk small and use Golder Search and interpolation techniques to find the

value of k' that maximizes the value function. Then we need to iterate the value function until V(k,z)=V(k',z').

The method of having a large nk is ideal to parallelize in the GPU processer that has a large number of cores that can solve simple tasks simultaneously, and in a cycle can evaluate the value function in as many points as the number of cores. Using a small nk with Golden Search and interpolation techniques should be parallelized in the CPU, given that each task is complex and the CPU can solve as many tasks simultaneously as the number of cores it has. The final result of both techniques is the same, but depending on the CPU and the GPU of the computer, it can be faster to solve either on the CPU or in the GPU.

The code that solves the problem with a large nk is the following. Note that you need the markov.m and tauchen.m matlab functions to run the code.

```
clear all; close all; clc;

%Parameters
theta = 0.4;  %decreasing returns to scale
delta = 0.1;  %capital depreciation rate
sigma = 2;    %coefficient of relative risk aversion
beta  = 0.96; %time discount variable
nk    = 1000; %number of k grid points
nz    = 7;    %number of possible states for the economy
a     = 0;
rho   = 0.9;  %persistency of the shock
stdz  = 0.05;  %standard deviation of the shocks
m     = 1;
simul = 1000; %number of simulations

%Discretizing the z grid
[z,zprob] = tauchen(nz,a,rho,stdz,m);
z=exp(z);

%steady state values

ky_ss =(beta*theta)/(1-beta*(1-delta));
y_ss  =((beta*theta)/(1-beta*(1-delta)))^(theta/(1-theta));
k_ss  =((beta*theta)/(1-beta*(1-delta)))^(1/(1-theta));
c_ss  =((beta*theta)/(1-beta*(1-delta)))^(1/(1-theta))*(1-beta*(1-delta)-beta*...
theta*delta)/(beta*theta);

%Discretizing the k grid
kmin = 0.5*k_ss;
kmax = 1.5*k_ss;
k=kmin:(kmax-kmin)/nk:kmax-(kmax-kmin)/nk;

%value function initial values

u    = c_ss^(1-sigma)/(1-sigma);
V    = ones(nz,nk)*u/(1-beta); %value function in the steady state
```

```
v    = zeros(nz,nk);
g    = zeros(nz,nk);
newV = zeros(nz,nk);

%Tolerance levels
tolv    = 10^-4;
maxiter = 2000;
iter    = 0;
diffv   = 1;
%Value function iteration


while diffv>tolv && iter<maxiter
iter=iter+1;
for iz=1:nz
parfor ik=1:nk %parallelize the code in k
c = z(iz)*k(ik)^theta+(1-delta)*k(ik)-k; %consumption evaluated at
c = max(c,1e-8); %to prevent c from being negative
v = (c.^(1-sigma))/(1-sigma)+beta*zprob(iz,:)*V(:,:);
[newV(iz,ik),index] = max(v);
g(iz,ik) = k(index);
end
end
diffv = norm(newV-V)
V  = newV;
end


figure(1)
plot(k,g(:,:))
title('Policy funtions')

%Simulation

zsim    = ones(simul+1,1);
ksim    = zeros(simul+1,1);
ysim    = zeros(simul,1);
csim    = zeros(simul,1);
ksim(1) = k_ss;

zsim(2:end)   = markov(zprob,simul+1,z(2),z);
for i=1:simul
ksim(i+1)   = g(find(z==zsim(i+1)),find(ksim(i)==k));
end
ysim          = zsim(1:end-1).*ksim(1:end-1).^theta;
csim          = ysim+(1-delta)*ksim(2:end)-ksim(1:end-1);

%Plots
```

```
figure(2)
subplot(221)
plot([ksim(1:end-1)/k_ss])
title('Deviations of capital from SS')
subplot(222)
plot([csim/c_ss])
title('Deviations of consumption from SS')
subplot(223)
plot([ysim/y_ss])
title('Deviations of output from SS')

%Correlation between c and y and k and y

[rho1,pval1]=corrcoef(ksim(1:end-1),ysim);
[rho2,pval2]=corrcoef(csim,ysim);
[rho1(2,1) pval1(2,1);rho2(2,1) pval2(2,1)]
```

The code that solves the problem with a small number of nk is the following

```
clear all; close all; clc;

%Parameters
theta = 0.4;  %decreasing returns to scale
delta = 0.1;  %capital depreciation rate
sigma = 2;    %coefficient of relative risk aversion
beta  = 0.96; %time discount variable
nk    = 6; %number of k grid points
nz    = 7;    %number of possible states for the economy
a     = 0;
rho   = 0.9;  %persistency of the shock
stdz  = 0.05;  %standard deviation of the shocks
m     = 1;
simul = 2000; %number of simulations

%Discretizing the z grid
[z,zprob] = tauchen(nz,a,rho,stdz,m);
z=exp(z);

%steady state values

ky_ss =(beta*theta)/(1-beta*(1-delta));
y_ss  =((beta*theta)/(1-beta*(1-delta)))^(theta/(1-theta));
k_ss  =((beta*theta)/(1-beta*(1-delta)))^(1/(1-theta));
c_ss  =((beta*theta)/(1-beta*(1-delta)))^(1/(1-theta))*(1-beta*(1-delta)-beta*...
theta*delta)/(beta*theta);

%Discretizing the k grid
kmin = 0.5*k_ss;
kmax = 1.5*k_ss;
```

```
k=kmin:(kmax-kmin)/nk:kmax-(kmax-kmin)/nk;


%value function initial values

u    = c_ss^(1-sigma)/(1-sigma);
V    = ones(nz,nk)*u/(1-beta); %value function in the steady state
v    = zeros(nz,nk);
g    = ones(nz,nk);
newV = zeros(nz,nk);


%Tolerance levels
tolc    = 10^-4;
tolv    = 10^-4;
maxiter = 2000;
iter    = 0;
diffv   = 1;

%Value function iteration

while diffv>tolv && iter<maxiter
iter=iter+1;
parfor iz=1:nz
for ik=1:nk
%Initial values for the golden search algorithm
c1=0.65*c_ss;
c2=1.35*c_ss;
cl=c1+((3-5^0.5)/2)*(c2-c1);
ch=c1+((5^0.5-1)/2)*(c2-c1);

while abs(c1-c2)>tolc
kpl = -cl+z(iz)*k(ik)^theta+(1-delta)*k(ik);
kph = -ch+z(iz)*k(ik)^theta+(1-delta)*k(ik);
if kpl>max(k)%just to guarantee we do no go to a path where we end with assets higher than th
Vl=-100;
else
Vl  = (cl.^(1-sigma))/(1-sigma)+...
beta*zprob(iz,:)*interp1(k,V(:,:)',kpl,'spline')';
end
if kph<min(k)%just to guarantee we do no go to a path where we end with assets lower than the
Vh=-100;
else
Vh  = (ch.^(1-sigma))/(1-sigma)+...
beta*zprob(iz,:)*interp1(k,V(:,:)',kph,'spline')';
end

if Vl>Vh
```

```
c2=ch;
ch=c1+(5^0.5-1)/2*(c2-c1);
cl=c1+(3-5^0.5)/2*(c2-c1);
else
c1=cl;
ch=c1+(5^0.5-1)/2*(c2-c1);
cl=c1+(3-5^0.5)/2*(c2-c1);
end
end

newV(iz,ik)  = Vl;
g(iz,ik)  = kpl;
end
end
diffv = norm(newV-V)
diff(iter) = diffv;
V  = newV;
end


figure(1)
plot(k,g(:,:))
title('Policy funtions')

%Simulation

zsim    = ones(simul+1,1);
ksim    = zeros(simul+1,1);
ysim    = zeros(simul,1);
csim    = zeros(simul,1);
ksim(1) = k_ss;

zsim(2:end)   = markov(zprob,simul+1,z(2),z);
for i=1:simul
ksim(i+1)   = interp1(k,g(find(z==zsim(i+1)),:),ksim(i));
end
ysim          = zsim(1:end-1).*ksim(1:end-1).^theta;
csim          = ysim+(1-delta)*ksim(2:end)-ksim(1:end-1);

%Plots
figure(2)
subplot(221)
plot([ksim(1:end-1)/k_ss])
title('Deviations of capital from SS')
subplot(222)
plot([csim/c_ss])
title('Deviations of consumption from SS')
subplot(223)
```

```
plot([ysim/y_ss])
title('Deviations of output from SS')

%Correlation between c and y and k and y

[rho1,pval1]=corrcoef(ksim(1:end-1),ysim);
[rho2,pval2]=corrcoef(csim,ysim);
[rho1(2,1) pval1(2,1);rho2(2,1) pval2(2,1)]
```

The correlations between capital and output in both solution methods is around 83% and statistically signficant, and between consumption and output is around 87% in both. Notice that the simulation graphs will change each time you run the code given that z is stochastic. Policy functions and simulations of deviations of consumption, capital and output from the steady state values are presented in figures 1, 2, 3 and 4.

