1.2. Solving DSGE Models

Occasionally Binding Constraints in DSGE Models

Jonathan Swarbrick[1]
Bank of Canada

Bank of Canada – CMFE-Carleton Virtual Series
Advanced Topics in Macroeconomic Modelling

January 2021

[1]The views expressed are those of the authors and should not be interpreted as reflecting the views of the Bank of Canada.

General problem

- A **D**ynamic **S**tochastic **G**eneral **E**quilibrium model has the general form:

$$\mathbb{E}_t f\left(x_{t+1}, x_t, x_{t-1}, u_t\right) = 0 \tag{1}$$

  where $f(\cdot)$ is a *known* function.

- Usually recursive representation, i.e., same state variables $\Longrightarrow$ same decisions, implies policy:

$$x_t = g\left(x_{t-1}, u_t\right) \tag{2}$$

  in general, $g(\cdot)$ is an *unknown* function.

- Because $g(\cdot)$ is unknown, it must be solved (approximated) numerically

OBCs and local approximation 1/2

Approximating $g(\cdot)$ when the model is large is usually done using perturbation

- e.g., linearizing around the deterministic steady state.

What happens to any OBCs?

- Suppose the true policy function is:

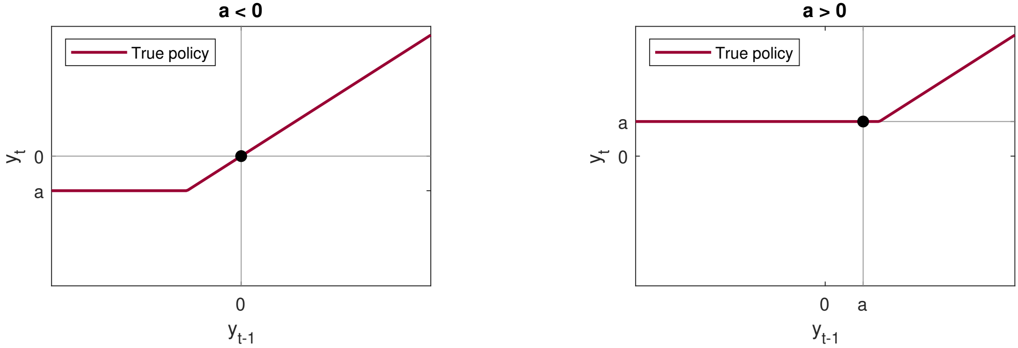
$$y_t = \begin{cases} \rho y_{t-1} + \epsilon_t & \text{if } (\rho y_{t-1} + \epsilon_t) \geq a \\ a & \text{otherwise} \end{cases} \tag{3}$$

where $0 < \rho < 1$. That is:

$$y_t = \max\{a, \rho y_{t-1} + \epsilon_t\} \tag{4}$$

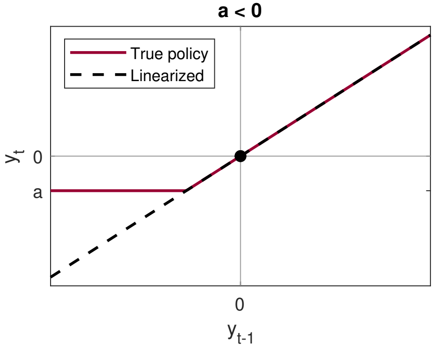
OBCs and local approximation 2/2

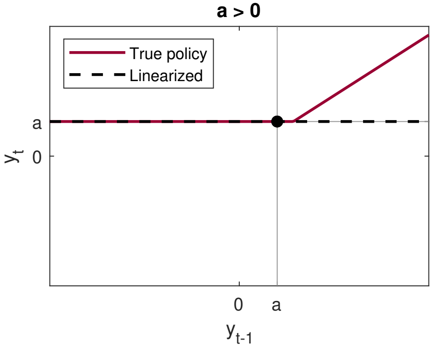
True policy function: $y_t = \max\{a, \rho y_{t-1} + \epsilon_t\}$



OBCs and local approximation 2/2

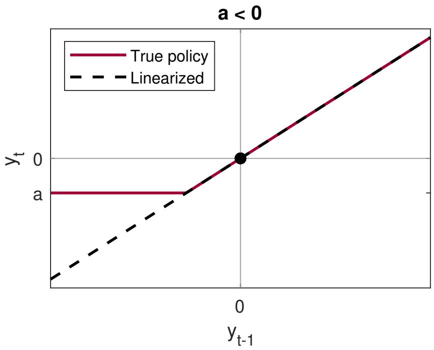
True policy function: $y_t = \max\{a, \rho y_{t-1} + \epsilon_t\}$

