# Macroeconomics

## Lecture 9: dynamic programming methods, part seven

Chris Edmond

1st Semester 2019

# This class

- Practical stochastic dynamic programming
  - numerical integration to help compute expectations
  - using collocation to solve the stochastic optimal growth model

# Numerical integration (quadrature)

- Consider integral of a function $f(x)$ against weights $w(x)$

$$\int f(x) w(x)\, dx$$

- Often not possible to calculate the integral exactly

- Can approximate the integral value by choosing an appropriate set of *quadrature nodes* $x_i$ and *weights* $w_i$ so that

$$\int f(x) w(x)\, dx \approx \sum_{i=1}^{n} f(x_i)\, w_i$$

- Various procedures for choosing nodes $x_i$ and weights $w_i$ (Newton-Cotes, Gaussian, Monte Carlo, etc)

# Gaussian quadrature

- Choose nodes $x_i$ and weights $w_i$ to satisfy $2n$ '*moment conditions*'

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

(2$n$ nonlinear equations in 2$n$ unknowns, nontrivial but standard routines exist)

- If $x$ is a continuous random variable with PDF $w(x)$ then Gaussian quadrature discretizes $x$, replacing it with $n$ discrete points $x_i$ and a PMF $w_i$ on those discrete points

- The discretized version approximates the continuous version in the sense that the first $2n$ moments are the same

# Gaussian quadrature, 3-point example

- Suppose $w(x) = \phi(x)$, the standard normal density
- Choose 3 nodes and 3 weights to satisfy 6 moments

$$\mathbb{E}[x^0] = 1, \quad \mathbb{E}[x^1] = 0, \quad \mathbb{E}[x^2] = 1,$$
$$\mathbb{E}[x^3] = 0, \quad \mathbb{E}[x^4] = 3, \quad \mathbb{E}[x^5] = 0$$

- Solution to system of 6 equations in 6 unknowns is

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -\sqrt{3} \\ 0 \\ +\sqrt{3} \end{pmatrix}, \qquad \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} 1/6 \\ 2/3 \\ 1/6 \end{pmatrix}$$

# Normal example

- Using CompEcon tools

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

- Then moments

$$\mathbb{E}[x] = \sum_{i=1}^{n} x_i w_i$$

$$\text{Var}[x] = \sum_{i=1}^{n} x_i^2 w_i - \mathbb{E}[x]^2$$

# Lognormal example

- Similarly

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

# Using quadrature: AR1 example

- Suppose we want to compute

$$\mathbb{E}[\, f(z') \,|\, z]$$

for some function $f(\cdot)$ that we can evaluate

- And suppose for given $z$ that

$$z' = \rho z + \varepsilon, \qquad \varepsilon \sim \text{IID } N(\mu, \sigma^2)$$

# Using quadrature: AR1 example

- The exact integral is

$$\mathbb{E}[\, f(z') \,|\, z] = \int f(\rho z + \varepsilon)\phi(\varepsilon)\, d\varepsilon$$

(where again $\phi(\cdot)$ denotes the standard normal density)

- We approximate this with the numerical integral

$$\mathbb{E}[\, f(z') \,|\, z] \approx \sum_{i=1}^{n} f\,(\rho z + \varepsilon_i\,)\, w_i$$

using the quadrature nodes and weights

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

# Quadrature and collocation

- Suppose we want to compute

$$\mathbb{E}[\, v(z') \mid z]$$

where $v(\cdot)$ is approximated by basis functions

$$v(z') \approx \sum_{j=1}^{m} a_j \varphi_j(z')$$

- Then using quadrature

$$\mathbb{E}[\, v(z') \mid z] \approx \sum_{i=1}^{n} \sum_{j=1}^{m} a_j \, \varphi_j(\rho z + \varepsilon_i) \, w_i$$

# Quadrature and collocation

- To calculate this sum, we need to evaluate terms like

$$\varphi_j(\rho z + \varepsilon_i)$$

- Do this using the CompEcon tools, for example

```
zprime = rho*z+epsilon(i)
```

then

```
funeval(a,fspace,zprime)
```

# Stochastic growth example

- Let's solve the Bellman equation

$$v(k,z) = \max_{k'} \left\{ u(f(k,z) - k') + \beta \mathbb{E}[v(k',z') \,|\, z] \right\}$$

with the usual specification

$$f(k,z) = zk^{\alpha} + (1-\delta)k$$

$$u(c) = \frac{c^{1-\sigma} - 1}{1-\sigma}$$

- And let's suppose that $z'$ is an AR(1) in logs

$$\log z' = \rho_z \log z + \varepsilon, \qquad \varepsilon \sim \text{IID } N(0, \sigma_z^2)$$