FIGURE 1. CPU's policy function

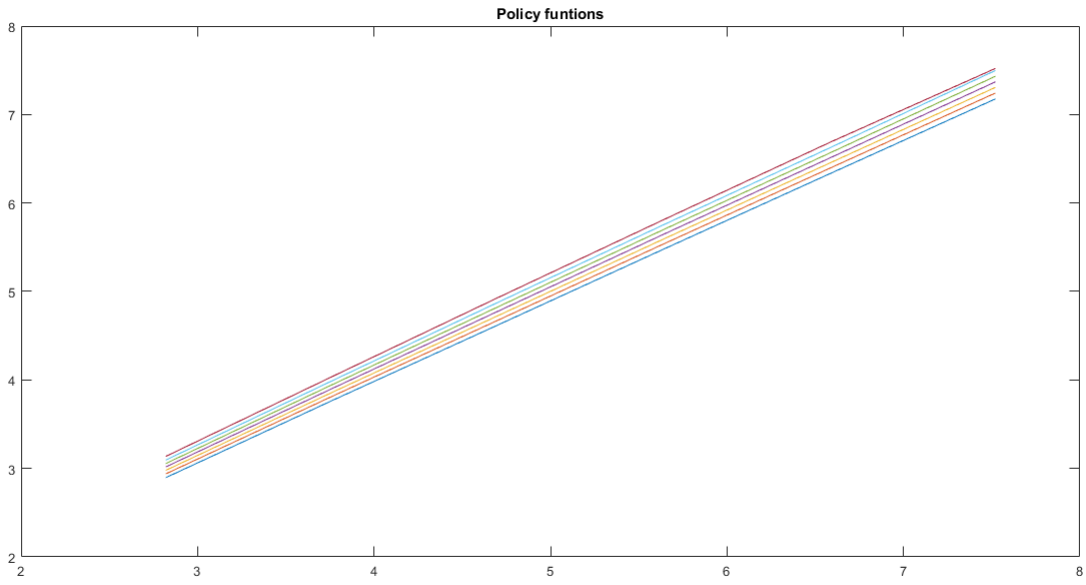


FIGURE 2. GPU's policy function

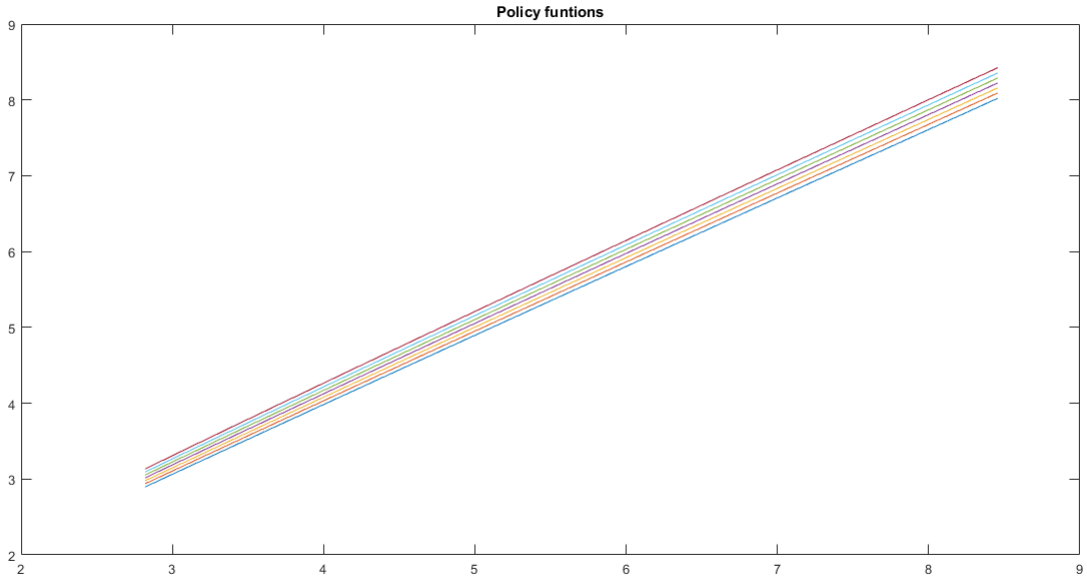
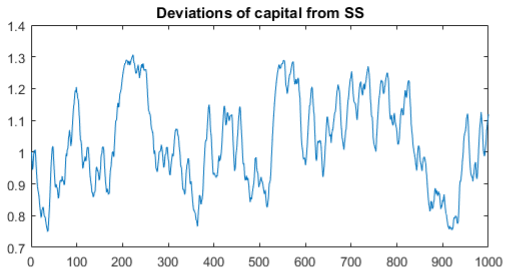
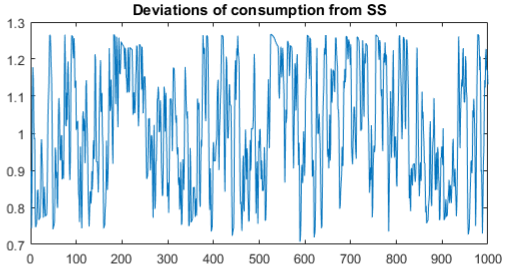


FIGURE 3. CPU's code simulations





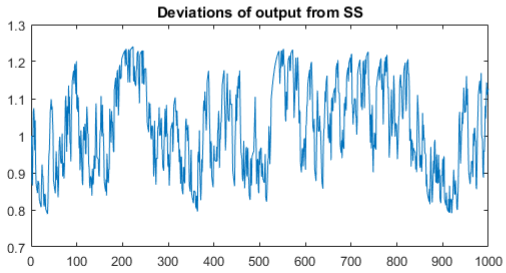
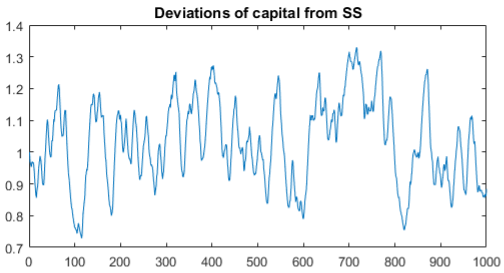
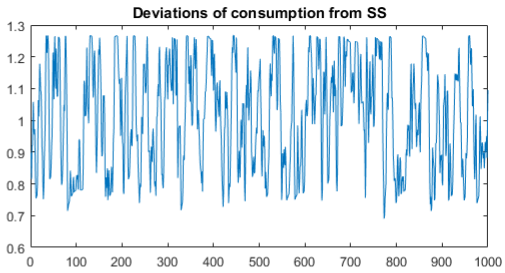
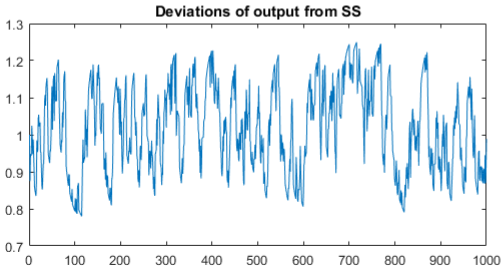


FIGURE 4. GPU's code simulations







CHAPTER 6

The Welfare Costs of Business Cycles

Most applications of the numerical methods introduced in Part I are going to be used in analyzing economic fluctuations for the most of Part II. A pertinent question relates to whether these fluctuations are meaningful and worth studying or being the concern of policy makers. In **?**, the author argues that the welfare gains from eliminating business cycles is trivial. In this section we will replicate his exercise for the U.S. economy.

1. Lucas' setup

Assume that agents are risk averse i.e. $u''(c) < 0$. Then, by construction:

$$u\left(\{c_t^d\}_0^T\right) > E\left[u\left(\{c_t^s\}_0^T\right)\right] \tag{1.1}$$

where $E\left[\{c_t^s\}_0^T\right] = \{c_t^d\}_0^T$.

The question Lucas poses is how much extra consumption would we need to give to an agent in an environment with stochastic consumption to make him indifferent between the stochastic and deterministic consumption schedules. This means, what is the value of λ such that:

$$u\left(\{c_t^d\}_0^T\right) = E\left[u\left(\{(1+\lambda)c_t^s\}_0^T\right)\right] \tag{1.2}$$

Lucas estimates the below model for stochastic consumption:

$$c_t^s = Ae^{\mu t}e^{-\frac{1}{2}\sigma^2}\varepsilon_t \tag{1.3}$$

Uses the following process for deterministic consumption:

$$c_t^d = Ae^{\mu t} \tag{1.4}$$

and constant relative risk aversion utility. The problem is then to find λ such that:

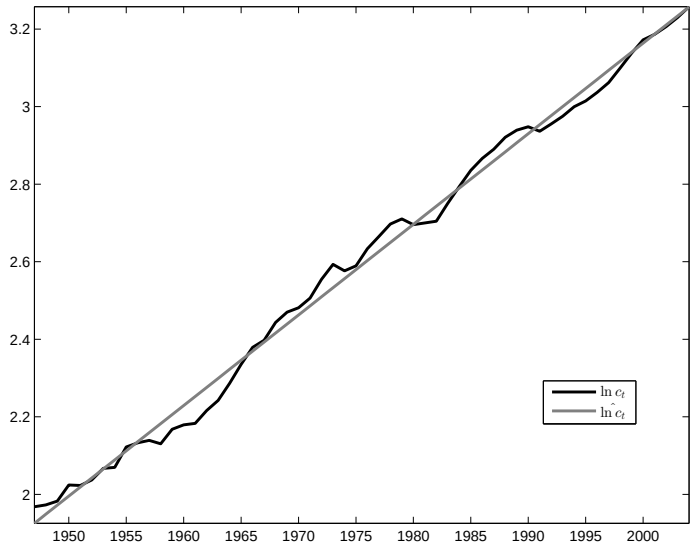
$$\sum_{t=0}^{\infty}\beta^t\frac{(Ae^{\mu t})^{1-\gamma}}{1-\gamma} = E_0\sum_{t=0}^{\infty}\beta^t\frac{\left[(1+\lambda)Ae^{\mu t}e^{-\frac{1}{2}\sigma^2}\varepsilon_t\right]^{1-\gamma}}{1-\gamma} \tag{1.5}$$

Assuming log-normality of the error term, standard values for γ and β and estimated values of μ, A and σ^2, Lucas estimates a value of $\lambda = 0.00023$, i.e., a mere 0.023% of consumption. Given the assumption of log-normality and the estimates of μ, A and σ^2, **?** solves the problem analytically. We will relax the assumption of log-normality (and use a different sample) which will make the problem computationally relevant and allow us to check the robustness of **?** findings.

2. Relaxing the log-normality assumption

Below we show the two schedules of consumption we are comparing, i.e., the observed stochastic consumption observed and its estimated certainty equivalent:

FIGURE 1. Observed and fitted $\ln c_t$



We estimate $A = 6.8584, \mu = 0.0234$ and $\sigma^2 = 0.0009$. We will now compute the optimal value of λ for $\gamma = \{0.5, 1, 1.5, 2, 2.5\}$. For that, follow the below procedure assuming $\beta = 0.96$:

(1) Regress $\ln c_t$ on a constant and a linear trend. Recover A, μ and σ^2 and save the residuals u_t.

(2) Simulate $\{c_t^s\}_0^T$ for $T = 3000$ using as shocks, random drawings with replacement from $\{u_t\}_1^{58}$ and equation 1.3.

(3) Compute the utility at $t = 0$ of this consumption schedule, for a grid of $\lambda = \{0, 0.00001, ..., 0.00025\}$ and each γ above using

$$\sum_{t=0}^{3000} \beta^t \frac{\left[(1+\lambda)Ae^{\mu t}e^{-\frac{1}{2}\sigma^2}\varepsilon_t\right]^{1-\gamma}}{1-\gamma} \tag{2.1}$$

(4) Repeat 2 and 3 1000 times and compute average utility for each choice of λ and γ to get:

$$E_0 \sum_{t=0}^{3000} \beta^t \frac{\left[(1+\lambda)Ae^{\mu t}e^{-\frac{1}{2}\sigma^2}\varepsilon_t\right]^{1-\gamma}}{1-\gamma} \tag{2.2}$$

(5) Compute, for each γ, the value of

$$\sum_{t=0}^{3000} \beta^t \frac{(Ae^{\mu t})^{1-\gamma}}{1-\gamma}$$

(6) For each value of γ, find the value of λ that make both consumption schedules have the same utility.

The results we find are given in Table 1 below. Notice that by relaxing the log-normality assumption, we get a λ about 4 times as high as **?**. However, this means that eliminating the business cycle would mean an improvement on lifetime consumption of just 0.08% which is still very small.