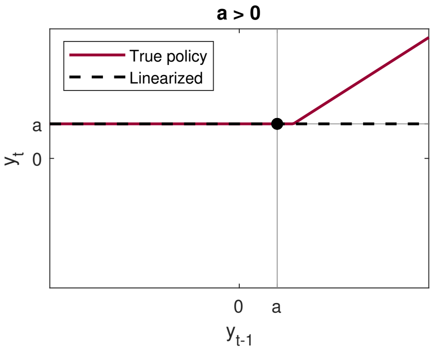




OBCs and local approximation 2/2

True policy function: $y_t = \max\{a, \rho y_{t-1} + \epsilon_t\}$

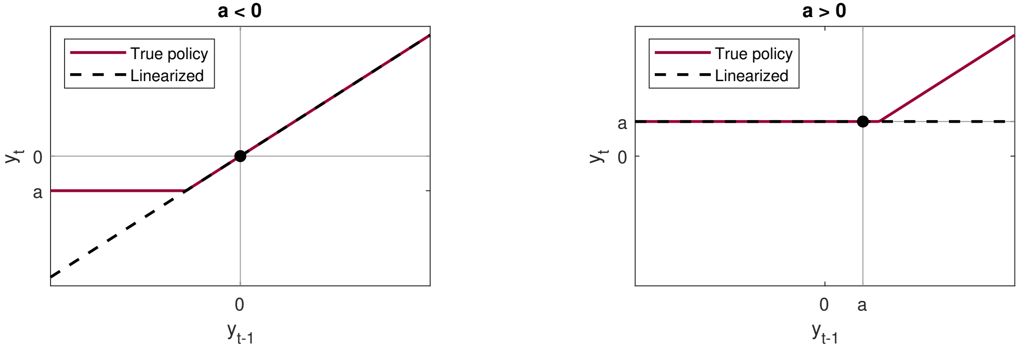




- We could use global approximation to retain kink

OBCs and local approximation 2/2

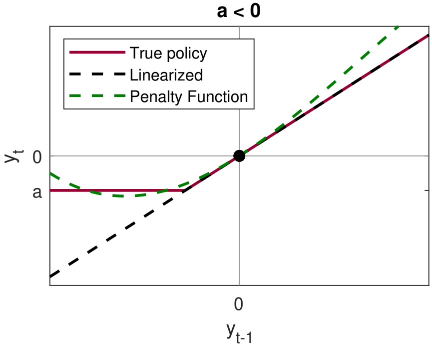
True policy function: $y_t = \max\{a, \rho y_{t-1} + \epsilon_t\}$

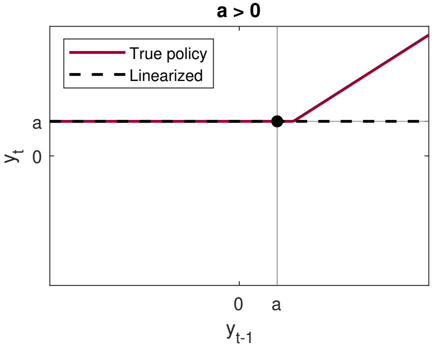


- We could use global approximation to retain kink
- We could use 'add-factors' in simulation to impose the bound: $y_t = \rho y_{t-1} + \epsilon_t + z_t$

OBCs and local approximation 2/2

True policy function: $y_t = \max\{a, \rho y_{t-1} + \epsilon_t\}$

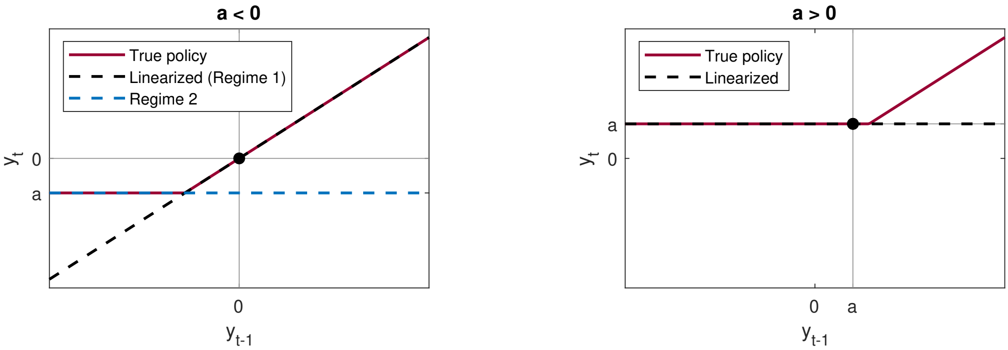




- We could use global approximation to retain kink
- We could use 'add-factors' in simulation to impose the bound: $y_t = \rho y_{t-1} + \epsilon_t + z_t$
- We could use a functional approximation to kink under higher-order local approximation
  $y_t = \rho y_{t-1} + \epsilon_t + b y_{t-1}^2 + c y_{t-1}^3$

OBCs and local approximation 2/2

True policy function: $y_t = \max\{a, \rho y_{t-1} + \epsilon_t\}$



- We could use global approximation to retain kink
- We could use 'add-factors' in simulation to impose the bound: $y_t = \rho y_{t-1} + \epsilon_t + z_t$
- We could use a functional approximation to kink under higher-order local approximation
- We could use local approximation with multiple regimes

Simple example

(Small) small open economy borrowing constraints model:

$$\max_{c_t, h_t, b_t} \mathbb{E}_0 \sum_t^{\infty} \beta^t \left(\log\left(c_t\right) + \chi \log\left(1 - h_t\right) - \delta b_t^2 \right) \tag{5}$$

$$\text{s.t.} \quad c_t + b_t = \exp\left(z_t\right) h_t + r b_{t-1} \tag{6}$$

$$b_t \geq \underline{b} \tag{7}$$

where

$$z_t = \rho z_{t-1} + \epsilon_t, \quad \epsilon_t \sim N(0, \sigma^2) \tag{8}$$

Note:

- $\delta > 0$ is a cost of non-zero b. Removes a unit root, can set $\delta \approx 0$
- SOE because r is exogenous
- Could also think of as partial equilibrium – could be used to compute decision rules in heterogeneous agent model

First-order conditions

Solving the household problem yields:

$$\frac{1}{c_t} - r\beta\mathbb{E}_t\left[\frac{1}{c_{t+1}}\right] - \mu_t + 2\delta b_t = 0 \tag{9}$$

$$\chi\frac{c_t}{1-h_t} = \exp\left(z_t\right) \tag{10}$$

$$\mu_t\left(b_t - \underline{b}\right) = 0 \tag{11}$$

$$b_t \geq \underline{b} \tag{12}$$

$$\mu_t \geq 0 \tag{13}$$

where μ_t is Lagrange multiplier on borrowing constraint.

