Dynamic Programming and Projections Methods

An Advanced Course In The Science and Art of DSGE Modelling

Hyungseok Joo

Ricardo Nunes

Donghyun Park

Centre for International Macroeconomic Studies

School of Economics, University of Surrey

Why should I bother?

- All other methods you might know assume recursivity, i.e. the main results of dynamic programming
- All economic models can be expressed as functional equations
- Strong presence of non-linearities that induces a very different behavior of the system when far from the steady state. Example: models with ZLB, see Braun et al. (2012).
- Non-convexity of the problem, implying FOCs are not sufficient. Example: (heterogeneous agents) models with discrete choices in labor (work/not work decision) as in Chang and Kim (2007).

TWO METHODS

We will see two methods for looking at these models:

- Dynamic programming
  - Value function iteration
  - Policy function iteration
- Projections methods
  - Background: standard numerical methods

MATERIAL

- Review of DP:
  - [LS] Ljungqvist, L., and T. J. Sargent (2004), *Recursive Macroeconomic Theory*, Second Edition, MIT Press
  - [AC] Adda J., and Cooper (2003) *Dynamic Economics: Quantitative Methods and Applications*, MIT Press
  - [SLP] Stokey, N., R. Lucas and E. Prescott (1989), *Recursive Methods for Economic Dynamics*, Harvard University Press
- Books on numerical methods:
  - [Judd] Judd K. (1998), *Numerical Methods in Economics*, MIT Press
  - [MF] Miranda, M. J. and P. L. Fackler (2002), *Applied Computational Economics and Finance*, MIT Press

Solving a simple RBC model

We can present the main idea of dynamic programming techniques by using a RBC model without uncertainty (for the moment)

Production function:

$$Y_t = F(A_t, L_t, K_{t-1}) \tag{1}$$

Utility function of the representative household

$$U_t = U(C_t) \tag{2}$$

SIMPLIFYING ASSUMPTIONS

ASSUMPTION 1 (UTILITY FUNCTION)
$U : R_+ \to R$ is bounded, continuously differentiable, strictly increasing, strictly concave and $\lim_{C\to 0} U'(C) = \infty$.

$\Rightarrow$ labor supply is equal to 1

Simplifying assumptions (cont.)

Assumption 2 (**Production function**)

The production function $F : R_+^3 \to R_+$ is continuously differentiable, strictly increasing, homogeneous of degree 1 and strictly quasi-concave, with

$$F_K > 0; F_{KK} < 0; F_L > 0; F(A, L, 0) = 0 \qquad \forall K, L > 0$$

$$\lim_{K \to 0} F_K(A, 1, K) = \infty, \lim_{K \to \infty} F_K(A, 1, K) = 0 \qquad \text{(Inada conditions)}$$

$\Rightarrow$ interiority and uniqueness of the solution. Cobb-Douglas satisfies these properties

Resource constraint

$$C_t + K_t - (1-\delta) K_{t-1} \leq F(A, 1, K_{t-1}) \tag{3}$$

where we have assumed that $G_t = 0$.

First welfare theorem

- No market failures
- First welfare theorem applies
- Competitive equilibrium is Pareto efficient
- Benevolent social planner problem: maximize HH discounted utility subject to resource constraint and law of motion for capital, K_{-1} given

Another way of writing it...

Define

$$f(K) \equiv F(A,1,K) + (1-\delta)K$$

Rewrite the planner's problem as

$$\max_{\{K_t\}_{t=0}^{\infty}} \quad \sum_{t=0}^{\infty} \beta^t U\left(f(K_{t-1}) - K_t\right) \tag{4}$$

$$s.t. \quad 0 \leq K_t \leq f(K_{t-1}), \quad t = 0, 1, ... \tag{5}$$

$$K_{-1} \text{ given}$$

Value function

Imagine we already solved the problem for any K_{-1}. Define

$$V(K_{-1}) \equiv \max_{\{K_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t U\big(f(K_{t-1}) - K_t\big)$$

$$s.t. \quad 0 \leq K_t \leq f(K_{t-1}), \quad t = 0, 1, \ldots$$

The function $V : R_+ \to R$ is called **the value function**.

Recursive formulation

The planner problem can be rewritten as:

$$\max_{C_0,K_0}\left[U(C_0)+\beta\max_{\{0\leq K_t\leq f(K_{t-1})\}_{t=1}^{\infty}}\sum_{j=0}^{\infty}\beta^j U\Big(f\left(K_{t+j-1}\right)-K_{t+j}\Big)\right]$$

$$s.t.\quad C_0+K_0\leq f\left(K_{-1}\right)$$

$$C_0,K_0\geq 0,\quad K_{-1}>0\quad \text{given}$$

Recursive formulation

The planner problem can be rewritten as:

$$\max_{C_0,K_0}\left[U(C_0)+\beta\underbrace{\max_{\left\{0\le K_t\le f(K_{t-1})\right\}_{t=1}^{\infty}}\sum_{j=0}^{\infty}\beta^{j}U\left(f\left(K_{t+j-1}\right)-K_{t+j}\right)}_{V(K_0)}\right]$$

$$s.t.\quad C_0+K_0\le f(K_{-1})$$

$$C_0,K_0\ge 0,\quad K_{-1}>0\quad \text{given}$$

$V(K_0)$ is the value of the utility from period 1 on, obtained with an initial capital K_0.

RECURSIVE FORMULATION

The planner problem can be rewritten as:

$$\max_{C_0,K_0}\left[U\left(C_0\right)+\beta V\left(K_0\right)\right] \tag{6}$$

$$s.t. \quad C_0+K_0\leq f\left(K_{-1}\right) \tag{7}$$

$$C_0,K_0\geq 0, \quad K_{-1}>0 \quad \text{given} \tag{8}$$

$V\left(K_0\right)$ is the value of the utility from period 1 on, obtained with an initial capital K_0.

Recursive formulation (here we go!)

- It must be that

$$V\left(K_{-1}\right) = \max_{0 \leq K_0 \leq f\left(K_{-1}\right)} \left\{ U\left(f\left(K_{-1}\right) - K_0\right) + \beta V\left(K_0\right) \right\}$$

- We don't need time subscripts, this is a static problem!

$$V\left(K\right) = \max_{0 \leq y \leq f\left(K_{-1}\right)} \left\{ U\left(f\left(K\right) - y\right) + \beta V\left(y\right) \right\} \tag{9}$$

- The unknown is a function (the value function V), this is a *functional equation* or *Bellman equation*

STOCHASTIC CASE

- A_t is Markov with discrete support: $A_t \in \left\{ \widehat{A_1}, \ldots, \widehat{A_S} \right\}$
  - (just for simplicity, can have continuous support)
- Exactly the same!

$$V(K,A) = \max_{0 \leq y \leq f(K,A)} U(K,y,A) + \beta \sum_{A'} \pi \left(A'|A\right) V\left(y,A'\right) \quad (10)$$

Value function iteration

- Very general, can *potentially* solve any model
- Pretty *slow*, becomes *unmanageable* as the state space increases
- Can be *adapted to models with time inconsistency, optimal policy problems and New Keynesian models* (see Marcet and Marimon (2011))
- Needs *substantial knowledge of general numerical methods, programming skills and coding time*

