

## Octave codes

### Setting the path and current folder automatically after running Octave

Create a file `.octaverc` (no extension) in catalogue bin of Octave installation (e.g. `C:\Octave\3.2.3_gcc-4.4.0\bin`). Enter there commands setting the path (`addpath HOME;`) and current folder (`cd HOME;`), where `HOME` is the full path of the catalogue where you make and store your calculations.

For example, the content of `.octaverc` may look like this:

```
addpath D:\Advanced_macro;  
cd D:\Advanced_macro;
```

### HP filter

Download from the Internet a Matlab/Octave code calculating the HP filter (e.g. the code written by I. IZOVORSKI available at <http://dge.repec.org/codes/izvorski/hpfilter.m>). Download appropriate data and save them as a matrix (without text headings) in one of the formats accepted by Octave, e.g. a comma delimited text file (CSV). Load the data to Octave, make appropriate transformations and use the function `hpfilter.m`.

An exemplary code (additionally generating a simple chart, storing it in an EPS format and exporting the data to a CSV file) may look like this:

```
load data_gdp.csv;  
lgdp=log(data_gdp(:,1));  
lgdp_hp=hpfilter(lgdp,1600);  
plot([lgdp lgdp_hp]);  
legend('Original data','HP filter 1600');  
axis([0, size(lgdp,1)]);  
print -depsc2 figure1.eps;  
save data_gdp_hp.csv lgdp lgdp_hp;
```

## Shooting

The algorithm below finds the initial level of consumption and generates the saddle path for consumption and capital in the following optimization problem (for given  $k_0 > 0$ ):

$$\max_{\{c_t\}_{t=0}^{\infty}, \{k_t\}_{t=1}^{\infty}} \sum_{t=0}^{\infty} \beta^t \frac{c_t^{1-\theta}}{1-\theta}$$

p.w.  $k_{t+1} = (1-\delta)k_t + k_t^\alpha - c_t$

```
%Setting parameters, initial conditions and the algorithm options
alpha=0.33; %capital share in output
beta=0.99; %discount factor
delta=0.025; %depreciation rate
theta=1; %inverse of the intertemporal elasticity of substitution
k0=10; %initial capital level
max_iter=1000; %maximum number of iterations
conv=0.001; %convergence criterion (in per cent of steady state consumption)

%Steady state
k_ss=((1/beta-1+delta)/alpha)^(1/(alpha-1));
c_ss=k_ss^alpha-delta*k_ss;

%Choosing bounds for c0
if k0<k_ss
    c_minmax=[0 c_ss];
elseif k0>k_ss
    c_minmax=[c_ss (1-delta)*k0+k0^alpha];
else
    c0=c_ss;
    disp(['The economy starts from the steady state. c0=' num2str(c0)]);
    break;
end

for i=1:max_iter
    %Setting c0
    c0=0.5*(c_minmax(1,2)+c_minmax(1,1));
    %Variable initialization
    k=[k0];
    c=[c0];
    kt=k0;
    ct=c0;
    %Inner loop generating paths until they "bend"
    while (k0<k_ss & kt>=k(max(end-1,1),1) & ct>=c(max(end-1,1),1)) | ...
        (k0>k_ss & kt<=k(max(end-1,1),1) & ct<=c(max(end-1,1),1))
        kt=(1-delta)*k(end,1)+k(end,1)^alpha-c(end,1);
        ct=(beta*(alpha*kt^(alpha-1)+1-delta))^(1/theta)*c(end,1);
        k=[k;kt];
        c=[c;ct];
    end
    %Checking if we are sufficiently close to the steady state
    if 100*abs(c(end,1)/c_ss-1)<conv
        break;
    end
end
```

```
%Checking if c0 is too big or too small and appropriate correction of c_minmax
if c(end,1)>c(end-1,1)
    c_minmax(1,2)=c0;
else
    c_minmax(1,1)=c0;
end
end

%Checking if the number of iteration is sufficient
if i==max_iter
    disp('Maximum number of iterations reached. The convergence criterion is not satisfied.
Increase max_iter.');
```