- Equations (9)-(13) are the **Kuhn-Tucker conditions** and characterize the solution to an optimization problem with inequality constraints.
- Note that μ_t is the value of relaxing the borrowing constraint. $\mu_t = 0$ when the borrowing constraint is slack, $b_t > \underline{b}$; and $\mu_t > 0$ when the borrowing constraint binds $b_t = \underline{b}$
- Therefore, we can write

$$\min\left\{\mu_t, b_t - \underline{b}\right\} = 0 \tag{14}$$

The Bellman equation

The state variables in the model are b_t and z_t. Note, state variables are

- Predetermined variables – capital stock, or debt levels (here: saving b_{t-1})
- Exogenous variables (here: productivity z_t)
- Variables determined within the period are not state variables (here: consumption and hours c_t, h_t)

The Bellman equation

The state variables in the model are b_t and z_t. Note, state variables are

- Predetermined variables – capital stock, or debt levels (here: saving b_{t-1})
- Exogenous variables (here: productivity z_t)
- Variables determined within the period are not state variables (here: consumption and hours c_t, h_t)

We could have written the household problem

$$V\left(b_{t-1}, z_t\right) = \max_{c_t, h_t, b_t} \log\left(c_t\right) + \chi \log\left(1 - h_t\right) - \delta {b_t}^2 + \mathbb{E}_t \beta V\left(b_t, z_{t+1}\right) \tag{15}$$

$$\text{s.t.} \quad c_t = \exp\left(z_t\right) h_t + r b_{t-1} - b_t \tag{16}$$

$$b_t \geq \underline{b} \tag{17}$$

Equation (15) is a *Bellman equation*.

Solution methods

The model solution will imply a policy function in the form:

$$b_t = g\left(b_{t-1}, z_t\right) \tag{18}$$

$c_t = c\left(b_{t-1}, z_t\right)$ and $h_t = h\left(b_{t-1}, z_t\right)$ can already be solved recursively in closed form.

Solution methods

The model solution will imply a policy function in the form:

$$b_t = g\left(b_{t-1}, z_t\right) \tag{18}$$

$c_t = c\left(b_{t-1}, z_t\right)$ and $h_t = h\left(b_{t-1}, z_t\right)$ can already be solved recursively in closed form.

Options to approximate $g\left(b_{t-1}, z_t\right)$:

1. Solve the household first-order conditions first and use:
   - Projection methods to solve a *global* approximation
   - Perturbation to solve a *local* approximation
2. Directly solve the Bellman numerically
   - Gives a global approximation
   - Slower than projection but more robust

Quick note on global methods

In this course we only touch on global methods, we will brush over topics such as:

- Function approximation: there are different classes (e.g., polynomials and splines), ways of constructing basis functions and defining nodes;
- Numerical integration: different versions of quadrature, various Monte Carlo methods;
- Maximization methods (for solving Bellman)

For further reading check out: Ljungqvist & Sargent (2004) (dynamic programming), Judd (1998) and Miranda & Fackler (2004) (numerical methods).

Value Function Iteration

Value function iteration (VFI)

Bellman:

$$V\left(b,z\right)=\max_{b'}U(b,b',z)+\mathbb{E}\beta V\left(b',z'\right) \tag{19}$$

We want to solve the value function $V^{i}\left(b,z\right)$ iteratively

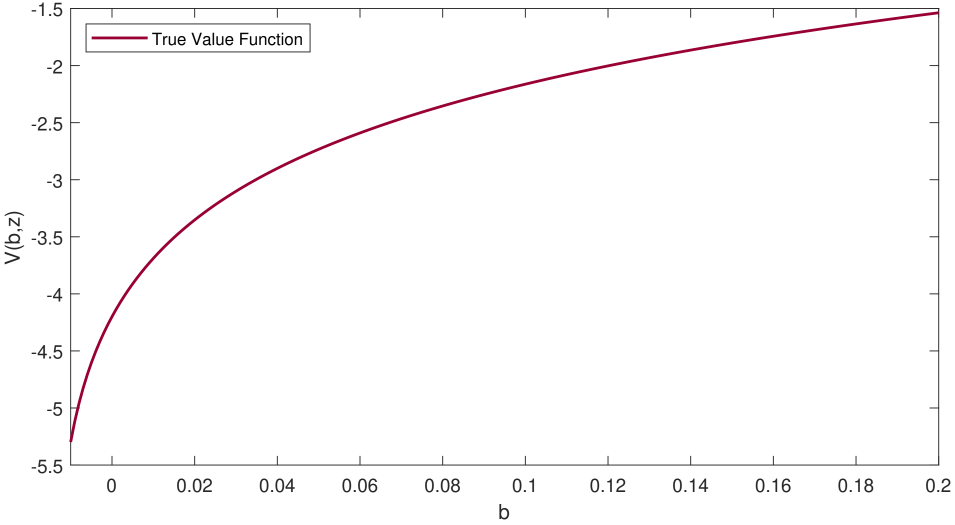
- i is the iteration number
- can choose from many functional forms for V

Obtain $V^{i+1}\left(b,z\right)$ from

$$V^{i+1}\left(b,z\right)=\max_{b'}U(b,b',z)+\beta\int_{z'}p(z'|z)V^{i}\left(b',z'\right) \tag{20}$$