How to find the solution

Algorithm

1. Start from an arbitrary value function $V^{(0)}$
2. Get $V^{(1)} = TV^{(0)}$,
3. Keep iterating on step 2 until you reach the fixed point V

Contraction mapping theorem: convergence is pointwise

Discrete grid methods

- Computers and continuous variables do not love each other
- We need to **discretize** the number of possible choices we have (**grid**)
- This is a first source of approximation (depends on number of points)

Define a grid $G \equiv \{x_1, ..., x_m\}$

Value function iteration algorithm

1. We form an initial guess for the value function for each point on the grid
2. For any $n \geq 0$, compute $V^{(n+1)}(\cdot) = TV^{(n)}(\cdot)$ for any point in the grid
3. Stop if $\left\| V^{(n+1)} - V^{(n)} \right\| < \varepsilon$

SOLVING STOCHASTIC GROWTH MODEL

Same as deterministic, except for:

$$y_t = A_t F(k_t, n_t)$$

where A_t is a Markov process with transition probability matrix $\mathscr{P}$

Vectorizing the problem

- Productivity shock: $[A_1, A_2]$
- n grid points for capital $[k_1, k_2, ..., k_n]$
- Define two matrices U_j such that:

$$U_j(i,h) = u\left(A_j f(k_i) + (1-\delta)k_i - k_h\right), \quad i = 1, ..., n, \quad h = 1, ..., n$$

Vectorizing the problem (cont.)

- Define two $n \times 1$ vectors $V_j, j = 1, 2$ such that

  $V_j\left(i\right) = V_j\left(k_i, A_j\right), i = 1, ..., n.$
- Define an operator $T\left([V_1, V_2]\right)$ that maps couples of vectors

  $[V_1, V_2]$ into a couple of vectors$[TV_1, TV_2]$:

$$TV_1 = \max\{U_1 + \beta \mathscr{P}_{11} \mathbf{1} V_1^{\prime} + \beta \mathscr{P}_{12} \mathbf{1} V_2^{\prime}\}$$

$$TV_2 = \max\{U_2 + \beta \mathscr{P}_{21} \mathbf{1} V_1^{\prime} + \beta \mathscr{P}_{22} \mathbf{1} V_2^{\prime}\}$$

Compact form

We can write these equations in compact form:

$$\begin{bmatrix} TV_1 \\ TV_2 \end{bmatrix} = \max \left\{ \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} + \beta \left(\mathscr{P} \otimes \mathbf{1} \right) \begin{bmatrix} V_1^{'} \\ V_2^{'} \end{bmatrix} \right\} \tag{11}$$

We can solve by iterating on the operator T until convergence.

THE CODE

(BTW: now it's a good time to open Matlab...)

Three files:

- `parameters.m`: contains parameters' values and the grid
- `vfi_AM.m`: main routine that implements the value function iteration
- `figures.m`: draws graphs

A do-file makes easy for you to run the code: `do_vfi_AM.m`

parameters.m

```
%%  set parameter values
alpha  = 0.40;                % production parameter
beta   = 0.95;                % subjective discount factor
prob   = [ .5 .5; .5 .5]; % prob(A(t+1)=Aj | A(t) = Ai)
delta  = .90;                 % 1 - depreciation rate
A_high = 1.5;                 % high value for technology
A_low  = 0.5;                 % low value for technology
convcrit = 1e-7;        % convergence criterion (epsilon)

%%   generate capital grid
mink =   0.01;    % minimum value of the capital grid
maxk =  25.01;   % maximum value of the capital grid
nk   = 1000;            % number of grid points
kgrid = linspace(mink,maxk,nk)';  % the grid (linearly
    spaced)
ink = kgrid(2) - kgrid(1);          % increments
```

vfi_AM.m (I)

```
%  create the utility function matrices such that, for
%  zero or negative  consumption, utility remains a
%  large negative number so that such values will never
%  be chosen as utility maximizing

cons1 = bsxfun(@minus, A_high*kgrid'.^alpha  + delta*
    kgrid' ,kgrid);
cons2 = bsxfun(@minus, A_low*kgrid'.^alpha  + delta*
    kgrid' ,kgrid);

cons1(cons1<=0) = NaN;
cons2(cons2<=0) = NaN;

util1 =  log(cons1);
util2 =  log(cons2);

util1(isnan(util1)) = -inf;
util2(isnan(util2)) = -inf;
```

vfi_AM.m (II)

```
%%  initialize some variables

v = zeros(nk,2); % initial guess for value function:
                 % set to zero for simplicity
decis = zeros(nk,2);% initial value for policy function
metric = 10; % initial value for the convergence metric
iter = 0;
tme = cputime;
[rs,cs] = size(util1);
```

vfi_AM.m (III): THE MAIN LOOP

```
while metric > convcrit;
    contv= beta*v*prob'; % continuation value

    [tv1,tdecis1]=max(bsxfun(@plus,util1,contv(:,1)) );
    [tv2,tdecis2]=max(bsxfun(@plus,util2,contv(:,2)) );

    tdecis=[tdecis1' tdecis2'];
    tv=[tv1' tv2'];

    metric=max(max(abs((tv-v)./tv)));
    v=   tv; % .15*tv+.85*v; %
    decis= tdecis;%
    iter = iter+1;
    metric_vector(iter) = metric;
    disp(sprintf('iter = %g ; metric = %e', iter,metric
        ));
end;
disp('  ');
disp(sprintf('computation time = %f', cputime-tme));

% transform the decision index in capital choice
decis=(decis-1)*ink + mink;
```

do_vfi_AM.m

```
%% do file for vfi_AM.m
clear all

% load parameters and grid
parameters;

% run the code
vfi_AM;

% generate figures
figures;
```

PLAYING WITH THE CODE

1. Run the code `do_vfi_AM.m`. Familiarize with the output and the graphs. Now, after the line which loads parameter values, change the value for the discount factor to 0.995, by adding the following line:

   ```
         beta = 0.995
   ```

   This line changes the value set in the file `parameters.m` to a new value (this is a general way to do comparative statics by using a baseline set of parameters.) Save this file as `do_vfi_AM_beta.m` and run it. What do you notice in the convergence process? What happens to computational time? (Why?)

Playing with the code (II)

2. We now want to see what happens if we change the number of gridpoints. Change the code so that the grid has 100 gridpoints. (**Hint**: notice that after you set `nk=100`, then you also have to modify the variables `kgrid` and `ink`, therefore you need to add those lines too!!!). Save the file as `do_vfi_AM_smallgrid.m` and run it. What can you notice? Now try with 2000 gridpoints, save the file as `do_vfi_AM_largegrid.m`, and run it. Do you see any change?