TABLE 1. Optimal λ for each γ

γ	0.5	1	1.5	2	2.5
λ*100	0	0.0247	0.0370	0.0617	0.0864

Below the code used to solve the problem:

```
clear all; load lucasdata.txt; %loads the data
n=size(lucasdata,1); t=(0:n-1)'; %creates the time vector
a=ones(n,1); %creates a vector of ones
logc=zeros(n,1); %creates the logconsumption vector
logc(1:n)=log(lucasdata(1:n,3)*1000./lucasdata(1:n,2));

% Now all the necessary vectors are created, we shall estimate our linear
% regression model.

% step1
X=[a t]; %the explanatory variables
[elogc,bhats,u]=linreg(logc,X); %the linear regression logc=A+mu*t+e
A=bhats(1); mu=bhats(2); sigma2=var(u); A=exp(A);
disp(['The estimated regression is log(c)= ' num2str(A) ' + ' num2str(mu) ' * t'])
disp(' ')
disp([' Hence, A= ' num2str(A) '; mu= ' num2str(mu) ' and sigma^2= ' num2str(sigma2)])

% step2
%
m=82;%number of lambdas
n=5;%number of gammas
dmatrix=zeros(m,n);
beta=(0:3000);
beta=(0.96.^beta); %creates the beta vector
cmatrix=zeros(m,n);
gamma=linspace(0.5,2.5,n);    %the parameters space for gamma
lambda=linspace(0,0.01,m); %and lambda
simcons=zeros(3001,1);    %Consumption and time are defined for t=0,T
t=(0:3000)'; %with T being 3000, yielding 3001 periods
for d=1:1000 %(step4 - the 1000 loop of steps 2 and 3)
    for i=1:3001
        simcons(i)=A*exp(mu*t(i))*exp(-0.5*sigma2)*exp(u(floor(1+57*rand)));
```

```
    end %the consumption series is simulated
%
%step3
%
%
    for i=1:m
        for j=1:n
            if j~=2 %different case for gamma=1 i.e. log utility
                cmatrix(i,j)=((1+lambda(i))^(1-gamma(j)))/...
                    (1-gamma(j))*(beta*(simcons.^(1-gamma(j))));
            end
        end
    end
    for i=1:m
        cmatrix(i,2)=beta*log(((1+lambda(i))*simcons));
    end
    dmatrix=dmatrix+cmatrix; %dmatrix is summing all 1000 simulations of
end                          %steps 1 and 2 and
%step 4
dmatrix=dmatrix./1000;   %with this command, it computes the mean

%step 5
certain=zeros(n,1);%the 5 values of consumption under certainty

for j=1:n
    if j~=2 %as before, we have to take the log util case into account
        certain(j,1)=((A^(1-gamma(j)))/(1-gamma(j)))*...
            (beta*exp(mu*t*(1-gamma(j))));
    end
end
certain(2,1)=beta*log(A*exp(mu*t));
%
%step 6
%
finalmatrix=-abs(dmatrix-[certain';certain';certain';certain';certain';...
    certain';certain';certain';certain';certain';certain';certain';...
    certain';certain';certain';certain';certain';certain';certain';...
    certain';certain';certain';certain';certain';certain';certain';...
    certain';certain';certain';certain';certain';certain';certain';...
    certain';certain';certain';certain';certain';certain';certain';...
    certain';certain';certain';certain';certain';certain';certain';...
    certain';certain';certain';certain';certain';certain';certain';...
    certain';certain';certain';certain';certain';certain';certain';...
    certain';certain';certain';certain';certain';certain';certain';]);

[maxout,nlambda]=max(finalmatrix,[],1);
```

```
answ=[-maxout' nlambda'];

disp(' ')
disp(' ')
disp(' Solution to step 6 - optimal lambda as a function of gamma:')
disp(' ')
for i=1:n
    disp(['for gamma= ' num2str(gamma(i))...
        ' the optimal lambda is ' num2str(lambda(answ(i,2)))]);
end
```

The output of the code above is:

```
The estimated regression is log(c)= 6.8584 + 0.023362 * t

 Hence, A= 6.8584; mu= 0.023362 and sigma^2= 0.00090222


 Solution to step 6 - optimal lambda as a function of gamma:

for gamma= 0.5 the optimal lambda is 0.00037037
for gamma= 1 the optimal lambda is 0.00061728
for gamma= 1.5 the optimal lambda is 0.00074074
for gamma= 2 the optimal lambda is 0.00098765
for gamma= 2.5 the optimal lambda is 0.0012346
```

CHAPTER 7

The Life Cycle Model

We start by analyzing the model brought forth by **?**. Assume an individual lives T periods. During these periods, the agent can either consume c_t or save a_{t+1} units of consumption for the next period. To finance these, the agent earns a deterministic income w_t and has an asset position of Ra_t where R is the gross rate of return on asset holdings a_t that were held from the previous period. The individual discounts one-period-ahead consumption by β and derives utility from consumption $u\left(\{c_t\}_1^T\right) = \sum_{t=1}^{T} \beta^{t-1} u(c_t)$ i.e. utility is time-separable. Assume, without loss of generality, that the individual is born with no assets, i.e., a_1=0 and that leaves no bequest, i.e., $a_{T+1} = 0$.

The individual maximizes his life-time utility by choosing consumption and saving schedules that solve:

$$\max_{\{c_t, a_{t+1}\}_1^T} \sum_{t=1}^{T} \beta^{t-1} u(c_t) \tag{0.1}$$

subject to:

$$c_t + a_{t+1} = Ra_t + w_t \tag{0.2}$$

$$a_{T+1} = 0 \tag{0.3}$$

$$a_1 = 0 \tag{0.4}$$

Assume constant relative risk aversion utility (CRRA) with risk aversion parameter σ. Setting up the lagrangian:

$$\mathcal{L} = \sum_{t=1}^{T} \beta^{t-1} \frac{c_t^{1-\sigma} - 1}{1-\sigma} + \lambda_t \left(Ra_t + w_t - c_t - a_{t+1}\right) \tag{0.5}$$

and taking first order conditions w.r.t. c_t, a_{t+1} and λ_t yields the following system of equations:

$$c_t : \quad \beta^{t-1} c_t^{-\sigma} = \lambda_t \tag{0.6}$$

$$a_{t+1} : \quad \lambda_t = \lambda_{t+1} R \tag{0.7}$$

$$\lambda_t : \quad Ra_t + w_t = c_t + a_{t+1} \tag{0.8}$$

Substituting the first equation in the second, yields:

$$\beta^{t-1} c_t^{-\sigma} = \beta^t R c_{t+1}^{-\sigma} \tag{0.9}$$

which, after some rearranging and simplifying leads to

$$c_{t+1} = (\beta R)^{\frac{1}{\sigma}} c_t \tag{0.10}$$

Lastly, just solve the budget constraint in 0.2 w.r.t. to consumption and substitute it in 0.10 to get

$$Ra_{t+1} + w_{t+1} - a_{t+2} = (\beta R)^{\frac{1}{\sigma}} (Ra_t + w_t - a_{t+1}),\ \{t\}_1^{T-1} \tag{0.11}$$

Together with $a_0 = 0$ and $a_{T+1} = 0$, equation 7 fully characterizes the solution. This is a non-autonomous second order difference equation in a_t. The solution implies finding the sequence $\{a_t\}_1^{t+1}$ such that $a_0 = 0$ and $a_{T+1} = 0$ solves .

First notice that we have all we need to solve the problem. The solution requires two initial conditions, which we have, given our assumptions upon a_1 and a_{T+1}. Another way to see this, is to notice that (7) consists of $T-1$ equations in $T+1$ unknowns. Hence the need for the two intial conditions. The system of equations is shown below:

$$a_1 = 0 \tag{0.12}$$

$$Ra_2 + w_2 - a_3 = (\beta R)^{\frac{1}{\sigma}} (Ra_1 + w_1 - a_2) \tag{0.13}$$

$$\vdots$$

$$Ra_T + w_T - a_{T+1} = (\beta R)^{\frac{1}{\sigma}} (Ra_{T-1} + w_{T-1} - a_T) \tag{0.14}$$

$$a_{T+1} = 0 \tag{0.15}$$

We will now explore different solution methods to solve the next exercise.

EXERCISE 12. *Solve equation (7) using the following parameters:* $T = 90$, $\beta = 0.96$, $\sigma = 2$, $R = 1.03$. *Use the methods described in the following sections and compare the execution time. Plot assets, wages and consumption against age. Use the interpolated wage data in* `wages.m` .

1. The Life-cycle model as a $n-$dimensional system of equations

Given the model parameters and a sequence $\{w_t\}_1^{T+1}$, we can use many different methods. The most immediate and straightforward way is to treat this as a system of $T+1$ equations in $T+1$ unknowns in a_t. We can use Newton-Rhapson's root finding algorithm provided by MatLab in `fsolve.m`. Solving equations is a root-finding problem. We therefore need to cast the system (0.12)-(0.15) in the form $f(\mathrm{x}) = 0$. We then create an auxiliary function file, `focs1.m` that for given model parameters and a wage vector defined in the base workspace, will yield the residuals for the system of equations.

```
function [res]=focs1(a)

% focs1 is an auxiliary function. It takes as input the guessed
% assets  and returns the  associated euler equation residuals

% These variables are called from the base environment
global T beta R w sigma % set in the main file
```

```
c=zeros(T,1); % creates the consumption vector

% Backout consumption from the budget constraint
c(1:T)=R*a(1:T)+w(1:T)-a(2:T+1);

ac = [a(1:end-1) c];
assignin('base', 'sol1', ac); % exports a and c to base environment

% The euler residuals
euler(1:T-1,1)=c(2:T)-(beta*R)^(1/sigma)*c(1:T-1);

% all the residuals
res = [a(1);euler;a(T+1)];
```

An alternative solution method would have been to use linear algebra, since we are solving a system of linear equations. The system in(0.12)-(0.15) can be written in matrix form as $\mathbf{Ca} = \mathbf{w}$ where $\mathbf{C}$ is the matrix of coefficients:

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ -R(\beta R)^{\frac{1}{\sigma}} & R+(\beta R)^{\frac{1}{\sigma}} & -1 & 0 & 0 & \dots & 0 \\ 0 & -R(\beta R)^{\frac{1}{\sigma}} & R+(\beta R)^{\frac{1}{\sigma}} & -1 & 0 & \dots & 0 \\ 0 & 0 & -R(\beta R)^{\frac{1}{\sigma}} & R+(\beta R)^{\frac{1}{\sigma}} & -1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 1 \end{bmatrix} \tag{1.1}$$

$\mathbf{a}$ is the vector of assets

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ \vdots \\ a_{T+1} \end{bmatrix} \tag{1.2}$$

and $\mathbf{w}$ the vector with the wages

$$\mathbf{w} = \begin{bmatrix} 0 \\ (\beta R)^{1/\sigma} w_1 - w_2 \\ (\beta R)^{1/\sigma} w_2 - w_3 \\ (\beta R)^{1/\sigma} w_3 - w_4 \\ \vdots \\ 0 \end{bmatrix} \tag{1.3}$$

The solution is then $\mathbf{a} = \mathbf{C^{-1}w}$.

In terms of programming, the only difficulty is to generate a matrix with the coefficients above. Below the code that implements this method:

```
function a = Caw()

% CaW is an auxiliary function. It takes no inputs as it makes use of
% globally set variables in the base environment
```

```
global T beta R sigma w

% convert the wage process to the w vector in the notes
 ws = [0;w(1:T-1)*(beta*R)^(1/sigma)-w(2:T);0];

 %initiates the coefficients matrix
 x = zeros(T+1,T+1)*NaN;
  for i= 1:T+1
     for j = 1:T+1
         if i==j
             x(i,j)=-R*(beta*R)^(1/sigma);
         elseif i==j-1
             x(i,j)=R+(beta*R)^(1/sigma);
         elseif i==j-2
             x(i,j)= -1;
         else
             x(i,j)=0;
         end
     end
  end

% adds the first and last rows and converts C to a sparse matrix
C = sparse([1 zeros(1,T);x(1:T-1,:);zeros(1,T) 1]);

% solves for a
a=C\ws;
c = zeros(T,1)*NaN;
%backs out consumption path implied by assets schedule
for i  = 2:T
        c(1) = w(1)-a(2) ;
        c(i) = (beta*R)^(1/sigma)*c(i-1);
end
sol2 = [a(1:T,1) c];
assignin('base', 'sol2', sol2); % exports c to base environment
```

Notice how in `Caw.m` we make use of the `sparse.m` command. Since matrix **C** is indeed sparse i.e. most elements are zero, it is computationally efficient to tell MatLab to store in memory only the elements that different from zero.