# Stochastic growth example

Uses Matlab files in "*stochastic_growth_example.zip*" in LMS

```
%%%%% economic parameters

alpha = 1/3;          %% capital's share in production function
beta  = 0.95;         %% time discount factor
delta = 0.05;         %% depreciation rate
sigma = 1;            %% CRRA (=1/IES)

rhoz  = 0.95;         %% AR1 coefficient, productivity shocks
sigz  = 0.1;          %% innovation std dev, productivity shocks
```

# Productivity shocks

```
%%%%% grid for productivity shocks

nz          = 29;        %% number of breakpoints for z grid
nez         = 15;        %% number of nodes for quadrature (shocks)

% quadrature nodes and weights
[ez, wz] = qnwunif(nez,1e-9,1-1e-9);
ez          = sigz*norminv(ez,0,1);

zmin        = exp(-4*sqrt(1/(1-rhoz^2))*sigz);
zmax        = exp( 4*sqrt(1/(1-rhoz^2))*sigz);
zgrid       = exp(nodeunif(nz, log(zmin), log(zmax)));
```

Inverting the uniform nodes gives a bit more control of the tails

# Capital stock

```
%%%%% grid for capital stock

nk        = 99;        %% number of breakpoints for k grid
curv      = 0.25;      %% (curv = 0 log-spaced, curv = 1 linear)

kmin      = 1e-3;
kmax      = (zmax/delta)^(1/(1-alpha));
kgrid     = nodeunif(nk, kmin.^curv, kmax.^curv).^(1/curv);
```

# Function space for approximations

```
%%%%% setup state space using CompEcon tools

fspace     = fundef({'spli', kgrid},...
                    {'spli', zgrid}); % function space structure

grid       = funnode(fspace);% nodes where we solve the problem
Phi        = funbas(fspace); % matrix of collocation basis vectors
                             % Phi_{ij} = phi_j(k_i)

kgrid      = grid{1}; % extra 2 points for 3rd-order spline
zgrid      = grid{2}; % extra 2 points for 3rd-order spline

kmin       = kgrid(1);
kmax       = kgrid(end);

zmin       = zgrid(1);
zmax       = zgrid(end);
```

# Matrix with all combinations of states

```
%%%%% form collection of states

s          = gridmake(grid); % ns-by-2 matrix where ns=nk*nz
ns         = size(s,1);

k          = s(:,1);
z          = s(:,2);
```

# Matrix with all combinations of states

- For example

```
S = gridmake([1;2;3],[4;5])
```

- Gives the matrix

$$\mathtt{S} = \begin{bmatrix} 1 & 4 \\ 2 & 4 \\ 3 & 4 \\ 1 & 5 \\ 2 & 5 \\ 3 & 5 \end{bmatrix}$$

# Initial guess at collocation coefficients

```
%%%%% initial guess at collocation coefficients "a"

c = alpha*beta*z.*k.^alpha; % guess for consumption policy
v = log(c)/(1-beta);        % guess for value function

a = Phi\v;                  % implied collocation coefficients
```

# Solve Bellman equation by collocation

```
%%%%% solve Bellman equation

for i=1:max_iter;

%%%%% optimal consumption given these coefficints

c = solve_brent('rhs_bellman',s,parameters,a,fspace,cmin,cmax,tol)

%%%%% maximized rhs of Bellman equation

v = rhs_bellman(c,s,parameters,a,fspace); %% v(a)
```

Numerical routine **`solve_brent`** does the maximization

# RHS of the Bellman equation

```
function y = rhs_bellman(c,s,parameters,a,fspace)

beta  = parameters.beta;
sigma = parameters.sigma;

u   = utility(c,sigma);

Ev = expected_value(c,s,parameters,a,fspace);

y   = u+beta*Ev;
```

# Evaluating $\mathbb{E}[v(k', z')\,|z]$

```
function y = expected_value(c,s,parameters,a,fspace)
```

```
Ev = 0;

kprime = z.*(k.^alpha) + (1-delta)*k-c;

for j=1:numel(ez),

zprime = max(min(z.^rhoz.*exp(ez(j)), zmax), zmin);

sprime = [kprime,zprime];

Ev = Ev+wz(j)*funeval(a,fspace,sprime);

end

y = Ev;
```

# Evaluating $\mathbb{E}[v(k', z') \,|z]$

- This last step computes

$$\mathbb{E}[\, v(k', z') \,|\, z]$$

where $v(\cdot)$ is approximated by basis functions

$$v(k', z') \approx \sum_{i=1}^{n_s} a_i \phi_i(k', z')$$

- Using quadrature to compute the expectation

$$\mathbb{E}[\, v(k', z') \,|\, z] \approx \sum_{j=1}^{n_z} \sum_{i=1}^{n_s} a_i \, \phi_i( \, z k^{\alpha} + (1-\delta)k - c \,, \; z^{\rho_z} e^{\varepsilon_j} \,) \, w_j$$