PLAYING WITH THE CODE (III)

3. We want to do a series of comparative statics exercises. In order to do that, we can modify the file `do_compstat`, which is a basic structure for this task. Open the file `do_compstat.m`. You should see the following lines (I have excluded the lines that plot the simulated results for brevity):

Playing with the code (III)

```

%% Generate solution and simulation for case A
% load parameters and grid
parameters;
% solve the model via VFI
vfi_AM;
% store results in few new variables
v_A = v;
decis_A = decis;
controls_A = controls;

%% Generate solution and simulation for case B
% load parameters and grid
parameters;
% modify parameters here

% solve the model via VFI
vfi_AM;
% store results in few new variables
v_B = v;
decis_B = decis;
controls_B = controls;
```

Playing with the code (III)

This file compares a case A and a case B, where the case A is the benchmark (i.e. the solution with default parameters), and case B is the one with the updated parameter value. Notice the line that says:

```
    % modify parameters here
```

We can just put the new value we want to consider there. Then by running the file we should get the graphs of case A and B and compare them. As a first try, set `delta = 0.8` (notice that this implies a larger depreciation rate for capital, given the definition of δ in the code). Save the file as `do_compstat_delta.m`. Run the do file, what changes with respect to the benchmark case?

Playing with the code (IV)

4. Now let's change the values for the shock realizations `A_high` and `A_low`. In particular, let's have a mean-spread transformation such that the the mean is the same given the i.i.d hypothesis for the transition matrix. Set `A_high=1.25` and `A_low=0.75`, save the file as `do_compstat_shock.m` and run the code. What can you notice?

Playing with the code (V)

5. Let's now relax the assumption of i.i.d shocks. In order to do that, we need to change the transition matrix `prob` by inducing some persistence in the stochastic process. Modify the matrix in such a way that each realizations has a probability of repeating itself in the following period of 95%, i.e. the probability of a high (low) realization tomorrow given that the realization today is high (low) is 0.95. Save the file as `do_compstat_persistent.m` and run the code. What conclusions can you draw from the graphs?

Irreversible investment

VFI makes easy to incorporate inequality constraints. In order to see this, modify the code by introducing an irreversibility constraint for investment, i.e. $K_t \geq (1-\delta)K_{t-1}$.

1. Repeat the previous exercise for the model with irreversible investment. The relevant codes are `do_vfi_AM_irrinv.m` and `do_compstat_irrinv.m`.

Convex adjustment costs

It is also quite straightforward to introduce convex adjustment costs.

Modify the code by adding the adjustment costs in the form $AC(K_t, K_{t-1}) \equiv \zeta(K_t - K_{t-1})^2$, where ζ is a parameter. Choose $\zeta = .25$ to begin with, then play with it and see what happens.

Reversible vs. irreversible investment

Now let's compare the reversible investment model with the one with irreversible investment. In order to do that, we use the file `do_compare.m`. (Notice that in this case we will have to change parameters in two points of the code!)

1. Run the code as it is, and look at the graphs. What are the main differences between the reversible and irreversible investment models? Check in particular the simulated series of consumption and investment.

Exercise: Reversible vs irreversible investment (II)

2. Now let's how the depreciation rate is crucial. First change the depreciation rate of capital to `delta=0.99`. You can do this by inserting `delta=0.99` in the appropriate spaces (remember: you have two do it twice for this code!!!). Save the new file as `do_compare_delta.m` and run it. What do you observe? Can you explain it intuitively? Now set `delta=0.5` (in both the appropriate spaces!!!) and run the code again. What now?

Exercise: Reversible vs irreversible investment (III)

3. Let's see how the model with irreversible investment reacts when the variance of the shocks is reduced. Set `A_high=1.25` and `A_low=0.75`, save the file as `do_compare_shock.m` and run the code. This must be surprising for you! (Is it?)

Exercise: Reversible vs irreversible investment (IV)

4. Finally, what about persistence? Set the transition matrix `prob` such that each realization has a probability of repeating itself in the following period of 95%, i.e. the probability of a high (low) realization tomorrow given that the realization today is high (low) is 0.95. Save the file as `do_compare_persistent.m` and run the code.

Many realizations of A_t (difficult)

More than two realizations of the shocks: we need to modify the code in several points

1. we need a new variable that stores the number of realizations of the shocks (call it `num_realiz`) in the parameters' file
2. we need to provide a transition matrix `prob` which adapts to `num_realiz` (for the moment we stick to the i.i.d case for simplicity) in the parameters' file
3. we need a vector where we store the actual values for technology shock realizations in the parameters' file
4. we need to adapt the way in which we create the matrices U_j (hint: use the Kronecker product, use Matlab help for the `kron` command)
5. we need to adjust the way in which we compute the Bellman operator, in particular for the expectations' part
6. finally, we must adapt the simulations to the generic case with many possible realizations of the Markov chain

We can do the same for the model with irreversible investment. The two solution codes are `vfi_AM_general.m` and `vfi_AM_irrinv_general.m`.

Policy function iteration

- Value function iteration is slow, and for many economic problems computational speed is important
- PFI is similar approach, same convergence properties, but faster
- Also known as Howard's improvement algorithm
- Remember our problem:

$$V(K_{-1}) \equiv \max_{\{K_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t U\big(f(K_{t-1}) - K_t\big)$$

$$s.t. \quad 0 \le K_t \le f(K_{t-1}), \quad t = 0, 1, \ldots$$

The algorithm

There are 5 steps:

1. Start with a guess for the policy function $K' = g^{(0)}(K)$
2. Calculate the value associated with this policy function:

$$V^{(0)}(K) = \sum_{t=0}^{\infty} \beta^t U\left(f(K_{t-1}) - g^{(0)}(K_{t-1})\right)$$

THE ALGORITHM (CONT.)

3 For any $n \geq 0$, get a new policy function $K' = g^{(n+1)}(K)$ by solving

$$\max_{K'} \left\{ U\left(f(K) - K'\right) + \beta V^{(n)}\left(K'\right) \right\}$$

4 Calculate the value associated with this policy function:

$$V^{(n+1)}(x) = \sum_{t=0}^{\infty} \beta^t U\left(f\left(K_{t-1}\right) - g^{(n+1)}\left(K_{t-1}\right)\right)$$

5 Iterate over 3-4. Stop if $\left\| V^{(n+1)} - V^{(n)} \right\| < \varepsilon$

THE STOCHASTIC GROWTH MODEL

- The system is in one of N predetermined positions x_i, $i = 1, 2, ..., N$.
- Matrices P where $P_{ij} = Prob\{x_{t+1} = x_j \mid x_t = x_i\}$ are our choice
- We can write the Bellman equation as:

$$v(x_i) = \max_{P \in \mathscr{M}} \left\{ u(x_i) + \beta \sum_{j=1}^{N} P_{ij} v(x_j) \right\}$$

- In a more compact form:

$$v = \max_{P \in \mathscr{M}} \{u + \beta P v\} \tag{12}$$

The stochastic growth model (cont.)

We can rewrite it as as

$$v = Tv$$

where T is the operator corresponding to the RHS of (12). Define another operator $B \equiv T - I$ such that

$$Bv = \max_{P \in \mathscr{M}} \{u + \beta P v\} - v$$

and therefore we can see the policy function iteration as solving $Bv = 0$.