2. The Life cycle Model as a single equation problem

Both methods above miss one key element in the structure of the problem. In reality, one can look at the second order difference equation as single equation problem. Assume that $a_1 = 0$. Then, guess a_2 and compute a_3. Iterate forward and check which value of a_{T+1} is implied by our initial assumption that $a_1 = 0$ and our guess that $a_2 = 0$. If we find that $a_{T+1} > 0$, it means that our initial guess of a_2 was too high. Conversely, if $a_{T+1} < 0$ it means that our guess was too low. This shows that there exists a mapping $f : f(a_2) = a_{T+1}$ and that solving the second order difference equation boils down to finding the root of $g(a_2) = f(a_2) - a_{T+1}$. This is typically described as solving the difference equation by forward iteration.

The code below creates a function that implements the forward iteration algorithm.

```
function atp1  = forwi(a2)

% forwi is an auxiliary function. It takes no inputs as it makes use of
% globally set variables in the base environment

global T beta R sigma w

c = zeros(T,1);
a = zeros(T+1,1);

for i  = 2:T
        a(2) = a2;c(1) = w(1)-a(2) ;
        c(i) = (beta*R)^(1/sigma)*c(i-1);
        a(i+1) = R*a(i)+w(i)-c(i);
end
ac = [a(1:end-1) c];
assignin('base', 'sol3', ac); % exports to base environment a and c
atp1 = a(T);
```

Following the reasoning from the forward iteration, one could have solve the problem backwards instead. Assume that $a_{T+1} = 0$. Guess a_T and solve for a_{T-1}. Iterate backwards to get the value of a_1. If $a_1 = 0$, then a_T is the solution and one can backout the entire path for assets. This is called solving the problem by backward iteration.

3. Features of the Life Cycle model

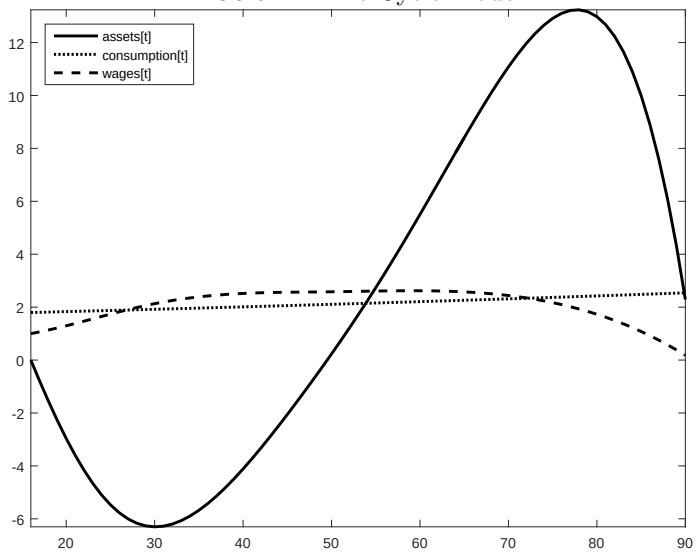
Below sample code that solves the model through the three methods, shows computation time for each and produces a plot with the solution.

```
clear all;close all; clc;
global T beta R w sigma

% Parameters%%%
beta  = 0.98; R  = 1.03; sigma = 2; T = 75;
load wages.mat; w = wages;  %gets wages and age vectors

% use focs1.m
tic; [a] = fsolve(@focs1,zeros(T+1,1)); %solves the focs
a = a(1:end-1); toc %remove the a_{T+1}

% use Caw.m and remove a_{T+1}
tic; a=Caw; a=a(1:end-1); toc

% use forwi.m
tic; [a1]=fsolve(@forwi,-1);toc%solves the focs
```

```
plot(age,[sol3 w]);axis tight;
legend('(assets[t]','consumption[t]','Location','Northwest');
```

The time the three methods took to run was of 0.198659, 0.005602 and 0.056684 seconds respectively. In such a simple model, all times are quite small. However, in more complex problems it may matter. Notice that the time it took to solve the problem using matrix algebra was one order of magnitude lower than the forward iteration algorithm. As it is expected, solving a system of $T+1$ equations is much more computationaly intensive than solving a one equation problem or invert a matrix. Below the plot with the asset, consumption and wage schedules.

FIGURE 1. Life Cycle Model



With regard to Figure 1 notice that, with complete markets, the agent is fully capable of using the asset market to smooth consumption over the entire life cycle. The timing of the wage schedule does not matter, only its net present value. To see this, we generate 4 different wage paths with the same net present value and see that they imply different assets' schedules but consumption does not change.

The first panel just repeats Figure 1 for comparison. In the second panel, a white noise shock is added to the original wage schedule. Notice that this is still a perfect foresight model. The shock was added to illustrate how the assets schedule is used to smooth the variability in the wage. Agents still know the complete wage profile since the first period.

In the third panel one can see that for a concave wage profile, the optimal asset holdings mirror the shape we had observed before. In the fourth panel, the wage is set to a constant equal to the net present value of the wage schedule in panel one, but with an added white noise shock.

If there is a large and unexpected increase in income for one period, consumption will not rise accordingly because the agent will spread the extra income over the rest of his lifetime. This gives raise to the permanent income hypothesis by

?. Remember that in the traditional Keynesian framework, consumption is a fixed share of disposible income and an unexpected increase in income for one period is expected to affect consumption proportionally..

FIGURE 2. Timing of Wages does not matter

4. Extension: Concave consumption

In this section we will present a simple extension to the model. Observed data for consumption show that consumption has a hump-shaped schedule over the life cycle. By presenting this simple extension, we will make the model closer to the data and at the same time provide better intuition for the consumption smoothing mechanism implied by the Euler equation.

The problem will be the same as before, but now we will make households more patient as time goes by, by making the subjective discount factor β age dependent. We will use forward iteration and bisection to solve the model.

EXERCISE 13. *Use the same environment as before, but now assume that* $\beta(t) = 80 - t$ *and that* $\sigma = 1$. *Plot consumption and the process for beta in the same plot. How is the relation between* β *and* R *and consumption growth?*

Since β now changes with age, we will have a different version of the Euler equation:

$$c_t^{-\sigma} = \frac{\beta_{t+1}}{\beta_t} R c_{t+1}^{-\sigma} \tag{4.1}$$

Below the code that solves the model:

```
% guess an interval for assets in the period before last
alb = -100; aub = 100;

% set max iterations, iterations counter and precision
itermax = 100; iter = 0; epsilon = 10^-10; sigma = 1;

betai    = (80:-1:1)'; % set process for discount factor

while iter<itermax
    c      = zeros(T,1);
    a      = zeros(T,1);
    a(1)   = (alb+aub)/2;
    c(1)   = w(1)-a(1);
    for i=2:T
        c(i) = (betai(i)/betai(i-1)*R)^(1/sigma)*c(i-1);
        a(i) = R*a(i-1)+wages(i)-c(i);
    end

    % check if a(1)=0

    res = abs(a(T)-0);
    if res < epsilon;
        break
    end
    if a(T) > 0
        aub = a(1);
    else
        alb = a(1);
    end
    iter = iter+1;
end

c = c';
```

And as before, we plot assets, wages and consumption. The time profile of β_t induces a concave consumption schedule as it can be observed in Figure 3. Note that since β_t is decreasing in t, as time increases consumption becomes less and less valuable. At the end of the life cycle consumption is almost close to zero, despite the instant marginal utility being very high.

Next we analyze the shape of consumption over the life cycle and relate it to the interaction between β_t and R. In Figure 4 we plot the consumption schedule and the growth rate of consumption $\frac{c_{t+1}}{c_t}$, which is given by $R\frac{\beta_{t+1}}{\beta_t}$. In this case, any process for β_t such that $R\frac{\beta_{t+1}}{\beta_t} \geq 1$ for $[1, t_0]$ and $R\frac{\beta_{t+1}}{\beta_t} \leq 1$ for $[t_0, T]$ will result in a concave consumption shedule with a maximum at t_0.

FIGURE 3. Life Cycle Model - Concave Consumption

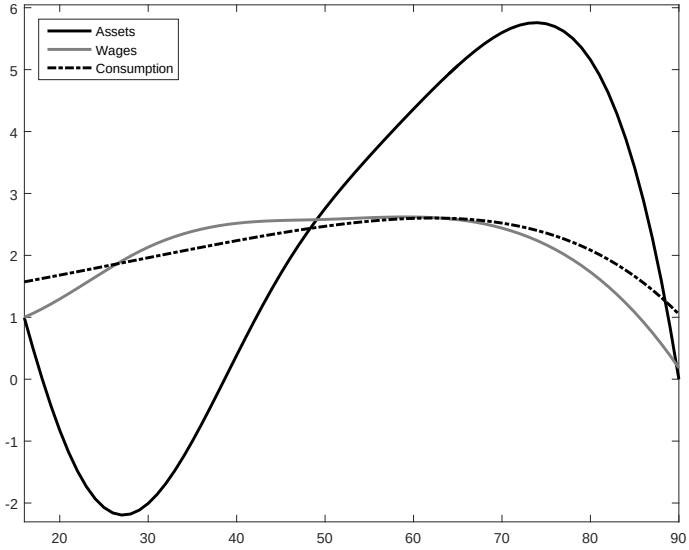
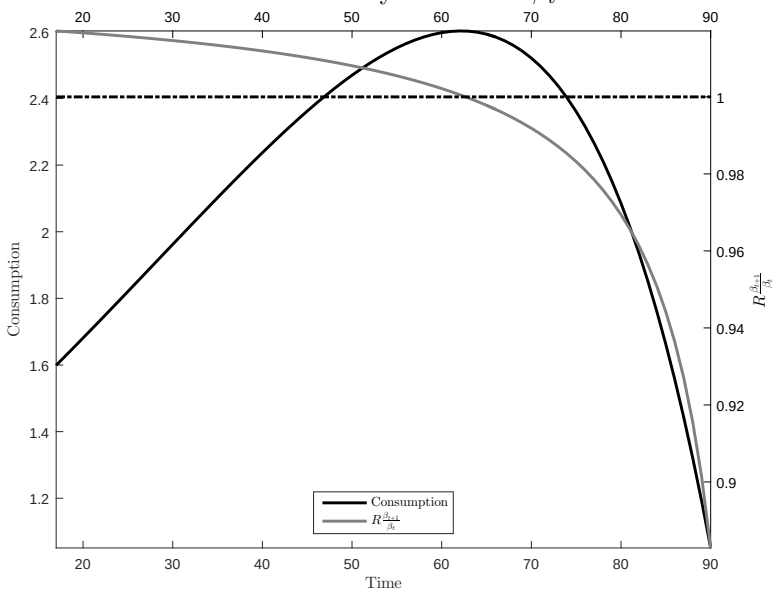


FIGURE 4. Life Cycle Model - β_t vs R



5. Extension: Idiosyncratic productivity

In this section we will see how to introduce idiosyncratic productivity in a life cycle model. In the data we observe that there exists income inequality for agents of the same age cohort. So, not only age generates income inequality, but also other agents characteristics can induce this inequality. To generate this inequality in the model we introduce an idiosyncratic productivity shock.