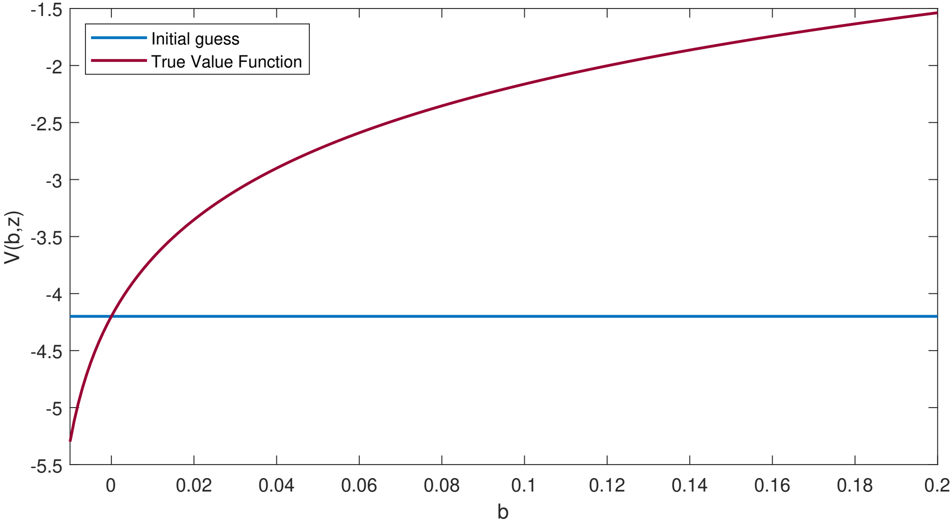
Approximating function for $V(b, z)$

Example $V(b, z) \equiv \log(b + 0.015)$



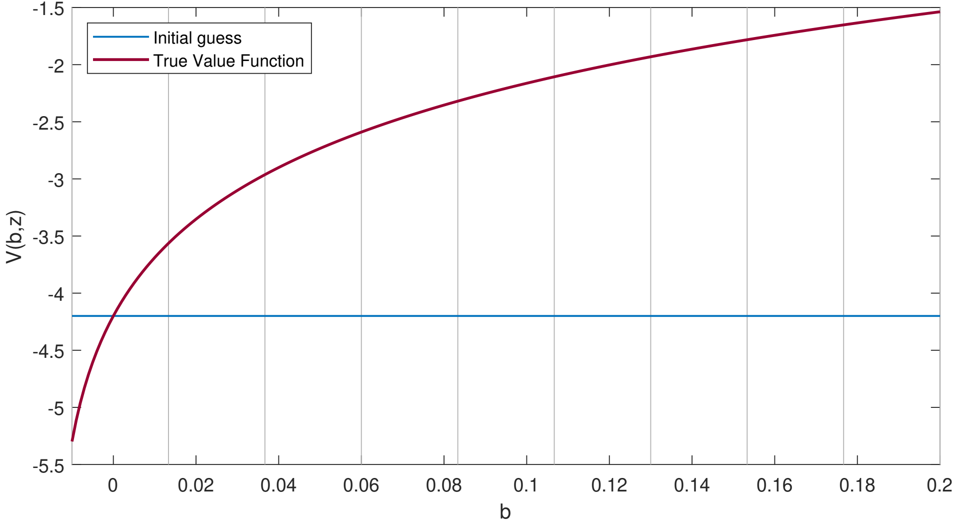
Approximating function for $V(b, z)$

Example $V(b, z) \equiv \log(b + 0.015)$



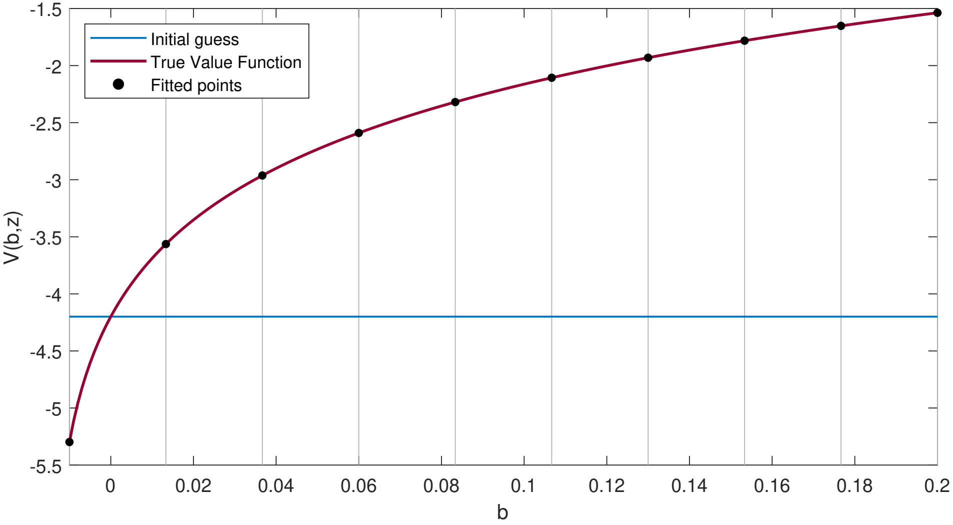
Approximating function for $V(b, z)$

Example $V(b, z) \equiv \log(b + 0.015)$



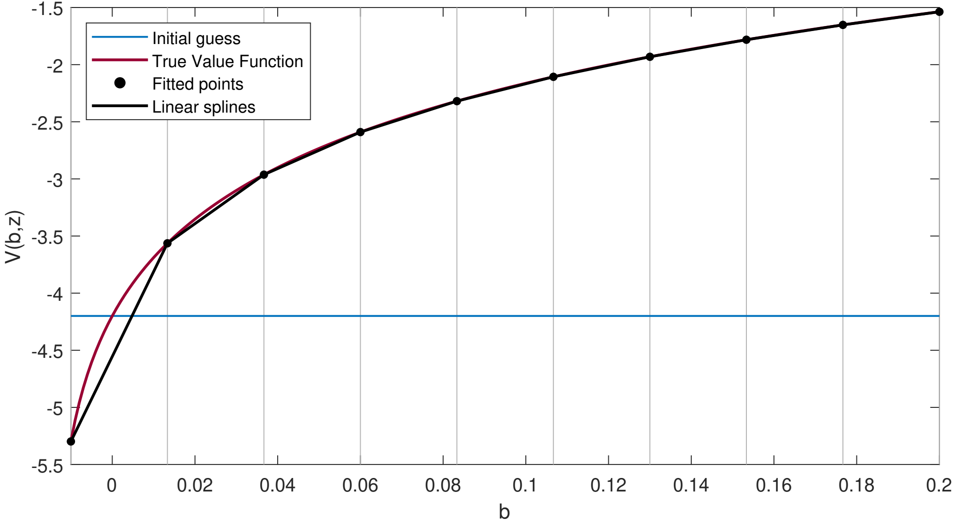