THE ALGORITHM REFORMULATED

The algorithm becomes:

1. Given $P^{(n)}$, get $v^{(n)}$ from

$$\left(I-\beta P^{(n)}\right)v^{(n)}=u^{(n)} \tag{13}$$

2. Find $P^{(n+1)}$ such that

$$u^{(n+1)}+\left(\beta P^{(n+1)}-I\right)v^{(n)}=Bv^{(n)} \tag{14}$$

3. Iterate until convergence

THE ALGORITHM REFORMULATED (CONT.)

First step is a linear algebra problem:

$$v^{(n)} = \left(I - \beta P^{(n)}\right)^{-1} u^{(n)}$$

Numerical implementation

- Define the two $n \times n$ matrices:

$$J_h\left(k_i, k_j\right) = \left\{ \begin{array}{lll} 1 & \textit{if} & g\left(k_i, A_h\right) = k_j \\ 0 & o/w & \end{array} \right\}$$

- Given $k' = g\left(k, A\right)$, define two vectors U_h such that:

$$U_h\left(k_i\right) = u\left(A_h f\left(k_i\right) + \left(1 - \delta\right) k_i - g\left(k_i, A_h\right)\right)$$

Numerical implementation (cont.)

- Assume the policy function is used forever
- We can associate the two vectors $V_h(k_i)$ as the values associated with starting from state (k_i, A_h):

$$\begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} + \beta \begin{bmatrix} \mathscr{P}_{11} J_1 & \mathscr{P}_{12} J_1 \\ \mathscr{P}_{21} J_2 & \mathscr{P}_{22} J_2 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix}$$

Solving by linear algebra

We can therefore solve for V_h by means of elementary linear algebra and have:

$$\begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \left[I - \beta \begin{pmatrix} \mathscr{P}_{11} J_1 & \mathscr{P}_{12} J_1 \\ \mathscr{P}_{21} J_2 & \mathscr{P}_{22} J_2 \end{pmatrix} \right]^{-1} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} \tag{15}$$

Algorithm reformulated in linear algebra

1. Given an initial feasible policy function, calculate U_h and find V_h with

$$\begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \left[I - \beta \begin{pmatrix} \mathscr{P}_{11} J_1 & \mathscr{P}_{12} J_1 \\ \mathscr{P}_{21} J_2 & \mathscr{P}_{22} J_2 \end{pmatrix} \right]^{-1} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}$$

2. Do one iteration on the Bellman equation, by using value functions found in step 1, and find a new policy function
3. Iterate until convergence

pfi_AM.m, PART 1

```
while metric > convcrit;
    contv= beta*v*prob'; % continuation value
    [tv1,tdecis1]=max(bsxfun(@plus,util1,contv(:,1)) );
    [tv2,tdecis2]=max(bsxfun(@plus,util2,contv(:,2)) );

    tdecis=[tdecis1' tdecis2'];
```

pfi_AM.m, PART 2

```
    % Build return vectors
    r1 = zeros(cs,1);
    r2 = zeros(cs,1);
    for i=1:cs
        r1(i) = util1(tdecis1(i),i);
        r2(i) = util2(tdecis2(i),i);
    end

    % create matrices Js (see lecture notes)
    g2=sparse(cs,cs);
    g1=sparse(cs,cs);
    for i=1:cs
        g1(i,tdecis1(i))=1;
        g2(i,tdecis2(i))=1;
    end
    % This is the matrix P (see lecture notes)
    trans=[ prob(1,1)*g1 prob(1,2)*g1; prob(2,1)*g2
       prob(2,2)*g2];

    % Linear algebra step to get the value function
       associated with P
    tv(:) = ((speye(2*cs) - beta.*trans))\[ r1; r2 ];
```

Exercise: Irreversible investment vs. adjustment costs II (the revenge)

Modify the PFI code to analyse the model with irreversible investment and the model with investment adjustment costs seen in the section about value function iteration. (Hint: are the modifications done for the VFI code enough?)

NUMERICAL METHODS

Numerical methods

- The codes presented for VFI and PFI have many drawbacks!!!
- Brief introduction to basic numerical tasks
- Not exhaustive
- References:
  1. **Miranda and Fackler (2002)**: good book, simple introduction to numerical methods with Matlab, comes with a (very useful) toolbox (we use it in the projection methods codes)
  2. **Judd (1998)**: THE BIBLE

CompEcon Toolbox and LIBM

- CompEcon Toolbox: useful numerical routines for the economist
- LIBM: another library with efficiently coded routines (created by Michael Reiter)
- Installation: just add them to the Matlab path (if you have some troubles, please let me know)

Calculating integrals

$$\int_I f(x)w(x)dx \approx \sum_{i=1}^{n} f(x_i)w_i \tag{16}$$

- Different ways to choose nodes x_i and the weights w_i
- Newton-Cotes methods: approximate f between nodes using low order polynomials, then sum the integrals of those polynomials (which are easy to calculate)
    - trapezoid rule: piecewise linear interpolants
    - Simpson's rule: piecewise quadratic interpolants
- Gaussian quadrature: chooses nodes and weights by matching some moments of the distribution
- Monte Carlo methods: randomly choose nodes
- Replicating a continuous process with one with discrete support, for example Tauchen's method

CALCULATING INTEGRALS

NEWTON-COATES METHODS

- Newton-Cotes methods: approximate f between nodes using low order polynomials, then sum the integrals of those polynomials (which are easy to calculate)
  - trapezoid rule: piecewise linear interpolants

    ```
    [x,w] = qnwtrap(n,a,b);
    ```

  - Simpson's rule: piecewise quadratic interpolants

    ```
    [x,w] = qnwsimp(n,a,b);
    ```

CALCULATING INTEGRALS

GAUSSIAN QUADRATURE

- Gaussian quadrature: chooses nodes and weights by matching some moments of the distribution
- Gaussian quadrature of order n chooses n quadrature nodes that satisfy the $2n$ moment matching conditions:

$$\int_I x^k w(x) dx = \sum_{i=1}^{n} w_i x_i^k, k = 0, \ldots, 2n-1$$

- If $w(x) = 1$: Gauss-Legendre quadrature

```
[x,w] = qnwlege(n,a,b);
integral = w'*log(x)
```

Calculating integrals

Exercise 3.1: Integrals

1. Calculate the integral of e^{-x} on the interval $[-1,1]$ using the trapezoid rule.
2. Calculate the integral of $|x|^{\frac{1}{2}}$ on the interval $[-1,1]$ using Simpson's rule.
3. Calculate the integral of $(1+25x^2)^{-1}$ on the interval $[-1,1]$ using Gauss-Legendre method.

Calculating integrals

Exercise 3.2: Compare the methods

Write a short script that compares the three methods. Calculate the integral of $e^{-x}, |x|^{\frac{1}{2}}$ and $(1+25x^2)^{-1}$ on the interval $[-1,1]$ by hand (these are pretty easy to calculate). Then use CompEcon commands and calculate the integrals with 10, 20, 30, 100 nodes for each method. Save the results in a matrix and then compare the accuracy, i.e. the difference between the numerical integral and the correct integral calculated by hand. What can you notice?