EXERCISE 14. *Using the same environment as before, introduce idiosyncratic risk. The idiosyncratic risk takes the form of a persistancy productivity shock. Simulate the economy and plot the saving decision of the agents. How does this changes the saving decisions of the agents?*

The idiosyncratic shock considered here will be persistent and will follow a markov chain. We assume there are three stages of productivity: low, medium and high. In the initial period the agent will withdraw a given productivity $z \in [z_l, z_m, z_h]$. In the next period the probability of the agent withdrawing a given productivity level is given by $Pr(z_{t+1} = z_j | z_t = z_i) \equiv \pi_{ij} \geq 0$ with π_{ij} being the probability of going from productivity i today to productivity j tomorrow with $\sum_{j=1}^{N} \pi_{ij} = 1$, where N is the number of possible productivity states. To calculate the productivity levels and the transition matrix we use the `tauchen.m`matlab function.

We have two ways to solve this problem: either we find the zero for the euler equation using MatLab's implementation of the Newton-Rhapson algorithm introduced earlier, the function `fsolve.m`, or we maximize the value function using the Golden Search algorithm introduced previously.

The euler equation of this problem will now become

$$c_t^{-\sigma} = \frac{\beta_{t+1}}{\beta_t} R E[c_{t+1}^{-\sigma}]$$

with the expected value of consumption in the next period being given by

$$E[c_{t+1}^{-\sigma}] = \sum_{j=1}^{N} \pi_{ij} c_{j,t+1}^{-\sigma}$$

with i being the state of productivity today and j the state of productivity tomorrow. To solve the problem using the euler equation we need to find the policy function that for each asset level and each productivity state will give us the saving decision for the next period that solves the euler equation.

In the last period (T) we know that the optimal saving decision is zero, so the agents consumes everything. Given that we now have the policy function for the last period we can find the policy function for T-1, that for each given asset level and each productivity state will give us the optimal saving decision. Given that there is no guarantee that my optimal decision of saving is in the assets' grid we need to interpolate the policy function in the next period so that we can evaluate it at the optimal saving decision. And we repeat this for every period so that we get the policy functions for every period.

Then using this policy functions we simulate the model for 10000 agents. Using the `markov.m`matlab function we simulate the productivity path for each agent in the economy. Then using the policy functions, and assuming everyone is born with zero assets, we see the saving decision for each agents.

The code to solve this problem is the following

```
simul = 10000; %number of simulations

%discretizing idiosyncratic shock

nz    = 3; %number of states
```

```
a     = 0;
rho   = 0.9; %persistency of the shock
sigma = 0.1; %standard deviation of the shocks
m     = 1;

%tauchen package will generate a matriz z with nz states of productivity and zprob transition
[z,zprob] = tauchen(nz,a,rho,sigma,m);
z=exp(z+1);

% Policy function in the last period

g=zeros(size(a,2),size(z,2),T);


for t=T-1:-1:1
for i=1:size(a,2)
for j=1:size(z,2)
g(i,j,t)=fsolve(@(ap1) w*prod(t)*z(j)+R*a(i)-ap1-beta*R*...
zprob(j,:)*(w*prod(t+1)*z'+R*ap1-interp1(a,g(:,:,t+1),ap1)'),0.1);
end
end
end

chain  = ones(simul,T);
assets = zeros(simul,T);

for i=1:simul
[chain(i,:)]=markov(zprob,T+1,z(2),z);
for t=2:T
assets(i,t)=interp1(a,g(:,find(z==chain(i,t)),t),assets(i,t-1));
end
end
```

If we instead use value function we can either solve the problem using interpolation techniques as in the euler equation solution or we can solve in the gpu using a grid for assets with a lot of points and find the maximum for the value function in that grid.[1]

Let us start with the interpolation techniques. As we are not finding a zero for the function, as in the euler equation problem, we will now use the Golden Search technique introduced beforeto find the maximum of the value function. The value function of the problem will be given by

$$V(a, z, t) = U(c) + \beta E[V(a', z', t + 1)]$$

where a is the asset level at the beginning of the period, z is the productivity state and t is the age. The expected value function in the next period is nothing more than

[1]Note: In the euler function we can also solve the problem with interpolation techniques and in the gpu using a large grid of asset points.

$$E[V(a', z', t+1)] = \sum_{j=1}^{N} \pi_{ij} V(a', z_j, t+1)$$

Once again we will solve the problem by backward induction. In the last period of life the value function will be equal to the utility function and the policy function will be equal to zero given that the agent is going to consume everything. Then using the Golden search and the value function in period T we will find the consumption that maximizes the value funtion in T-1 for a given level of assets a. Once again we use interpolation techniques so that we can evaluate the value function in t+1 at the saving decision today so that we can find the maximum for the value function today. The policy function will be equal to nothing more than the saving decision associated with the consumption that maximizes the value funtion.

In the end we simulate the economy in the same way we did before and plot the assets of each agents along their life. The code that solves the problem is bellow.

```
% Policy function

g=zeros(size(a,2),size(z,2),T);

Last period value function

for ia=1:size(a,2)
c=w.*prod(T).*z(:)+a(ia);
if c>0
V(ia,:,T)=(c.^(1-gamma))./(1-gamma);
else
V(ia,:,T)=-10;
end
end

%Value function from 1 until T-1 using golden search

for t=T-1:-1:1
for ia=1:size(a,2)
for iz=1:size(z,2)
%Initial values for the golden search algorithm
c1=0.01;
c2=max(a(ia)-0.01,w*prod(t)*z(iz)+R*a(ia));
cl=c1+((3-5^0.5)/2)*(c2-c1);
ch=c1+((5^0.5-1)/2)*(c2-c1);

if c2<0 %just to guarantee we do no go to a path where we end with negative consumption
V(ia,iz,t)=-100;
else
while abs(c1-c2)>tolc
%assets associated with each consumption point so that we can evaluate the value function in
apl = -cl+w*prod(t)*z(iz)+R*a(ia);
aph = -ch+w*prod(t)*z(iz)+R*a(ia);
```

```
if apl<min(a)%just to guarantee we do no go to a path where we end with assets lower than the
Vl=-100;
else
Vl=(cl.^(1-gamma))./(1-gamma)+...
beta*zprob(iz,:)*interp1(a,V(:,:,t+1),apl,'spline')';
end

if aph<min(a)%just to guarantee we do no go to a path where we end with assets lower than the
Vh=-100;
else
Vh=(ch.^(1-gamma))./(1-gamma)+...
beta*zprob(iz,:)*interp1(a,V(:,:,t+1),aph,'spline')';
end

if Vl>Vh
c2=ch;
ch=c1+(5^0.5-1)/2*(c2-c1);
cl=c1+(3-5^0.5)/2*(c2-c1);
else
c1=cl;
ch=c1+(5^0.5-1)/2*(c2-c1);
cl=c1+(3-5^0.5)/2*(c2-c1);
end
end

V(ia,iz,t)=Vl;
g(ia,iz,t)=apl;
end
end
end
end



chain  = ones(simul,T);
assets = zeros(simul,T);


for i=1:simul
[chain(i,:)]=markov(zprob,T+1,z(2),z);
for t=2:T-1
assets(i,t)=interp1(a,g(:,find(z==chain(i,t)),t),assets(i,t-1));
end
end
```

The last way that we present here to solve this problem is by using an asset grid with many points and evaluating the value function in each of this points. This

the way to program the model to run in the gpu given the problem is very simple, is just repeated many times.

To solve the problem in the gpu we create an assets grid with 700 points. We start once again by solving the problem in the last period of life knowing that the saving will be zero and so the value function will be just equal to the utility. We solve it for each point in the asset grid.

Then with the last period value function, we solve for period T-1. For all the points in the asset grid we find the saving decision that maximizes the value function by evaluating the value function at each possible saving decision. The policy function will then be the optimal saving decision for each initial asset holding.

Once again we simulate the model and find the optimal asset path for each agent. The code to solve this problem is the following

```
%discretize assets

a = [-2:0.01:5];


V=zeros(size(a,2),size(z,2),T,'gpuarray');

%Value function in the last period

for ia=1:size(a,2)
c=w.*prod(T).*z(:)+a(ia);
for ic=1:size(c)
if c(ic)>0
V(ia,ic,T)=(c(ic).^(1-gamma))./(1-gamma);
else
V(ia,ic,T)=-100;
end
end
end

g=zeros(size(a,2),size(z,2),T);

%Value function problem

for t=T-1:-1:1
for ia=1:size(a,2)
for iz=1:size(z,2)
c = w*prod(t)*z(iz)+R*a(ia)-a;
for ic=1:size(c,2)
if c(ic)>0
v(ic) = (c(ic).^(1-gamma))./(1-gamma)...
+beta*zprob(iz,:)*V(ic,:,t+1)';
else
v(ic)=-100;
end
```

```
end
[V(ia,iz,t),index] = max(v);
g(ia,iz,t) = a(index);
end
end
end

chain  = ones(simul,T);
assets = zeros(simul,T);


for i=1:simul
[chain(i,:)]=markov(zprob,T+1,z(2),z);
for t=2:T-1
assets(i,t)=interp1(a,g(:,find(z==chain(i,t)),t),assets(i,t-1));
end
end

t=1:1:65;
plot(t,assets(:,:))
```

CHAPTER 8

The Overlapping Generations Model

In this section, we introduce the overlapping generations model, in the spirit of **?**. We assume the existence of finitely lived different cohorts of agents that coexist simultaneously. As it turns out, given some simplifying assumptions, this model is mathematically remarkably similar to the life cycle model. The only difference is that the agents now live in an environment where the interest rate on savings is determined by supply and demand i.e. general equillibrium. For that, we will have an aggregate production function and markets for capital that will set factor prices i.e. interest and wage rates. Note that with no population growth or uncertainty, aggregate variables and prices are constant and the economy is always at the steady state.