Approximating function for $V(b, z)$

Example $V(b, z) \equiv \log(b + 0.015)$



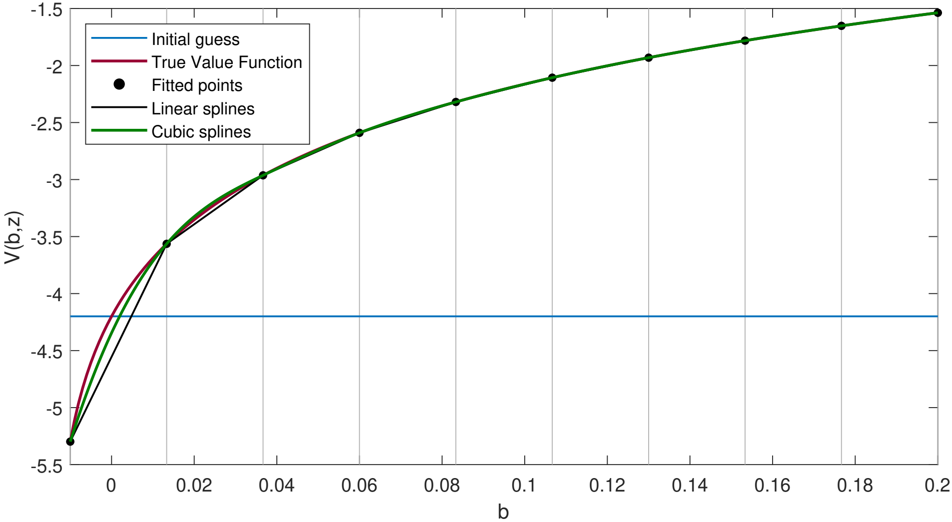
Approximating function for $V(b, z)$

Example $V(b, z) \equiv \log(b + 0.015)$



Approximating function for $V(b, z)$

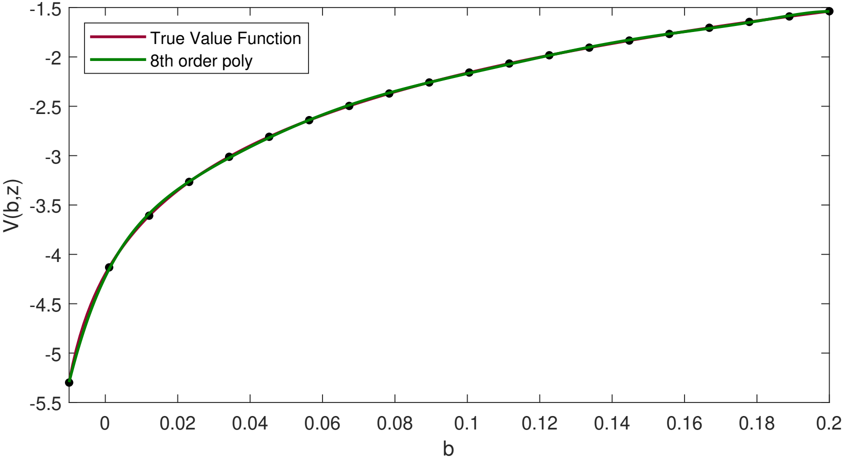
Example $V(b, z) \equiv \log(b + 0.015)$



Other options to approximate $V(b, z)$

Other approximations: e.g.:

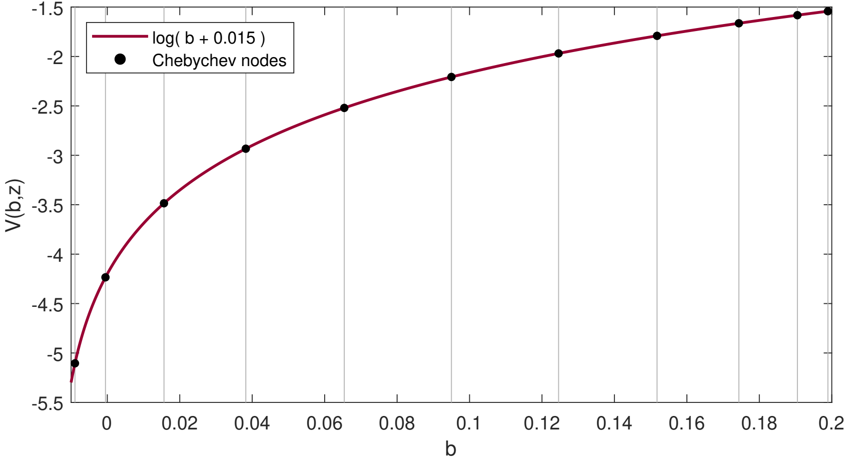
- Polynomials (fitting with OLS)



Other options to approximate $V(b, z)$

Other approximations: e.g.:

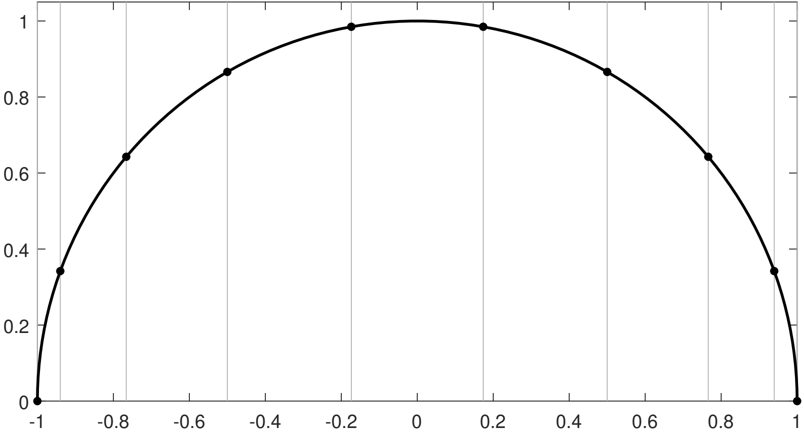
- Polynomials (fitting with OLS)
- Chebychev polynomials



Other options to approximate $V(b, z)$

Other approximations: e.g.:

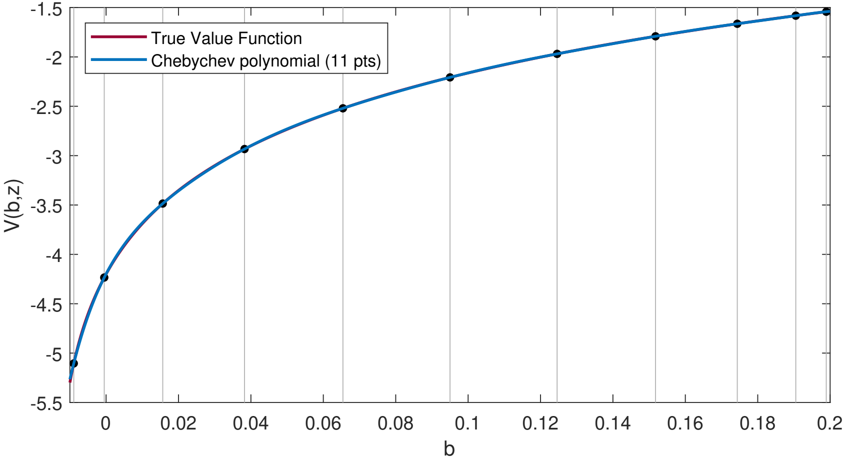
- Polynomials (fitting with OLS)
- Chebychev polynomials



Other options to approximate $V(b, z)$

Other approximations: e.g.:

- Polynomials (fitting with OLS)
- Chebychev polynomials



Maximisation

Another key ingredient is the maximisation part of the algorithm.

- This is the most computationally demanding aspect.

We want to solve:

$$\max_{b'} U(b, b', z) + \beta \sum_{z'} p(z'|z) V^i \left(b', z'\right) \tag{21}$$

Conditional on grid $\{b, z\}$, the integration points and weights $\{p(z'|z), z'\}$ and the current value function $V^i \left(b', z'\right)$

Maximisation

Another key ingredient is the maximisation part of the algorithm.

- This is the most computationally demanding aspect.

We want to solve:

$$\max_{b'} U(b, b', z) + \beta \sum_{z'} p(z'|z) V^i \left(b', z'\right) \tag{21}$$

Conditional on grid $\{b, z\}$, the integration points and weights $\{p(z'|z), z'\}$ and the current value function $V^i \left(b', z'\right)$

One option is a grid search:

1. Define a (fine) grid in b'
2. Compute $U(b, b', z) + \beta \sum_{z'} p(z'|z) V^i \left(b', z'\right)$
   - If N_b, N_z and $N_{b'}$ are number of b, z and b' grid points respectively, this will give a $N_b \times N_z \times N_{b'}$ matrix.
3. Find the value of b' for each b, z that maximizes the value from this matrix

Better optimization options

Many better options for the maximization:

- Golden search: derivative-free iterative search algorithm
- Newton's (or quasi-Newton) method: derivative based iterative algorithm
  - Can be applied to Kuhn-Tucker conditions via sequential quadratic programming (with matlab's `fmincon`)
- Many other options for more difficult problems
  - E.g.: pattern search, genetic algorithms, simulated annealing, swarm search (all available in Matlab global optimization toolbox)

Better optimization options

Many better options for the maximization:

- Golden search: derivative-free iterative search algorithm
- Newton's (or quasi-Newton) method: derivative based iterative algorithm
  - Can be applied to Kuhn-Tucker conditions via sequential quadratic programming (with matlab's `fmincon`)
- Many other options for more difficult problems
  - E.g.: pattern search, genetic algorithms, simulated annealing, swarm search (all available in Matlab global optimization toolbox)

Very parallelizable – can solve each grid point independently.