CALCULATING INTEGRALS

EXERCISE 3.2: COMPARE THE METHODS

REMINDER:

$\int_{-1}^{1} e^{-x} dx = (-e^{-x})\Big|_{-1}^{1}$

$\int_{-1}^{1} |x|^{\frac{1}{2}}\, dx = \int_{-1}^{0} |x|^{\frac{1}{2}}\, dx + \int_{0}^{1} |x|^{\frac{1}{2}}\, dx = \left(\frac{2}{3}(-x)^{\frac{3}{2}}\right)\Big|_{-1}^{0} + \left(\frac{2}{3}x^{\frac{3}{2}}\right)\Big|_{0}^{1}$

and $\int_{-1}^{1} (1+25x^2)^{-1} dx = \left(\frac{1}{5}\arctan 5x\right)\Big|_{-1}^{1}$

Calculating integrals

Gaussian quadrature for normal distribution

- `qnwnorm` calculates the nodes and weights for multidimensional normal distributions

```
[x,w] = qnwnorm(n,mu,var);
```

  - `n`: number of nodes in each dimension
  - `mu`: mean vector
  - `var`: variance-covariance matrix.

- Calculate the expectations of e^{-x}, with $x \sim \mathbf{N}(0,1)$, and 30 nodes

```
[x,w] = qnwnorm(30,0,1);
expectations = w'*exp(−x);
```

Calculating integrals

Normal distribution

Exercise 1 Univariate normal

Calculate the expected value of $x^{-\sigma}$, for $\sigma = \{0.5, 1, 2, 10\}$ and $x \sim \mathbf{N}(0,1)$. Use 30 nodes.

Exercise 2 Multivariate normal

Calculate the expected value of $e^{x_1+x_2}$, with x_1 and x_3 jointly normal with $Ex_1 = 3$, $Ex_2 = 4$, $Var(x_1) = 2$, $Var(x_2) = 4$, $Cov(x_1, x_2) = -1$. Use 10 nodes in the x_1 direction and 15 nodes in the x_2 direction.

Nonlinear equations

Bisection method

$$f(x) = 0$$

Bisection methods: continuous real-valued function on a real interval $[a, b]$

- Start with two points a_1, b_1 such that $f(a_1) < 0, f(b_1) > 0$
- Choose a new point $x_2 \in [a_1, b_1]$ and calculate the sign of $f(x_2)$. If positive, now restrict your search to $[a_1, x_2]$, if negative to $[x_2, b_1]$.

```
f = inline('x^3 - 5');
x = bisect(f,1,2);
```

Exercise 3 Bisection

1. Find the roots of $e^{-x^2} - \cos(x)$ over the interval $[-4, 6]$.
2. Plot the function over the interval. What can you observe?

NONLINEAR EQUATIONS

FUNCTION ITERATION

$$f(x) = 0$$

Function iteration: $f : R^n \to R^n$

- Write the equation as $x = g(x)$
- Guess $x^{(0)}$, calculate $x^{(1)} = g(x^{(0)})$
- Iterate over the recursion $x^{(n+1)} = g(x^{(n)})$ until convergence

```
g = inline('x^0.5');
x = fixpoint(g,0.4);
```

Exercise 4 Function iteration

Find the roots of $e^{-x^2} - \cos(x)$ over the interval $[-4, 6]$. (Hint: first you have to transform the equation from the form $f(x) = 0$ into $x = x - f(x)$).

Nonlinear equations

Newton's method

$$f(x) = 0$$

Newton's method: $f : R^n \to R^n$

- First order Taylor approximation around $x^{(k)}$:
$$f(x) \approx f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}) = 0$$
- Guess $x^{(0)}$, calculate $x^{(1)} = x^{(0)} - [f'(x^{(0)})]^{-1} f(x^{(0)})$
- Iterate over the recursion $x^{(n+1)} = x^{(n)} - [f'(x^{(n)})]^{-1} f(x^{(n)})$ until convergence
- In CompEcon: `newton`

Nonlinear equations

Newton's method

The syntax is slightly more complicated. We need to create a function file in the form:

```
[fval,fjac]=f(x,optional additional parameters)
```

where `fjac` is the Jacobian of the function.

Then the command is

```
          x = newton(f,x0)
```

where `x0` is an initial guess.

Exercise 5 Newton method

Find the roots of $e^{-x^2} - \cos(x)$ using the Newton method.

Nonlinear equations

Quasi-Newton's methods

$$f(x) = 0$$

Like Newton's method, but use a computable Jacobian (not the exact one)

- Very popular (and my favourite): **Broyden's method**
- Guess $x^{(0)}$, and $A^{(0)}$, calculate $x^{(1)} = x^{(0)} - [A^{(0)}]^{-1} f(x^{(0)})$
- Update A with the smallest possible change that satisfies the *secant condition*: $f(x^{(n+1)}) - f(x^{(n)}) = A^{(n+1)}(x^{(n+1)} - x^{(n)})$ (for speed: update the inverse of the Jacobian)
- Iterate over the recursion $x^{(n+1)} = x^{(n)} - [A^{(n)}]^{-1} f(x^{(n)})$ until convergence

Nonlinear equations

`broydn.m`

Included in LIBM, very efficient

```
[x, flag] = broydn(fname,xold,TOLF,iadmat,iprint,
    varargin);
```

Exercise 6 Broyden method

1. Find the roots of $e^{-x^2} - \cos(x)$ using the Broyden's method. Play with the initial guess to see how the solution changes.

Exercise 7 Broyden method

1. Imagine you have a two-periods economy with an initial amount of savings s_0. Therefore, the equations that describe the intertemporal decision of the agent are given by:

$$c_1^{-\sigma} = \beta(1+r)c_2^{-\sigma}$$
$$c_1 + \frac{c_2}{1+r} = y_1 + \frac{y_2}{1+r} + s_0$$

   where $r = 0.05$, $\sigma = 2$, $\beta = .99$, and $y_1 = y_2 = 1$. Solve for the optimal allocation for different values of the initial savings. (Hint: write a function that takes as second input a vector of initial savings, and solve these equations in a vectorized way.

DERIVATIVE-FREE METHODS

- **Golden search**: similar to bisection, find the maximum on interval $[a,b]$
- Start with two points x_1, x_2 in the interval, such that $x_1 < x_2$, and evaluate $f(x_i)$
- If $f(x_1) \geq f(x_2)$, then the new interval is $[a, x_2]$, otherwise is $[x_1, b]$
- Iterate until convergence
- Use `golden`, from CompEcon

```
[x, fval] = golden(fname,a,b,varargin)
```

- Use `goldsvec.m`, in the LIBM library (from R^n to R^n)

```
 [x, fval] = goldsvec(fname,a,b,varargin)
```

Exercise 8 Golden Search

1. Use the `golden` command to maximize the MATLAB function `humps` on the interval $[-10, 10]$
2. Use the `golden` command to maximize the MATLAB function `humps` on the interval $0.2, 2]$. Comment.