1. General setup

A continuum of agents is born each period and lives for N periods. Each agent chooses optimal consumption and saving schedules over their life time, given preferences and their budget constraint. For now we will assume that agents always supply 1 unit of labor every period. We will relax this assumption later. Each period a generation dies and another takes its place such that at any given time the economy consists of N generations. Lifetime discounted utility for and agent agent aged i at period t is given by:

$$\sum_{i=0}^{N-1} \beta^i u(c_{t+i,i}) \tag{1.1}$$

The budget constraint for an agent born i periods ago is given by:

$$c_{t+i,i} + k_{t+i+1,i+1} = (1 + r_{t+i})k_{t+1,i} + \epsilon_i w_{t+i} \tag{1.2}$$

where r_t is the rental price of capital, ϵ_i age dependent labor productivity and w_t the wage rate. We do not allow agents to die with debt, so we impose that

$$k_{t,N} \geq 0 \tag{1.3}$$

Total output is given by

$$F(K_t, H_t) = K_t^{\alpha} H_t^{1-\alpha} \tag{1.4}$$

where

$$K_t = \sum_{i=1}^{N} k_{i,t} \tag{1.5}$$

$$H_t = \sum_{i=1}^{N} h_{i,t} \tag{1.6}$$

The problem for an agent born at period t is then to:

$$\max_{c_{t+i,i},h_{t+i,i},k_{t+i,i+1}} \sum_{i=0}^{N-1} \beta^i u(c_{t+i,i}, l_{t+i,i}) - \lambda_i \left[c_{t+i,i} + k_{t+i+1,i+1} - (1+r_{t+i})k_{t+i,i} - \epsilon_i w_{t+i} h_{t+i,i}\right] \tag{1.7}$$

Assuming an interior solution, the first order conditions are then:

$$\beta^i u_{c_{t+i,i}} = 0 \tag{1.8}$$

$$u_{h_{t+i,i}} + \lambda_{t+i}\epsilon_i w_{t+i,i} = 0 \tag{1.9}$$

$$-\lambda_{t+i} + \lambda_{t+i+1}(1 + r_{t+i+1}) = 0 \tag{1.10}$$

Substituting away the multiplier, we get

$$u_{h_{t+i,i}} + u_{c_{t+i,i}}\epsilon_i w_{t+i,i} = 0 \tag{1.11}$$

$$-u_{c_{t+i,i}} + \beta u_{c_{t+i+1,i+1}}(1 + r_{t+i,i+1}) = 0 \tag{1.12}$$

There is a representative firm that pays factor prices their marginal product:

$$r_t = F_K(K, H) \tag{1.13}$$

$$w_t = F_H(K, H) \tag{1.14}$$

The aggregate supply of savings is equal to firms' demand for capital and labor supply equals labor demand

$$K_t = \sum_{i=0}^{N-1} k_{t,i} \tag{1.15}$$

$$H_t = \sum_{i=0}^{N-1} \epsilon_i h_{t,i} \tag{1.16}$$

Capital depreciates at rate δ and therefore the gross interest rate is given by $R = 1 - \delta + F_K(K, H)$. The solution also satisfies the lifetime bugdet constraint and aggregate resource contraint:

$$\sum_{i=0}^{N} \left(\frac{1}{R}\right)^i \left[c_{t+i,i} - \epsilon w_t h_{t+i,i}\right] = 0 \tag{1.17}$$

$$\sum_{i=0}^{N-1} c_{t,i} + \delta k = 0 \tag{1.18}$$

In the next sections we will be solving different versions of this problem. We will assume that utility is given by:

$$u(c_t, h_t) = \frac{c_t^{1-\gamma}}{1-\gamma} + \phi \frac{l_t^{1-\mu}}{1-\mu} \tag{1.19}$$

Productivity over the life cycle is given by $\epsilon_i = -0.5350819 + 0.069134i - 0.000713i^2$, $\alpha = 0.3$, $\beta = 0.96$, $\delta = 0.08$. $\mu = 2$, $\gamma = 1$ and $\phi = 1$.

2. Exogenous labor supply

We start by solving the above problem by assuming that agents always supply one unit of labor. By doing so, we ignore the first order conditions with respect to the labor-leisure choice. This setup, though formally different, is mathematically very similar to the life cycle model from the preceeding section. The algorithm to solve it is:

(1) Guess aggregate capital

(2) Compute aggregate labor

(3) Compute $r = \frac{\partial Y}{K}$.

(4) Compute $w = \frac{\partial Y}{H}$.

(5) Given r, w solve the agents' (life cycle) problem.

(6) Compute aggregate assets A.

(7) If $A > K$ decrease r. If not, increase it.

(8) Repeat 1-7 until $A = K$.

As we see, the solution method for this problem is just to solve the life cycle model introduced before, with successive guesses for the capital stock until the demand and supply for assets equate.

Let's solve the following exercise then:

EXERCISE 15. *Assume agents are born into the economy at 20 years old and die at 84. Find the steady state capital stock and associated interest rate. Plot an agent's schedule for savings, consumption and assets. Check also that both the aggregate resource contraint and lifetime budget constraint are satisfied.*

Below the code that solves Exercise 15.

```
clear all;close all;clc;
global T beta R Fh prod% i'll need this variables when calling focs.m

%%%Parameters%%%
alpha=.3;beta=.96;delta=.08;
born=20; %when the agent is born
T=65; % number of periods the agent lives
dies=84; % agent dies
```

```
tol=1e-2; % convergence criteria
diffk=10; % initial value for the convergence test variable

%%%% Productivity vector %%%% - No retirement age in this model.
prod(1:T,1)=-0.5350819+0.0691343*(born:dies)-0.0007131*(born:dies).^2;

%guess aggregates
H=sum(prod);
K0=H; %I'm guessing that K is somewhere between H and 10*H
K1=10*H;
K=(K0+K1)/2;

%The Guess
guess=zeros(T-1,1); %i "guess" asset holdings per period
iter=0; %sets the initial value for the iteration parameter
itermax=30; %sets the maximum value of iterations to take before stopping

while (iter<=itermax && diffk>tol)
    iter=iter+1; %updates the number of iterations done

    % compute prices
    Fk=alpha*((K/H)^(alpha-1)); %marginal productivity of capital
    Fh=(1-alpha)*((K/H)^alpha); %marginal productivity of labor
    R=1-delta+Fk; %gross interest rate on capital

     %solve first order conditions%
     [assets]=fsolve(@focs1,guess); %solves the focs, starting with the guess
     A=sum(assets);

     %update K
     if A > K  % this ensures that markets clear - bissection method
         K0=K;
     else
         K1=K;
     end
     K=(K0+K1)/2;

     diffk = abs(K-A);disp('diffk');disp(K-A);
     guess=assets; % updates the guess
end

wts = prod*Fh; %wages
%plots & Capital stock and interest rate
figure('Name','Consumption, assets, steady state capital and interest rate')

[euler consumption]=focs1(assets);
assets=[assets;0];
```

```
% Check the lifetime budget constraint and the aggregate resource
% constraint

r=((1/R).*ones(65,1)).^((0:64)');
lfbc=r'*(consumption'-Fh*prod);
```

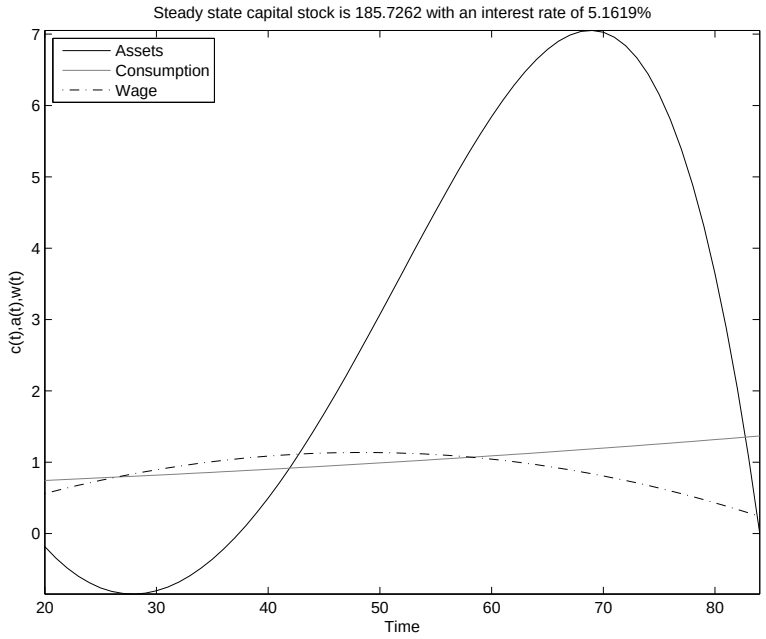
Notice that this script calls the `focs1.m`script, which containts the first order conditions:

```
function [euler,consumption]=focs1(x)

% Euler is an auxiliary function for olg1.m . It takes as input the guessed
% assets and returns the euler equation associated with that choice of
% assets.

global T beta R Fh prod % these are the variables declared global by ps6.m

a=x;
c=zeros(T,1); % creates the consumption vector

% consumption in the first period is just
% the wage*productivity-savings
c(1)=Fh*prod(1)-a(1);

% during the life cycle, consumption = R*(assets holding at t)-assets
% bought at t+wages*productivity
c(2:T-1)=R*a(1:T-2)+Fh.*prod(2:T-1)-a(2:T-1);

% In the last period, the agents don't save. Hence, consumption is:
c(T)=R*a(T-1)+Fh*prod(T);

% The output  is then the euler equation (note log util-> u'=1/c):
euler(1:T-1,1)=-c(1:T-1).^(-1)+(beta*R).*c(2:T).^(-1);

% And the time path for consumption
consumption(1:65)=[c(1);c(2:T-1);c(T)];
```

As we can see in Figure 1, the life cycle paths for each generation of agents does not differ at all of the behavior we observe for the partial equillibrium version of the model. As mentioned before, to make this general equillibrium, the only thing needed is to make the interest rate endogeneous. To do that, we added an aggregate production function and computed equilibrium factor prices.

For this setup, the aggregate capital stock is $K = 185.73$, the gross interest rate $R = 5.16\%$ and both the aggregate resource and lifetime budget contraints are satisfied.

FIGURE 1. Life cycle paths



3. Borrowing constraints

In this section we are going to implement the Karush-Kuhn-Tucker method introduced earlier to simulate the overlapping generations model when agents are not allowed to have negative assets over the life cycle. This translates into adding a non-negativity constraint to savings to the set of first order conditions:

$$\mu_{t+i} k_{t+i+1,i+1} = 0 \tag{3.1}$$

and also adding the corresponding multiplier to the Euler equation, since it needs no longer to hold with equality:

$$-u_{c_{t+i,i}} + \beta u_{c_{t+i+1,i+1}} (1 + r_{t+i,i+1}) + \mu_{t+i} = 0 \tag{3.2}$$

Notice that the general structure of the problem remains essentially the same. The borrowing constraints only affect the optimality of the savings decision for the agent. Hence, we need only to modify the auxiliary file which contains the first order conditions for the problem. Now we will have $N-1$ new equations, one per period for the $mu_{t+i} k_{t+i+1,i+1} = 0$ condition and the Euler equations will have an extra variable, precisely mu_{t+i} to account for the situations when the borrowing constraint binds and agents are not able to smooth consumption. The method is just a generalization to $N-1$ equations of the method introduced in the Optimization chapter.

```
function [euler,consumption]=focs2(x)
```

```
% focs2 is an auxiliary function for olg2.m . It takes as input the guessed
% assets and returns the euler equation associated with that choice of
% assets.

global T beta R Fh prod l mu% these are variables declared global by olg2

%T=65;
a=x(1:T-1);
mu=x(T:2*T-2);
c=zeros(T,1); % creates the consumption vector

% consumption in the first period is just the wage*productivity-savings
c(1)=Fh*prod(1)-a(1);

% during the life cycle, consumption = R*(assets holding at t)-assets
% bought at t+wages*productivity
c(2:T-1)=R*a(1:T-2)+Fh.*prod(2:T-1)-a(2:T-1);

% In the last period, the agents don't save. Hence, consumption is:
c(T)=R*a(T-1)+Fh*prod(T);

% The output  is then the euler equation (note log util-> u'=1/c):
euler(1:T-1,1)=-c(1:T-1).^(-1)+(beta*R).*c(2:T).^(-1)+max(0,mu(1:T-1)).^l;

% we now add a second set of equations to make k>=0
euler(T:2*T-2,1)=a(1:T-1)-max(0,-mu(1:T-1)).^l;

% And the time path for consumption
consumption(1:65)=[c(1);c(2:T-1);c(T)];
```

The code to make use of the auxiliary function above is exactly the same as previously, the only difference is that now we need to provide also an initial guess for the set of multipliers. Below in Figure 2, the life cycle schedules of consumption and assets. Notice how consumption has a kink when the borrowing constraint ceases to bind, while until then assets are zero. Notice also that the steady state capital stock is higher (due to the increase in savings) and the interest rate is lower, when both are compared with an environment without borrowing constraints.