# Updating coefficients

```
Jacobian = 0;

%%%%% implied by optimal consumption
kprime = z.*(k.^alpha) + (1-delta)*k-c;

%%%%% build up Jacobian matrix of v(a)
for j=1:numel(ez),

zprime = max(min(z.^rhoz.*exp(ez(j)), zmax), zmin);

sprime = [kprime,zprime];

Jacobian = Jacobian+beta*wz(j)*funbas(fspace,sprime);

end

%%%%% Newton's method
anew = a - (Phi-Jacobian)\(Phi*a-v);
```

# Check if converged

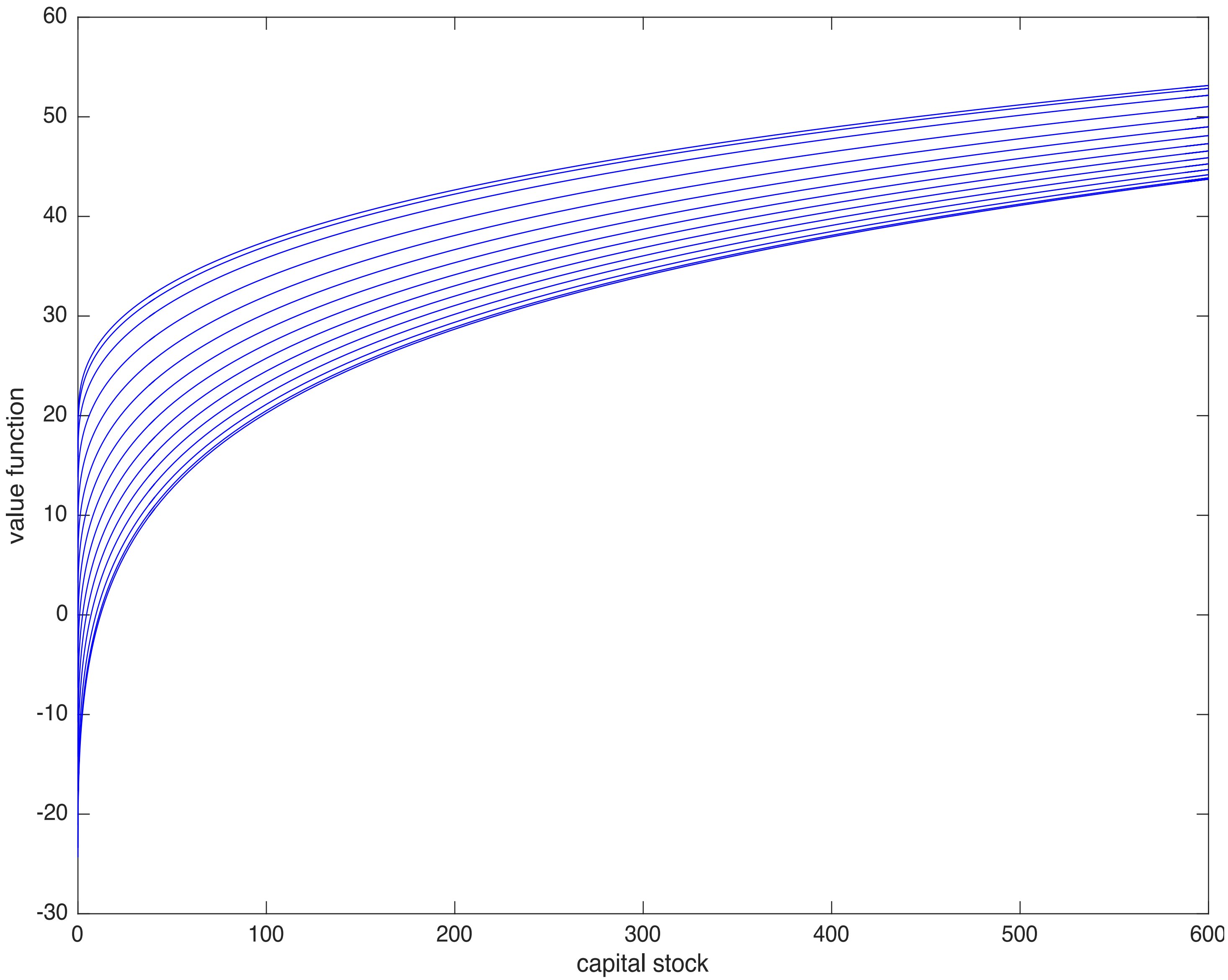
```
%%%%% check if converged

error = norm(anew-a,inf);

fprintf('%4i %6.2e \n',[i, error]);

if error<tol, break, end;

%%%%% if not converged, update and try again

a = anew;

end
```