Exercise 9 Golden Search

1. *(Difficult)* Imagine you have a two-periods economy with an initial amount of savings s_0, per-period CRRA utility function $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$, and discount factor β. The intertemporal budget constraint of the agent is:

$$c_1 + \frac{c_2}{1+r} = y_1 + \frac{y_2}{1+r} + s_0$$

where $r = 0.05$, $\sigma = 2$, $\beta = .99$, and $y_1 = y_2 = 1$. Solve for the optimal allocation for different values of the initial savings using the `goldsvec` routine.

Optimization

Newton-Raphson method

- Similar to Newton's methods for nonlinear equations
- Take second order Taylor expansion of the maximand:

  $f(x) \approx f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T f''(x^{(k)})(x - x^{(k)})$
- FOCs are: $f'(x^{(k)}) + f''(x^{(k)})(x - x^{(k)}) = 0$
- Hence: $x^{(n+1)} = x^{(n)} - [f''(x^{(n)})]^{-1} f'(x^{(n)})$
- Iterate until convergence
- (and guess what: there are of course Quasi-Newton methods that use a computable Hessian)
- Matlab routines in the Optimization Toolbox (e.g. `fmincon`)

Optimization

"I-am-in-deep-trouble" problems

Once in a while, you will face a very irregular optimization problem

- Global methods: line search, pattern search
- Genetic algorithms
- Simulated annealing
- Swarm search optimization
- Ad-hoc techniques (problem-specific)
- Most of the time: good for finding initial guesses, then use standard methods

CURSE OF DIMENSIONALITY

- VFI relies on setting up a grid for the state variables
- Need many points to get a good approximation,
- if $N =$ n. states, and discretize each state in m grid points $\Rightarrow$ your value function needs to be evaluated in m^N points
- Memory-intensive task! This is called the *curse of dimensionality*.
- Several ways to reduce it:
  - Interpolation of the value function and/or choice variables
  - Smart choice of gridpoints
- Projection methods need substantially less grid points than the other techniques

Parallelization

- VFI is an embarrassingly parallelizable algorithm, i.e. the maximization step can be performed gridpoint by gridpoint
- Each single (i.e. one grid point) maximization problem can be sent to a different processor.
- Helpful in problems with high-dimensional state spaces

PROJECTION METHODS

PROJECTION METHODS

- How do we solve functional equations in general?
- **Projection methods**
- Widespread use in macroeconomic theory, no limits to applications
- References: Miranda and Fackler (2002), Judd (1998), Judd (1992)

Main idea

We want to solve a functional equation of the type:

$$\mathscr{N}\left(g\left(x\right)\right) = 0 \tag{17}$$

- We can get only an approximation of our solution $g\left(x\right)$

The algorithm

1. Approximate the function $g(x)$ as a combination of some simple functions $\phi_i(x)$: $\widehat{g}(x) = \sum_{i=1}^{n} a_i \phi_i(x)$
2. Calculate the residual function, i.e. an approximated version of the functional equation: $R(x; \mathbf{a}) \equiv \left(\widehat{\mathscr{N}}(\widehat{g})\right)(x)$
3. Calculate the projections with some functions $p_i(x)$: $P_i(\cdot) \equiv \langle R(\cdot; \mathbf{a}), p_i(\cdot)\rangle, i = 1, ..., n$
4. Find the vector of coefficients $\mathbf{a}$ that solves $P_i(\cdot) = 0$, $i = 1, ..., n$.

First step

Choose:

- How we approximate the solution:
  - Usually: a linear combination of some simple functions $\phi_i(x)$ (we call them **basis functions**), like polynomials (more details later)
- An appropriate concept of distance in order to measure the accuracy of our calculated solution.

Second step

Choose:

- a degree of approximation n, i.e. how many basis functions we want to use.
- a computable approximation $\widehat{\mathcal{N}}$ for $\mathcal{N}$, if the exact operator is not directly computable
- functions $p_i, i = 1, ..., n$ that we will use to calculate the projections (many times we use the basis functions)
- $\Rightarrow$ our approximation is $\widehat{g}(x) = \sum_{i=1}^{n} a_i \phi_i(x)$ for any x
- Any solution can be summarized by a vector of coefficients $\mathbf{a}$

Third step

- Compute numerically the approximated policy function $\widehat{g}(x) = \sum_{i=1}^{n} a_i \phi_i(x)$ for a particular guess of $\mathbf{a}$
- Compute the so called residual function

$$R(x;\mathbf{a}) \equiv \left(\widehat{\mathscr{N}}(\widehat{g})\right)(x)$$

The first guess can be important: it is crucial to start with a good guess

Fourth step

- Calculate the projections

$$P_i(\cdot) \equiv \langle R(\cdot;\mathbf{a}), p_i(\cdot)\rangle, i = 1, ..., n \tag{18}$$

- the typical choice for the inner product used in the calculation of the projections is, given a weighting function $w(x)$:

$$\langle f(x), h(x)\rangle \equiv \int f(x)h(x)w(x)dx$$

FIFTH STEP

We look for **a** that makes $P_i(\cdot)$ equal to zero

We iterate over step 3 and 4 to get a vector of coefficients **a** that sets the projections (18) to zero

CHOICE OF BASIS FUNCTIONS

- Ordinary polynomials $1, x, x^2, x^3, \ldots$ However, problematic
- Two broad categories: spectral methods and finite element methods.
  - Spectral methods: basis functions are almost everywhere nonzero, continuously differentiable as many time as needed, imposing smoothness on the approximated function (which sometimes is not a desirable feature)
  - Finite element methods: basis functions are zero except for a small support

Chebychev polynomials

Spectral method, defined as

$$T_n(x) \equiv \cos(n \arccos x), \quad x \in [-1, 1]$$

and it is possible to generate them with the following recursive law:

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$
$$T_0(x) = 1, \quad T_1(x) = x$$

They satisfy the following orthogonality condition:

$$\int_{-1}^{1} T_i(x)\, T_j(x) \left(1 - x^2\right)^{-\frac{1}{2}} dx = 0, \quad i \neq j$$

CHEBYCHEV POLYNOMIALS (CONT.)

Let $z_l^n \equiv \cos\left(\frac{(2l-1)\pi}{2n}\right), l = 1, ..., n$ be the zeroes of T_n:

$$\Sigma_{l=1}^n T_i\left(z_l^n\right) T_j\left(z_l^n\right) = 0, \quad i \neq j$$

therefore if we use z_l^n as gridpoints, we can simplify computation and convergence is faster

Tent functions

- Finite element method, aka piecewise linear basis
- Take an approximation with support in $[a,b]$ and be $h=(a-b)/n$. Then, for $i=0,1,\ldots,n$:

$$\phi_i(x)=\begin{cases} 0 & a\le x\le a+(i-1)h \\ (x-(a+(i-1)h))/h & a+(i-1)h\le x\le a+ih \\ 1-(x-(a+(i-1)h))/h & a+ih\le x\le a+(i+1)h \\ 0 & a+(i+1)h\le x\le b \end{cases}$$

- A generalization of those bases is piecewise degree k polynomials, like Hermite polynomials and cubic splines.