- Useful for larger models

VFI implementation

See code: **/borrowing_constraints/VFI/soe_obc.m**

- Code begins with parameter values and initializations of the various options
- Start with grid of b, b' and z. Each is a vector of possible values:

```
b = linspace(min_b , max_b , b_pts);
b_p = linspace(min_b , max_b , b_p_pts);
z = linspace( min_z , max_z , z_pts );
```

  Note `min_b` is $\underline{b}$ and `max_b` should be high enough to encompass all possible b

- We make an initial guess for the value function $V^1(b, z)$:

```
V = steady.v * ones( b_pts , z_pts );
```

  where `steady.v` is the deterministic steady state value $\bar{V} = u(c, h)/(1 - \beta)$

- Wide choice for functional form for V, we choose cubic splines.

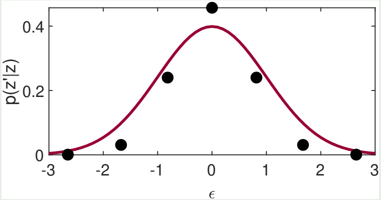
VFI: the loop

Given the defined grids and initial guesses, we can solve the Bellman:

1. Step one: maximise the value function

$$V^{i+1}(b,z) = \max_{b'} U(b,b',z) + \beta \sum_{z'} p(z'|z) V^{i}(b',z') \tag{22}$$

We use Gauss-Hermite Quadrature to choose nodes (values of z') and weights $p(z'|z)$:




```
[q_n,q_w]= hernodes(q_pts);
p_i = ( q_w ./ sqrt(pi) );
eps_p = sqrt(2)*p.sigma*q_n;
cont_V = zeros( b_p_pts , z_pts );
for i = 1:q_pts
  z_p = p.rho * z_grid2 + eps_p( i );
  cont_V = cont_V + p.betta * p_i(i) * V_fun( ...  );
end
```

Monte-Carlo methods or other quadrature options also available

VFI: the loop

Given the defined grids and initial guesses, we can solve the Bellman:

1. Step one: maximise the value function

$$V^{i+1}(b, z) = \max_{b'} U(b, b', z) + \beta \sum_{z'} p(z'|z) V^{i}(b', z') \tag{22}$$

Many ways to solve maximisation. We use 'brute force' grid search:

- Slow but robust
- Define grid points over range of b'
- Choose the value of b' that maximizes V for each $\{b, z\}$
- Note that this imposes $b \geq \underline{b}$ if the minimum b in grid is $\underline{b}$

```
[V_new,b_opt_ind] = max( utility( b_mesh , b_p_mesh , z_mesh , p ) + cont_V , [] , 3 );
```

VFI: the loop

Given the defined grids and initial guesses, we can solve the Bellman:

1. Step one: maximise the value function

$$V^{i+1}(b,z) = \max_{b'} U(b,b',z) + \beta \sum_{z'} p(z'|z) V^{i}\left(b', z'\right) \tag{22}$$

2. Step two: update value function, check for convergence:

```
crit = max( max( abs( ( V_new - V ) ./ V_new ) ) );
V = V_new;
```

VFI: the loop

Given the defined grids and initial guesses, we can solve the Bellman:

1. Step one: maximise the value function

$$V^{i+1}(b,z) = \max_{b'} U(b,b',z) + \beta \sum_{z'} p(z'|z) V^{i}\left(b',z'\right) \tag{22}$$

2. Step two: update value function, check for convergence:

```
crit = max( max( abs( ( V_new - V ) ./ V_new ) ) );
V = V_new;
```

3. If not converged (crit $<$ error tolerance) back to step 1, otherwise exit and transform optimal indices into bond choice (i.e., policy function)

```
b_policy = b_p( b_opt_ind );
```

,

The OBC in VFI

How $b \geq \underline{b}$ is treated depends on the maximisation procedure.

- It is easily imposed under a grid search with the grid range
- Will be an additional constraint for the many other methods available

The OBC in VFI

How $b \geq \underline{b}$ is treated depends on the maximisation procedure.

- It is easily imposed under a grid search with the grid range
- Will be an additional constraint for the many other methods available

OBCs also affect choice of approximating function and how we select grid points

- Piecewise linear or splines may peform better than polynomial due to kink
- Important to concentrate more grid points near bound

Projection Methods

Projection methods: general idea 1/2

- First derive the model first-order conditions and then solve a functional equation of the form:

$$\mathcal{H}(g) = 0 \tag{23}$$

  for our unknown policy function $g(\cdot)$

- We do this by approximating:

$$b' = g(b, z) \approx \tilde{g}(b, z; \eta_k) \tag{24}$$

  where $\tilde{g}(\cdot)$ is an approximating function
  - As before, subject to choice of approximating basis functions - e.g. polynomial, splines etc

Projection methods: general idea 2/2

- Then 'projecting' $\mathcal{H}(\cdot)$ against the approximating functions:

$$e(b, z; \eta_k) = \mathcal{H}\left(\tilde{g}(b, z; \eta_k)\right) \tag{25}$$

where we want to minimize error $e(b, z; \eta_k)$.