Increase max\_iter.');

```
else
    disp(['Solution: c0=' num2str(c0)]);
    plot(k,c,'o');
    title('Saddle path');
    xlabel('k');
    ylabel('c');
end
```

### Blanchard-Kahn algorithm<sup>1</sup>

The equilibrium of a standard real business cycle model with fixed labor can be written with the following three equations:

$$\begin{aligned}\ln A_{t+1} &= \rho \ln A_t + \varepsilon_{t+1} \\ k_{t+1} &= (1-\delta)k_t + A_t k_t^\alpha - c_t \\ c_t^{-\theta} &= \beta E_t \left\{ c_{t+1}^{-\theta} (1-\delta + \alpha A_{t+1} k_{t+1}^{\alpha-1}) \right\}\end{aligned}$$

After log-linearization we obtain:

$$\begin{aligned}\hat{A}_{t+1} &= \rho \hat{A}_t + \varepsilon_{t+1} \\ \hat{k}_{t+1} &= \beta^{-1} \hat{k}_t + \frac{\beta^{-1} - 1 + \delta}{\alpha} \hat{A}_t + \left( \delta - \frac{\beta^{-1} - 1 + \delta}{\alpha} \right) \hat{c}_t \\ -\theta \hat{c}_t &= -\theta E_t \{ \hat{c}_{t+1} \} + \beta (\beta^{-1} - 1 + \delta) E_t \{ \hat{A}_{t+1} \} - \beta (\beta^{-1} - 1 + \delta) (1 - \alpha) E_t \{ \hat{k}_{t+1} \}\end{aligned}$$

In this model, we hence have  $n=2$  state variables (technology and capital),  $m=1$  jumpers (consumption) and  $k=1$  shock (to technology).

In order to write this system as in equation (1), we have to get rid of expectations of the state variables. The first two equations imply:

$$\begin{aligned}E_t \hat{A}_{t+1} &= \rho \hat{A}_t \\ E_t \hat{k}_{t+1} &= \hat{k}_{t+1}\end{aligned}$$

The code below implements all steps of the Blanchard-Kahn algorithm for the model defined above, giving the solution in form of matrices of the recursive representation (see equations (8) and (9)):

$$\begin{aligned}P_t &= \Gamma X_t \\ X_t &= \Phi X_{t-1} + \Psi Z_t\end{aligned}$$

#### %Setting parameters

```
alpha=0.33;
beta=0.99;
delta=0.025;
theta=1;
rho=0.98;
```

```
n=2;
m=1;
```

#### %Ordering of variables: A, k, c

```
A1=[1 0 0; 0 1 0; 0 beta*(1/beta-1+delta)*(1-alpha) theta];
A0=[rho 0 0; (1/beta-1+delta)/alpha 1/beta delta-(1/beta-1+delta)/alpha; ...
```

<sup>1</sup> In this example we refer to the equations used in the additional material „Solving DSGE models using the Blanchard-Kahn algorithm“.

```

beta*(1/beta-1+delta)*rho 0 theta];
gamma=[1; 0; 0];
A=A1^(-1)*A0;

%Jordan decomposition
[C,lambda]=eig(A);
%Sorting matrices lambda and C
[i,h]=sort(diag(abs(lambda)));
lambda=lambda(h,h);
C=C(:,h);

Cinv=C^(-1);
m_star=sum(diag(abs(lambda))>1);
n_star=n+m-m_star;

%Checking conditions of Theorem 1
if m_star!=m
    disp('The Blanchard-Kahn condition is not satisfied. ');
    break;
end
if det(Cinv(n_star+1:end, n_star+1:end))==0
    disp('The rank condition is not satisfied. ');
    break;
end

%Solution
Gam=-(Cinv(n_star+1:end, n_star+1:end))^(-1)*Cinv(n_star+1:end,1:n_star);
Phi=(A(1:n_star,1:n_star)+A(1:n_star,n_star+1:end)*Gam);
Psi=A1(1:n_star,1:n_star)^(-1)*gamma(1:n_star,1);