Multidimensional state space

- Use tensor products: if $\{\phi_i(x)\}_{i=1}^{\infty}$ is the basis for a function in one variable, $\{\phi_i(x)\phi_j(y)\}_{i,j=1}^{\infty}$ for functions of two variables, and so on.
- Main problem: number of elements increases exponentially. Various ways to overcome this problem: one is to use complete polynomials of order k:

$$\mathscr{P}_k \equiv \left\{ x_1^{i_1} \cdot \ldots \cdot x_n^{i_n} \middle| \sum_{l=1}^{n} i_l \leq k, 0 \leq i_1, \ldots, i_n \right\}$$

CHOICE OF PROJECTIONS

- The choice of functions p_i characterizes different approaches
- **Least squares approach**:

$$\min_{\mathbf{a}} \langle R(x;\mathbf{a}), R(x;\mathbf{a}) \rangle$$

- **Galerkin method**:

$$P_i(\mathbf{a}) \equiv \langle R(x;\mathbf{a}), \phi_i(x) \rangle = 0, \quad i = 1, \ldots, n$$

CHOICE OF PROJECTIONS (CONT.)

- **Method of moments** uses the first n polynomials:

$$P_i(\mathbf{a}) \equiv \langle R(x;\mathbf{a}), x^{i-1} \rangle = 0, \quad i = 1, \ldots, n$$

- **Subdomain method** solves

$$P_i(\mathbf{a}) \equiv \langle R(x;\mathbf{a}), \mathbf{I}_{D_i} \rangle = 0, \quad i = 1, \ldots, n$$

where $\{D_i\}$ is a sequence of intervals covering the entire domain of the function, and $\mathbf{I}_{D_i}$ is the indicator function for D_i.

CHOICE OF PROJECTIONS (CONT.)

- **Collocation method** chooses n points $\{x_i\}_{i=1}^{n}$ in the domain and solves

$$R(x_i;\mathbf{a}) = 0, \quad i = 1,\ldots,n$$

- This is equivalent to solve

$$P_i(\mathbf{a}) \equiv \langle R(x;\mathbf{a}), \delta(x - x_i)\rangle = 0, \quad i = 1,\ldots,n$$

where $\delta(\cdot)$ is the Dirac delta function that is equal to zero everywhere but in zero, where it takes value 1

SPEED CONCERNS

- Main computational burden is calculating projections
- Collocation is very fast; other methods would require the computation of an integral
- **Orthogonal collocation** chooses collocation nodes as the zeroes of the basis function: even faster

COMPECON FOR PROJECTION METHODS

GENERATING FUNCTIONAL SPACES WITH `fundefn`

`fundefn` creates a Matlab structured variable that characterizes the functional space of the basis functions chosen by the programmer. The syntax is:

```
fspace = fundefn(bastype,n,a,b,order);
```

- `bastype`: type of basis function
  - Chebichev polynomials (`'cheb'`)
  - splines (`'spli'`)
  - linear spline basis with finite difference derivatives (`'lin'`)

COMPECON FOR PROJECTION METHODS

GENERATING FUNCTIONAL SPACES WITH `fundefn`

`fundefn` creates a Matlab structured variable that characterizes the functional space of the basis functions chosen by the programmer. The syntax is:

```
fspace = fundefn(bastype,n,a,b,order);
```

- `n`: vector indicating the degree of approximation along each dimension
- `a`, `b`: identify the left and right endpoints for interpolation intervals for each dimension
- `order`: spline order, default is cubic splines

CompEcon for projection methods

Generating functional spaces with `fundefn`

Example: generating a functional basis space for 5th degree Chebychev polynomials for a univariate function in the interval $[-5,6]$:

```
fspace = fundefn('cheb',5,-5,6);
```

Generating a cubic spline space in two dimensions, with 10 basis functions in the first dimension and 8 in the second on the interval $\{(x_1,x_2): -5 \leq x_1 \leq 6, 2 \leq x_2 \leq 9\}$:

```
fspace = fundefn('spli',[10 8],[-5 2], [6 9]);
```

CompEcon for projection methods

Evaluating functions with `funeval`

- We want to evaluate $\widehat{g}(x) = \sum_{i=1}^{n} c_i \phi_i(x)$, given coefficients $\mathbf{c}$ and basis functions $\{\phi_i(x)\}_{i=1}^{n}$
- Define a set of points `x`, a vector of coefficients `c` and a functional space `fspace`:

  ```
  y = funeval(c,fspace,x);
  ```
- `x` is a $m \times k$ matrix, where m is the number of points, and k is the dimensionality of the space.

CompEcon for projection methods

Other functions: `funbas` and `funnode`

- `funbas` returns the value of the basis functions calculated in a particular set of points `x`

  ```
  Basis = funbas(fspace,x);
  ```

- Useful for evaluating a function at `x`, but with different `c` (equivalent to `funeval`):

  ```
  Basis = funbas(fspace,x);
  y = Basis*c;
  ```

COMPECON FOR PROJECTION METHODS

OTHER FUNCTIONS: `funbas` AND `funnode`

- `funnode` computes standard nodes
- Example: if Chebychev polynomials are used, we get the Chebychev's zeros with

  ```
  x = funnode(fspace);
  ```

THE STOCHASTIC GROWTH MODEL SOLVED WITH COLLOCATION

- We will solve the SGM with collocation over the first order conditions
- Notice: we assume continuous support for TFP shocks, can be persistent

$$u'(c_t) = \beta E_t \left[u'(c_{t+1}) \left(\alpha A_{t+1} k_{t+1}^{\alpha-1} + 1 - \delta \right) \right]$$

$$u'(\max(0, Ak^{\alpha} + (1-\delta)k - g(K,A))) = \beta E \Big[u'(\max(0, A'(g(k,A))^{\alpha} + (1-\delta)g(k,A) - g(g(K,A),A'))) \left(\alpha A'(g(k,A))^{\alpha-1} + 1 - \delta \right) \Big]$$

OUR OPERATOR $\mathscr{N}(g(x))$

$$\mathscr{N}(g(k,A)) \equiv u'(\max(0, Ak^{\alpha} + (1-\delta)k)) - \beta E\left[u'(\max(0, A'(g(k,A))^{\alpha} + \right.$$
$$\left.(1-\delta)g(k,A)))\left(\alpha A'(g(k,A))^{\alpha-1} + 1 - \delta\right)\right]$$

Preliminaries: installing libraries

- LIBM: add it to your Matlab path
- CompEcon: add it to your Matlab path with the option "Add with subfolders"
- As a test: launch the file solveSGM.m, and see if you receive an error message (you shouldn’t)