4. Endogenous labor supply

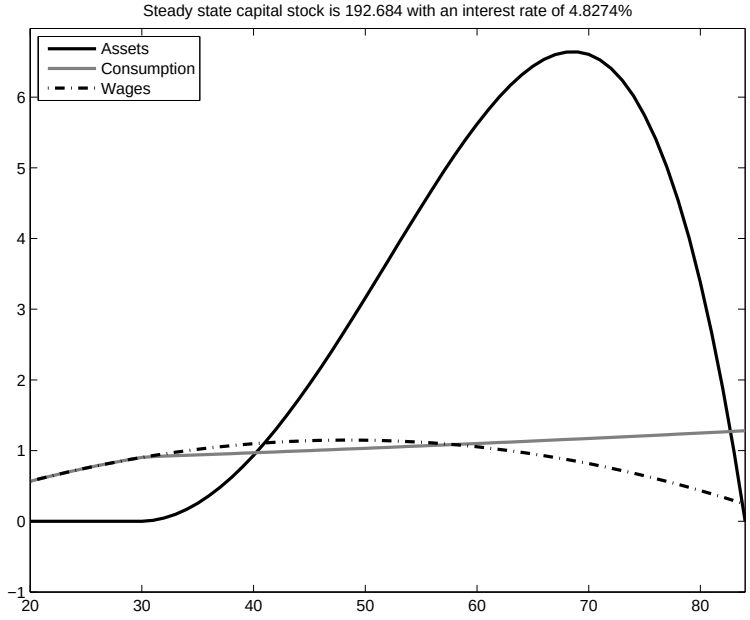
TBA

5. Extension: Social security system

An interesting extension for the OLG model is introducing a social security system. In reality people work until a certain age and then retire and their source of income is the retirment benefit.

Going back into the model without borrowing constraing, to introduce this feature in the model we will assume that agents will work for 45 years (will retire at the age of 65) and will die with certainty after 20 years of retirement (at age

FIGURE 2. Life cycle paths with Borrowing Constraints



84). We also need to add a social security system that will tax agents during their active life and pay the retirement benefits to retired agents.

The structure of the problem will remain the same, the only thing that changes is the agent's budget constraint. Until age 65 the agent will receive the age for his work and pay the social security tax over his wage, and after 65 he will receive a retirement benefit instead of the wage. So the budget constraint of an agent born i periods ago will become

$$c_{t+i,i} + k_{t+i+1,i+1} = \begin{cases} (1 + r_{t+i})k_{t+1,i} + \epsilon_i w_{t+i}(1 - \tau^{ss}) & \text{if } i \leq 45 \\ (1 + r_{t+i})k_{t+1,i} + \epsilon_i b & \text{if } i > 45 \end{cases}$$

(5.1)

where b are the retirement benefits.

To solve this problem we will need an extra condition in the steady state to guarantee that the social security system is in equilibrium. Assuming that we have a mass of agents equal to one in each age cohort this condition is given by

$$\sum_{i=1}^{45} (\tau^{ss} \epsilon_i w) = b * (84 - 64)$$

where 84 is the age at which the agents die with certainty and 64 is the last working year.

In the matlab code we need to add an extra loop to the code used in section "Exogenous labor supply" to guarantee that the condition for a balanced social security system is satisfied. The algorithm to solve this problem will be:

(1) Guess benefits

(2) Guess aggregate capital

(3) Compute aggregate labor

(4) Compute $r = \frac{\partial Y}{K}$.

(5) Compute $w = \frac{\partial Y}{H}$.

(6) Given r, w solve the agents' (life cycle) problem.

(7) Compute aggregate assets A.

(8) If $A > K$ decrease r. If not, increase it.

(9) Repeat 2-8 until $A = K$.

(10) If $B > T$ decrease b, If not, increase it

(11) Repeat 1-10 until $B = T$

where B are the total benefits pay by social security, T the total taxs and b the benefit per agent. As we see the solution is similar to the case of the exogenous labor supply, we just add another loop to guarantee a balanced social security system.

Let's solve the following exercise:

EXERCISE 16. *Assume agents are born at 20 years old, retire at 65 and die at 84. There is a social security system that pays retirement benefits and taxes the wages of the working agents. Assuming a social security tax of 0.15 find the steady state capital stock and associated interest rate. Plot an agent's schedule for savings, consumption and assets.*

The code that solves exercise 16

```
clear all; close all; clc;

global w T beta gamma R prod retire tau b

%parameter
alpha = 0.3;
beta  = 0.96;
delta = 0.08;
mu    = 2;
gamma = 1;
tau   = 0.15;

% life cycle variables

born   = 20;
dies   = 84;
retire = 45;
T      = 65;

% productivity function with retirement age
```

```
prod(1:retire,1) = -0.5350819+0.069134*(born:born+retire-1)-0.000713*(born:born+retire-1).^2;
prod(retire+1:T) = 0;

% Aggregate labor

H=sum(prod);

% Initial guess for benefits
b0    = -9;
b1    = 10;
b     = (b0+b1)/2;
B0    = b0*(dies-(retire+born));
B1    = b1*(dies-(retire+born));

% Tolerance levels and iterations

tolk    = 1e-2;
tolb    = 10^-6;
diffb   = 10;
itermax = 30;

while (abs(diffb)>tolb)
K0    = H;
K1    = 10*H;
K     = (K1+K0)/2;
guess = zeros(T-1,1);
iter  = 0;
diffk = 10;

while (iter<=itermax && diffk>tolk)
iter = iter+1; % updates the iteration number

% compute marginal productivity of capital and labor
r = alpha*((H/K)^(1-alpha));
w = (1-alpha)*((K/H)^(alpha));
R = 1-delta+r; %Gross interest rate on capital

% solve the euler equation
[assets]=fsolve(@focs3,guess);
A  = sum(assets);
Tl = sum(tau.*prod.*w);

% check if initial guess of K equals A
diffk = abs(K-A);disp('diffk');disp(K-A);
guess=assets; % updates the guess

%update K
if A>K
```

```
K0=K;
else
K1=K;
end

K=(K1+K0)/2;

end

B  = b*(dies-(retire+born));
diffb=abs(Tl-B)

%update the guess for b
if sign(Tl-B) == sign(Tl-B0);
b0=(b0+b1)/2;
elseif sign(Tl-B) == sign(Tl-B1);
b1=(b0+b1)/2;
end

B0    = b0*(dies-(retire+born));
B1    = b1*(dies-(retire+born));
b     = (b0+b1)/2;


end

wts = prod*w; %wages
%plots & Capital stock and interest rate
figure('Name','Consumption, assets, steady state capital and interest rate')
[euler consumption]=focs2(assets);
assets=[assets;0];
% Check the lifetime budget constraint and the aggregate resource
% constraint
r=((1/R).*ones(65,1)).^((0:64)');
lfbc=r'*(consumption'-w*prod);
set(0,'DefaultAxesColorOrder',[0 0 0;0.5 0.5 0.5])
set(0,'DefaultAxesLineStyleOrder','-|-.|.')
plot(20:84,[assets consumption' wts]);
title(['Steady state capital stock is ',num2str(K),...
' with an interest rate of ', num2str((R-1)*100),'%'])
xlabel('Time');ylabel('c(t),a(t),w(t)');axis('tight');
legend('Assets','Consumption','Wage','Location','NorthWest')
hold off
```

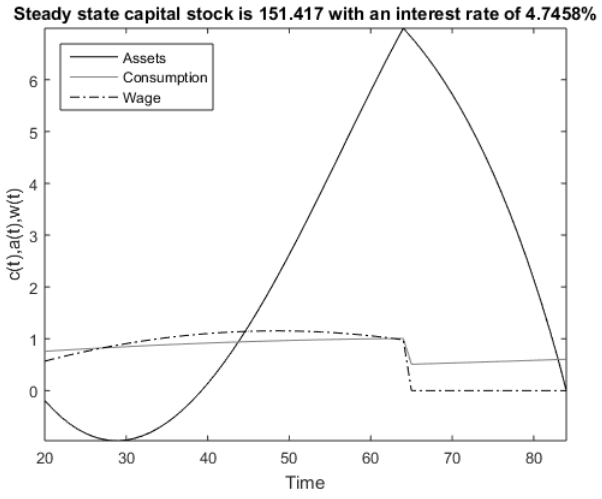
Notice that this code calls the focs3.m function, which contains the first order conditions.

```
function [euler,consumption]=focs3(x) %x initial guess of assets y initial guess of benefits
```

```
global w T beta gamma R prod retire tau b

a=x;
c=zeros(T,1);
%first period budget constraint
c(1) =prod(1)*w*(1-tau)-a(1);

%last period budget constraint
c(T) = b+a(T-1);

%working lyfe budget constraing
c(2:retire)=R*a(1:retire-1)+prod(2:retire).*w.*(1-tau)-a(2:retire); %budget constraint

%retired budget constraint
c(retire+1:T-1)=R*a(retire:T-2)+b-a(retire+1:T-1);

%solve the euler equation
euler(1:T-1,1)=-c(1:T-1).^(-gamma)+(beta*R).*c(2:T).^(-gamma);

%time path of consumption
consumption(1:T)=[c(1); c(2:T-1); c(T)];
```

As we can see in figure 3 the agents will save until the retirement age and then will consume everything they saved until they die. This happens since the benefit they will receive is smaller than the wage they received while working so they will save for retirement.

FIGURE 3. Life cycle paths with retirement



CHAPTER 9

The Solow Model

In this part we will study the model brought forth by **?**. It has become the foundation for modern macroeconomics and it is the basis for most models in the growth literature[1]. We will start by showing the equilibrium conditions of the model. Then we will simulate the model with the exact solution and with an approximation technique that has become standard in macroeconomics, log-linearization. Unlike most applications, in this case we have the exact solution and can compare it with the approximation.