```

## Value function iteration

Consider the following model of optimal capital accumulation:

$$\max_{\{c_t\}_{t=0}^{\infty}, \{k_t\}_{t=1}^{\infty}} \sum_{t=0}^{\infty} \beta^t \frac{c_t^{1-\theta}}{1-\theta}$$

p.w.  $k_{t+1} = (1-\delta)k_t + k_t^\alpha - c_t$

The Bellman equation for the problem above is:

$$V(k_t) = \max_{c_t} \left\{ \frac{c_t^{1-\theta}}{1-\theta} + \beta V((1-\delta)k_t + k_t^\alpha - c_t) \right\}$$

which alternatively can be written using the state variables only (i.e. capital) as:

$$V(k_t) = \max_{k_{t+1}} \left\{ \frac{\left( (1-\delta)k_t + k_t^\alpha - k_{t+1} \right)^{1-\theta}}{1-\theta} + \beta V(k_{t+1}) \right\}$$

The code below solves this problem by iterating the value function over a discretized state space.

```
%Setting parameters, initial conditions and the algorithm options
beta=0.99; %discount factor
delta=0.025; %depreciation rate
theta=2; %inverse of intertemporal elasticity of substitution
alpha=0.33; %capital share
k0=20; %initial capital level
S=100; %number of states between (and including) k0 and k_ss
conv=1e-6; %convergence criterion

%Steady state
k_ss=((1/beta-1+delta)/alpha)^(1/(alpha-1));
c_ss=k_ss^alpha-delta*k_ss;

%Vector of states
ks=k0:(k_ss-k0)/(S-1):k_ss;

%Instantaneous utility
reward=-Inf*ones(S,S);
for j=1:S %for every state today
    for i=1:S %for every state tomorrow
        if ((1-delta)*ks(j)+ks(j)^alpha-ks(i)>=0) %consumption cannot be negative
            reward(i,j)=(((1-delta)*ks(j)+ks(j)^alpha-ks(i))^(1-theta))/(1-theta);
        end
    end
end

value=zeros(1,S); %Starting from value function equal to zero

%Value function iteration
progress=1;
while progress>conv %until improvement in value function is sufficiently large
    valueold=value;
    [value contr]=max(reward+beta*value'*ones(1,S)); % Bellman equation
    progress=max(abs(value-valueold));
end
```

```
%Order of states for capital
```

```
korder(1)=1;
```

```
for i=2:S
```

```
    korder(i)=contr(korder(i-1));
```

```
end
```

```
%Solution
```

```
k(1:S)=ks(korder(1:S));
```

```
c(1:(S-1))=(1-delta)*k(1:(S-1))+k(1:(S-1)).^alpha-k(2:S); c(S)=c(S-1);
```

### Solving numerically a nonlinear system of equations

To solve a nonlinear system of equations using numerical methods one can use function `fsolve`. An illustrative implementation is discussed below.

Let us assume that we want to find a solution to the following system of equations:

$$ax_1^2 + \frac{1}{x_2} = b$$

$$x_1x_2 = c$$

where  $x_1$  and  $x_2$  are the unknowns, while  $a$ ,  $b$  and  $c$  are the parameters.

To solve this system we use function `fsolve`. Calling it, together with necessary declarations can be coded as follows:

```
%Setting parameters
```

```
a=1; b=2; c=3;
```

```
%Declaring parameters so that they are seen within the function to solve
```

```
global a b c;
```

```
%Starting values (e.g. x1=1, x2=1)
```

```
x0= [1;2];
```

```
%Calling fsolve
```

```
y = fsolve(@system,x0);
```

```
%Solution
```

```
x1=y(1)
```

```
x2=y(2)
```

where the system of equations to solve is defined in function `system.m`, which is an argument to `fsolve`. The contents of `system.m` are:

```
function y=system(x)
```

```
x1=x(1); x2=x(2);
```

```
global a b c;
```

```
y=[a*x1^2+1/x2-b;
```

```
    x1*x2-c];
```

The key to success while solving numerically highly nonlinear equations is a good choice of starting values. In economic applications, the steady state solution can serve as such. Of course, if the problem to solve has multiple solutions, using numerical methods will not always result in obtaining the one we are interested in. Hence, appropriate diagnostics is a necessary step (e.g. checking if the solution satisfies nonnegativity restrictions).