THE CODE

- Set parameters values and grid for states (in `solveSGM.m`)
- Generates the functional space for the basis functions, and a "good" initial guess for the coefficients (in `solveSGM.m`)
- Find coefficients that put the residual function (written in the function `focsSGM.m`) as close to zero as possible (this is in fact done by solving nonlinear equations with Broyden's method, by the file `mainSGM.m`)
- Test solution accuracy (in `mainSGM.m`) and then simulate the model (in `solveSGM.m`)

solveSGM.m, Part 1

```
global alpha betta rho sig sigma delta sigeps ;
global nQuadr QuadrWeights QuadrPoints;
global  RoundAppr rounds_approx  ;

%% PARAMETERS:
alpha =.4;        % production function coefficient
delta = .1;       % depreciation rate for capital
betta =  .95;       % discount factor
sig = 1; % .5;%    % CRRA utility (c^(1-sig))/(1-sig)
sigma = .05;       % S.D. of productivity shock
rho = 0;%;  .9; %  % persistence of productivity shock

%% create a grid for capital
kstar = (1/(alpha*betta) - ...
    (1-delta)/alpha )^(1/(alpha-1)); %det. steady state
k_min = .5*kstar;
k_max = 2*kstar;

%% parameters for quadrature
nQuadr = 50; % 100;% %number of quadrature points;
% we choose nQuadr high to get smoothness;
[QuadrPoints,QuadrWeights] = qnwnorm(nQuadr,0,sigma^2);
```

solveSGM.m, Part 2

```
%% Range for shock
sigeps = sigma/sqrt(1-rho^2);
% Range for shock:
A_max = 3*sigeps;
A_min =  -3*sigeps;
%% Parameters for the collocation algorithm
rounds_approx = 2; % number of rounds of approximation
Order_vector = [5 10; 5 10];%grid points for each round
ntest = 100; %gridpoints for testing for each dimension

%% Approximation type for CompEcon
    % approxtype = 'lin';  % piecewise linear
        approxtype = 'cheb'; % chebychev polynomials
%     approxtype = 'spli'; % splines
    splineorder = []; % splines' order

%% parameters for simulations
number_series =1;    % number of series
periods_simulation =  100; % n. periods for simulation
k0 = k_min.*ones(number_series,1);

%% Run main file
mainSGM; % solves the model
```

mainSGM.m, PART 1

```
for oo = 1: rounds_approx
    RoundAppr = oo;     % round of approximation

    % Range on which we approximate the solution:
    LowerBound = [k_min A_min ];
    UpperBound = [k_max A_max];

    % Approximation order
    Order = Order_vector(:,oo);

    % for reference, need this for guess after round 1
    if oo >= 2
        fspace_old = fspace;
    end

    disp(' '); disp(' ' );
    disp(sprintf('RoundAppr %d, # gridpoints = [%d %d]'
        ,...
        RoundAppr, Order(1), Order(2)));
    disp(' ');
```

mainSGM.m, PART 2

```
    % generate basis function space: we can choose
    % among chebychev polynomials, splines of
    %  different orders and piecewise linear functions
  if(strcmp(approxtype,'spli'))
        fspace = fundefn(approxtype,Order,LowerBound,
            UpperBound,splineorder);
    else
        fspace = fundefn(approxtype,Order,LowerBound,
            UpperBound,[]);
    end;

    % the following commands create gridpoints
    nodes = funnode(fspace);
    Grid = gridmake(nodes);

    % Set initial conditions
    if (RoundAppr == 1)
        knext = Grid(:,1);
    else    % if we are at second approx round, we use
        the solution of the first round
        % as initial conditions on the new larger grid
        knext = funeval(park, fspace_old, Grid);
    end;
```

mainSGM.m, PART 3

```
    % generate basis functions Basis at Grid :
    Basis = funbas(fspace,Grid);

    % set initial value for parameters of the approximation
    park =  Basis\knext;

    % solve FOCs with Broyden method for nonlinear equations
    [park,info] =  broydn('focsSGM',park,1e-8,0,1,Grid,fspace);
    disp(sprintf(' info = %d',info)); % if info=0, everything went fine, o/w the Broyden algorithm didn't converge
    disp(sprintf('    '));

end;
```

focsSGM.m, PART 1

```
%% FOCS for the stochastic growth model

function equ = focsSGM(park, Grid,fspace);

global alpha betta sig rho delta A_bar
% global LowerBound UpperBound
global nQuadr QuadrWeights QuadrPoints

LowerBound = fspace.a;
UpperBound = fspace.b;

%rename grid
k = Grid(:,1);
A = Grid(:,2);

% evaluate policy functions
knext = funeval(park,fspace, Grid);
fofk = exp(A).*(k.^alpha) + (1-delta).*k;
% c = fofk - knext;
c = max(fofk - knext, zeros(length(Grid),1));
```

focsSGM.m, PART 2

```matlab
n = length(k);
% generate nQuadr replications of the Grid, one for each realization of shock:
Grid_knext = kron(knext,ones(nQuadr,1));
% Exp. value of next period A, corresponding to Grid:
ExpA =  rho*A;
% all realizations of next A:
GridANext = kron(ExpA,ones(nQuadr,1)) + ...
    kron(ones(n,1),QuadrPoints);
% truncate it to state space:
GridANext = min(max(GridANext,LowerBound(2)),UpperBound(2));
GridNext = [Grid_knext GridANext];
% calculate variables at t+1
knextnext = funeval(park,fspace, GridNext);
fofknext = exp(GridANext).*(Grid_knext.^alpha) + (1-delta).*Grid_knext;
 cnext = max(fofknext - knextnext, zeros(length(Grid_knext),1));
mucnext = muc(cnext);
mpknext = mpk(GridNext);
% calculate expectations with quadrature
exp_mucnext = (QuadrWeights'*reshape(mpknext.*mucnext,nQuadr,n))';
```

focsSGM.m, PART 3

```
% equation to be solved: Euler equation
equu = (muc(c) - betta.*exp_mucnext)./muc(c);

% avoid strange solutions
if (any(cnext<0))  || (any(c<0)) || (any(knext<0))  ||
    (any(knextnext<0))
     equ(1) = 1e100;
end;
```

Exercise

Solving the RBC model with collocation over the Bellman equation

Create a code that solves the Belmann equation of the RBC model with collocation, starting from the code used to solve it with FOCs.

- First, write down the different components of your code (which files you will have to create)
- Choose a way to perform the maximization step in the Bellman operator. (Hint: can you do it with full discretization of the choice variable?)

Tricks and difficulties

Convergence

- There is no theorem guaranteeing convergence
- The researcher has to "guide" the convergence process with ad-hoc solutions
  - Homotopy methods: start from the solution of a simpler model and iteratively modify the $\mathscr{N}$ operator to match the new, more complicated model
  - Bounds for the first iterations
  - ...

Tricks and difficulties

Dependence on the initial guess

- Most of the times, the solution we obtain depends on the initial guess
- Important to get a good guess
- Check that we always converge to the same solution even if we start from different initial guesses

Tricks and difficulties

Ill-conditioning

- When using nonlinear solvers: Jacobians/Hessians must be computed
- Ill-conditioning is a very frequent problem, no easy fixes
- Smart choice of gridpoints and more rounds of approximation usually help

Tricks and difficulties

Speed considerations and models with large state spaces

- In high dimensional models, speed is a concern
- Recent work shows that projection methods work well even in these models
- Malin, Krueger and Kubler (2011): use Smoliak algorithm to smartly choose gridpoints
- Judd, Maliar and Maliar (2010): simulated ergodic grid methods
  - Guess a solution, simulate the solution many times
  - From the simulated points, find k clusters, and use the centroids of each cluster as your grid points.
  - Solve with collocation on the k grid points.
  - Iterate until convergence

1
101
10001
KEEP
CALM
AND
CODE
ON