# Reshape solution

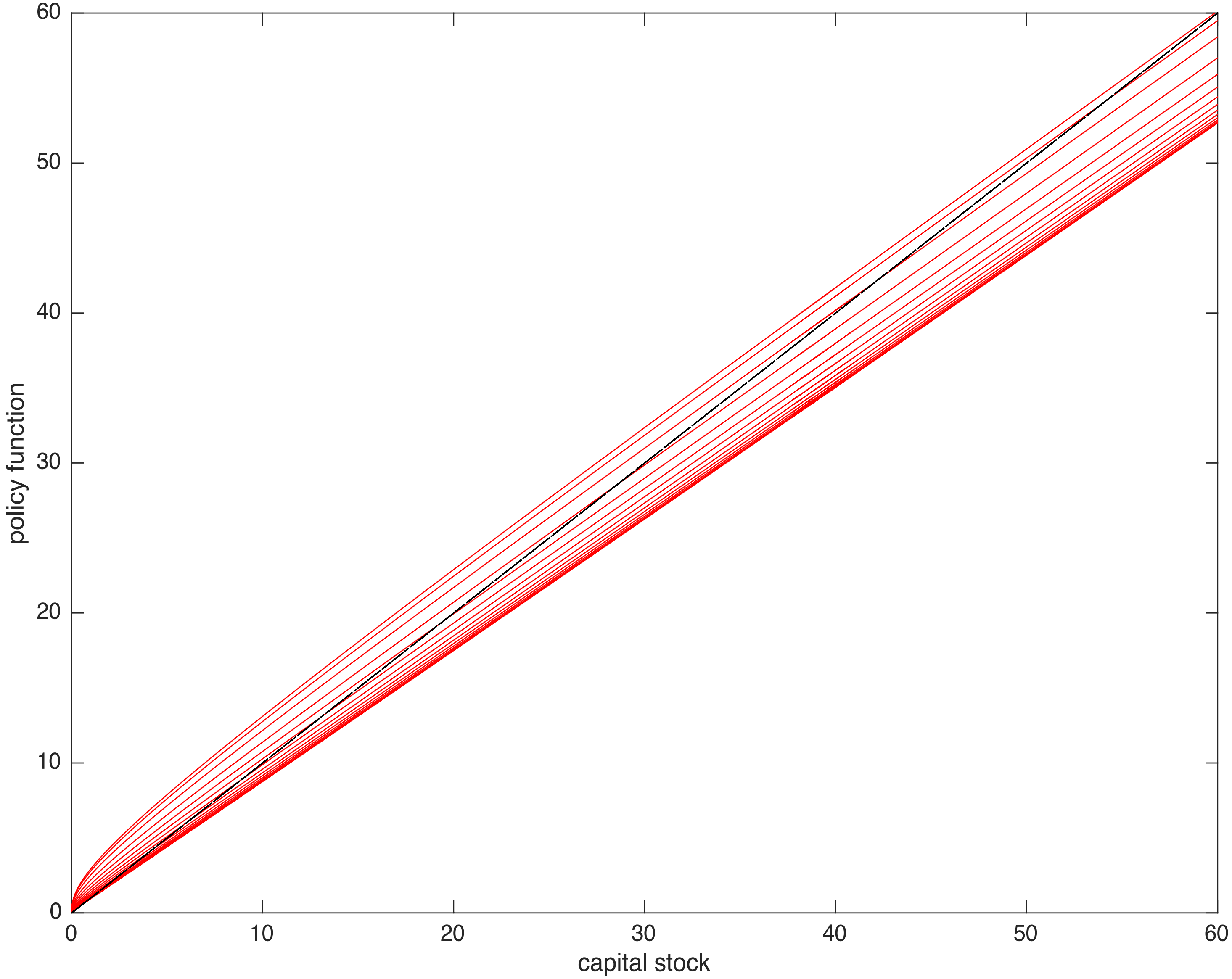
```
%%%%% reshape solution

Nk = numel(kgrid);
Nz = numel(zgrid);

VV = reshape(v,Nk,Nz); %% VV = v(k_i,z_j)
CC = reshape(c,Nk,Nz); %% CC = c(k_i,z_j)

%%%%% optimal policy k'=g(k,z)
GG = reshape(kprime,Nk,Nz); %% GG = g(k_i,z_j)
```

As usual with collocation methods, can also now interpolate as needed

# Value function $v(k, z)$ for various $z$



# Policy function $k' = g(k, z)$ for various $z$



# Next class

- Dynamic programming applications