1. The general setup

There is an aggregate production function that maps the level of technology A_t and capital and labor inputs, K_t and L_t respectively, to output Y_t:

$$Y_t = A_t F(K_t, L_t) \tag{1.1}$$

The capital next period consists of undepreciated capital plus investment:

$$K_{t+1} = (1-\delta)K_t + I_t \tag{1.2}$$

Assume that labour grows at a constant rate n:

$$L_{t+1} = (1+n)L_t \tag{1.3}$$

Assume that F is homogeneous of degree one,i.e.:

$$F(\alpha K_t, \alpha L_t) = \alpha F(K_t, L_t) \tag{1.4}$$

and let lowercase variables denote the variable value in per worker terms. We can write the production function as:

$$y_t = \frac{Y_t}{L_t} = A_t F\left(\frac{K_t}{L_t}, \frac{L_t}{L_t}\right) = A_t F(k_t, 1) \equiv A_t f(k_t) \tag{1.5}$$

Similarly, equation (1.2) becomes:

$$(1+n)k_{t+1} = (1-\delta)k_t + i_t \tag{1.6}$$

Investment equals savings in equilibrium, and since in this model, savings are a constant share of output:

$$i_t = s_t = \sigma y_t \tag{1.7}$$

We can now rewrite equation (1.6) as:

$$(1+n)k_{t+1} = (1-\delta)k_t + \sigma A_t f(k_t) \tag{1.8}$$

The level of technology is given by $A_t = A_0(1+\alpha)^t$ so that equation 1.8 becomes

[1]This section is a streamlined version of Chapter 1 of **?**

$$(1+n)k_{t+1} = (1-\delta)k_t + \sigma A_0(1+\alpha)^t f(k_t) \tag{1.9}$$

Assume now that there is no technological growth, i.e., $\alpha = 0$. Then we have that

$$k_{t+1} = g(k_t) = \frac{1-\delta}{(1+n)}k_t + \frac{\sigma A_0}{(1+n)}f(k_t) \tag{1.10}$$

where $g(k_t$ is the policy function, i.e., describes the next period capital stock as a function of the current environment. The economy has two stationary solutions. One is when $k_t = k_{ss} = 0$. To find the other we assume that $f(k) = k^\theta$. Then we set $k_{t+1} = k_t = k_{ss}$ and get:

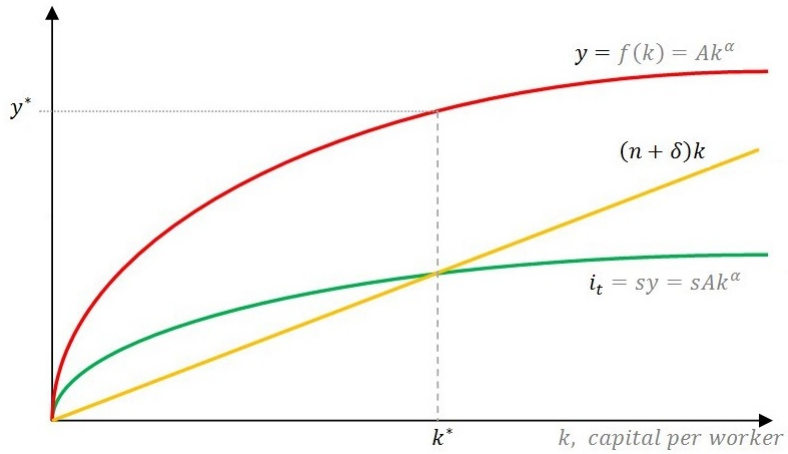
$$k_{ss} = \frac{1-\delta}{(1+n)}k_{ss} + \frac{\sigma A_0}{(1+n)}k_{ss}^\theta \tag{1.11}$$

Solving w.r.t. k_ss we get:

$$k_{ss} = \left(\frac{\delta+n}{\sigma A_0}\right)^{\frac{1}{\theta-1}} \tag{1.12}$$

To better understand the dynamics of the Solow model, the picture below plots output, savings and depreciated capital as a function of capital per worker.

FIGURE 1. The Solow Model



Notice that when capital is below the steady state, savings will be greater than depreciated capital adjusted by the population growth rate. In this case capital the next period will increase. This will continue until the two effects cancel and the economy reaches the steady state.

2. A stochastic Solow model

We will now assume that technology follows a stochastic process. This is the basis for real business cycle theory brought forth by **?**. Let

$$A_t = A_0 e_t^{\varepsilon} \tag{2.1}$$

where ε_t follows the normal distribution with mean zero.

Then the law of motion for capital is given by

$$k_{t+1} = \frac{1-\delta}{(1+n)} k_t + \frac{\sigma A_0 e_t^{\varepsilon}}{(1+n)} k_t^{\theta} \tag{2.2}$$

Subtract and divide both sides by $k_s s$ to get the law of motion for capital in deviations from the steady state:

$$\frac{k_{t+1} - k_{ss}}{k_{ss}} = \frac{(1-\delta)k_t + \sigma A_0 e_t^{\varepsilon} k_t^{\theta} - k_{ss}(1+n)}{(1+n)k_{ss}} \tag{2.3}$$

We will now conduct the log linear approximation to the equation above. For more details on the topic of log-linearization see **?**. We start by defining the notation for the logarithmic difference of a variable from its steady-state level

$$\tilde{X}_t = \ln X_t - \ln X_{ss} \tag{2.4}$$

Exponentiate both sides and rearrange to get:

$$X_t = X_{ss} e^{\tilde{X}_t} \tag{2.5}$$

For small deviations from steady state values, the first order approximations below are reasonably accurate:

$$e^{\tilde{X}_t} \approx 1 + \tilde{X}_t \tag{2.6}$$

$$e^{\tilde{X}_t + a\tilde{Y}_t} \approx 1 + \tilde{X}_t + a\tilde{Y}_t \tag{2.7}$$

$$\tilde{X}_t \tilde{Y}_t \approx 0 \tag{2.8}$$

$$E_t\left(a e^{\tilde{X}_{t+1}}\right) \approx E_t\left(a\tilde{X}_{t+1}\right) + c \tag{2.9}$$

We start with the law of motion for capital:

$$k_{t+1} = \frac{1-\delta}{(1+n)} k_t + \frac{\sigma A_0 e^t}{(1+n)} k_t^{\theta} \tag{2.10}$$

and replace k_t by $k_{ss} e^{\tilde{k}_t}$, where $\tilde{k}_t = \ln k_t - \ln k_{ss}$:

$$(1+n)k_{ss} e^{\tilde{k}_{t+1}} = (1-\delta)k_{ss} e^{\tilde{k}_t} + \sigma A_0 e^{\varepsilon_t} k_{ss}^{\theta} e^{\theta\tilde{k}_t} \tag{2.11}$$

Use the first rule of approximation to get:

$$(1+n)k_{ss}\left(1+\tilde{k}_{t+1}\right) = (1-\delta)k_{ss}\left(1+\tilde{k}_t\right) + \sigma A_0(1+\varepsilon_t)k_{ss}^t heta\left(1+\theta\tilde{k}_t\right) \tag{2.12}$$

In the steady-state we have that

$$(1+n)k_{ss} = (1-\delta)k_{ss} + \sigma A_0 k_{ss}^{\theta} \tag{2.13}$$

so we can use this to simplify equation 2.12 and get

$$(1+n)k_{ss}\tilde{k}_{t+1} = (1-\delta)k_{ss}\tilde{k}_t + \sigma A_0 \varepsilon_t k_{ss}^{\theta} + \sigma A_0 k_{ss}^{\theta}\theta\tilde{k}_t + \sigma A_0 k_{ss}^{\theta}\theta\varepsilon_t\tilde{k}_t \tag{2.14}$$

Since $\epsilon_t\tilde{k}_t \approx 0$, we get

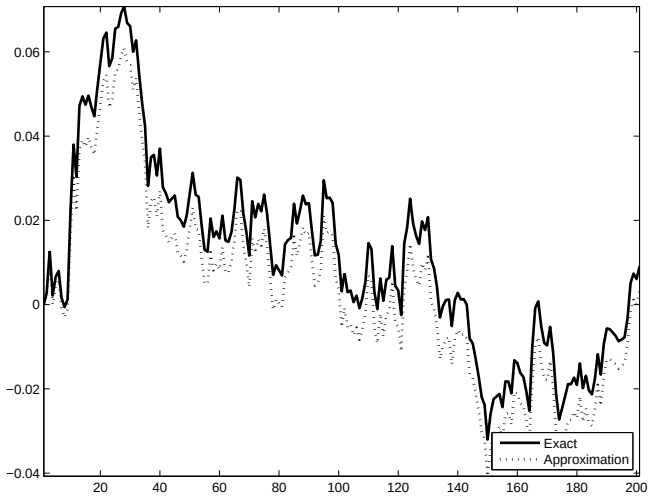
$$(1+n)k_{ss}\tilde{k}_{t+1} = (1-\delta)k_t ss\tilde{k}_t + \sigma A_0 \varepsilon_t k_{ss}^{\theta} + \sigma A_0 k_{ss}\theta\tilde{k}_t \tag{2.15}$$

After some manipulations we get our log-linear approximation:

$$\tilde{k}_{t+1} = \frac{1+\theta n - \delta(1-\theta)}{1+n}\tilde{k}_t + \frac{\delta+n}{1+n}\varepsilon_t \tag{2.16}$$

We are now ready to compare our log-linear and exact solutions. In Figure 2 we show the exact solution and the log-linear approximation for the case when $(var(\varepsilon_t) = 0.1)$:

FIGURE 2. Log-Linear approximation

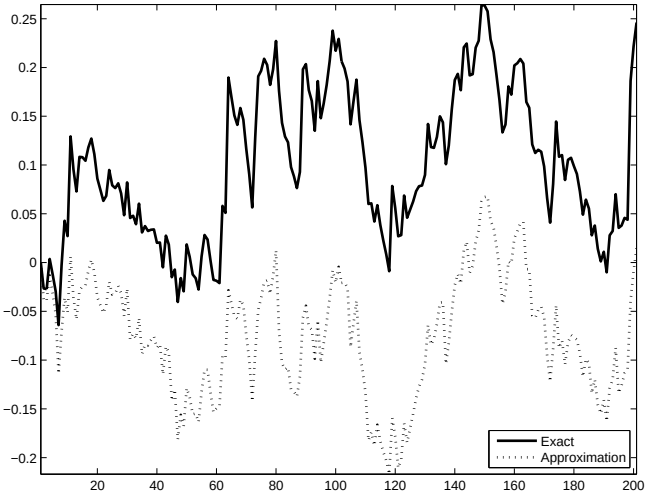


As we see, the approximation is reasonably accurate. However, let's see how the log-linear approximation behaves for larger shocks ($var(\varepsilon_t = 0.5)$ that lead the economy to substantially bigger deviations from the steady-state:

As we can see in Figure 3, the approximation behaves poorly for large deviations. One should be careful in analyzing conclusions from log-linear approximations when observing large deviations from steady-state. Below the code that generates the data for the plots above:

```
% Parameters
    A = 1;     s = 0.3;    delta = 0.05;  n  = 0.00;  theta = 1/3;
```

FIGURE 3. Log-Linear approximation




```
%steady state values
kss = ((delta+n)/(s*A)).^(1/(theta-1));  yss = kss.^theta;

% Generate processes for A
p = 200;
At1  = A*exp(0.1*randn(p,1));

%generate paths for capital for both exact solution and loglinear approx
kts = zeros(p,1)*NaN;lkt = zeros(p,1)*NaN;
kts(1) = kss;lkt(1)=0;
for i = 1:p
    kts(i+1)=((1-delta)*kts(i)+s*A*At1(i)*kts(i)^theta)/(1+n);
    lkt(i+1)=(1+theta*n-delta*(1-theta))/(1+n)*lkt(i)+...
        (delta+n)/(1+n)*log(At1(i));
end

% express the exact solution in deviations from the steady state
devkts = (kts-kss)./kss;
```

APPENDIX A

The First Appendix

The `\appendix` command should be used only once. Subsequent appendices can be created using the Chapter command.

APPENDIX B

The Second Appendix

Some text for the second Appendix.