Projection methods: solver or iteration

Minimize the error using either **minimisation routines/function solver** or **iteration**

1. **Minimisation routines / solver**: solve

$$\min_{\eta_k} \mathcal{H}\left(\tilde{g}\left(b, z; \eta_k\right)\right) \tag{26}$$

   Lots of options, including most common:

   - Collocation: setting $e(\cdot) = 0$ at each point, perhaps with Chebychev polynomials
   - Galerkin: uses minimization routine and error term weighting to minimize $e(\cdot)$

Projection methods: solver or iteration

Minimize the error using either **minimisation routines/function solver** or **iteration**

1. **Minimisation routines / solver**: solve

$$\min_{\eta_k} \mathcal{H}\left(\tilde{g}\left(b, z; \eta_k\right)\right) \tag{26}$$

   Lots of options, including most common:

   - Collocation: setting $e\left(\cdot\right) = 0$ at each point, perhaps with Chebychev polynomials
   - Galerkin: uses minimization routine and error term weighting to minimize $e\left(\cdot\right)$

2. **Iteration**: as with VFI – fixed point iteration:
   - Rearrange functional equation $\mathcal{H}\left(g\right)$ to be in the form:

$$\tilde{g}^{n+1}\left(b, z\right) = f\left(b, z, \tilde{g}^{n}\left(b, z\right), z'\right) \tag{27}$$

   for current iteration n – iterate until convergence $\tilde{g}^{n+1} - \tilde{g}^{n} \approx 0$

Projection methods: example 1/3

Recall that solving the household problem yields:

$$\frac{1}{c_t} - r\beta\mathbb{E}_t\left[\frac{1}{c_{t+1}}\right] - \mu_t + 2\delta b_t = 0 \tag{28}$$

$$\chi\frac{c_t}{1-h_t} = \exp\left(z_t\right) \tag{29}$$

$$\mu_t\left(b_t - \underline{b}\right) = 0 \tag{30}$$

$$b_t \geq \underline{b} \tag{31}$$

$$\mu_t \geq 0 \tag{32}$$

Projection methods: example 1/3

Recall that solving the household problem yields:

$$\frac{1}{c_t} - r\beta\mathbb{E}_t\left[\frac{1}{c_{t+1}}\right] - \mu_t + 2\delta b_t = 0 \tag{28}$$

$$\chi\frac{c_t}{1-h_t} = \exp\left(z_t\right) \tag{29}$$

$$\mu_t\left(b_t - \underline{b}\right) = 0 \tag{30}$$

$$b_t \geq \underline{b} \tag{31}$$

$$\mu_t \geq 0 \tag{32}$$

- The 'functional equation' to use is the Euler equation:

$$\frac{1}{c(b,b',z)} - r\beta\mathbb{E}_t\left[\frac{1}{c(b'',b',z')}\right] - \mu + 2\delta b' = 0 \tag{33}$$

Projection methods: example 2/3

- Functional equation:

$$\frac{1}{c(b, b', z)} - r\beta\mathbb{E}_t\left[\frac{1}{c(b'', b', z')}\right] - \mu + 2\delta b' = 0 \tag{34}$$

- Substitute in labour supply and budget constraint, and discretize state space:

$$\frac{1+\chi}{\exp(z) + rb - b'} - r\beta\sum_{z'} p(z'|z)\left[\frac{1+\chi}{\exp(z') + rb' - b''}\right] - \mu + 2\delta b' = 0 \tag{35}$$

Projection methods: example 2/3

- Functional equation:

$$\frac{1}{c(b,b',z)} - r\beta\mathbb{E}_t\left[\frac{1}{c(b'',b',z')}\right] - \mu + 2\delta b' = 0 \tag{34}$$

- Substitute in labour supply and budget constraint, and discretize state space:

$$\frac{1+\chi}{\exp(z) + rb - b'} - r\beta\sum_{z'} p(z'|z)\left[\frac{1+\chi}{\exp(z') + rb' - b''}\right] - \mu + 2\delta b' = 0 \tag{35}$$

- We can substitute in the policy function, $b' = g(b,z)$:

$$\frac{1+\chi}{\exp(z) + rb - g(b,z)} - r\beta\sum_{z'} p(z'|z)\left[\frac{1+\chi}{\exp(z') + rg(b,z) - g\left(g(b,z),z'\right)}\right] - \mu + 2\delta b' = 0 \tag{36}$$

Projection methods: example 3/3

We then:

1. Rearrange for to solve for $g(b, z)$
2. Substitute in $\tilde{g}_{n+1}(\cdot)$ on the LHS and $\tilde{g}_n(\cdot)$ on the LHS
3. Impose the borrowing constraint with a max operator

$$\tilde{g}_{n+1}(b, z) = \max\left\{\exp(z) + rb - \frac{1}{r\beta\sum_{z'} p(z'|z)\left(\exp(z') + r\tilde{g}_n(b, z) - \tilde{g}_n\left(\tilde{g}_n(b, z), z'\right)\right)^{-1} - \frac{2\delta b'}{1+\chi}}, \underline{b}\right\} \tag{37}$$

- Iterate over this until $e = \tilde{g}_{n+1}(b, z) - \tilde{g}_n(b, z)$ is within required tolerance

The OBC 1/2

- Returning to the original notation, equation (37) uses:

$$b_t = \max\left\{\exp\left(z_t\right) + rb_{t-1} - \frac{1+\chi}{r\beta\mathbb{E}_t\left[\frac{1}{c_{t+1}}\right] - 2\delta b_t}, \underline{b}\right\} \tag{38}$$

- Does this satisfy the Kuhn-Tucker conditions?

The OBC 2/2

- The Kuhn-Tucker conditions are:

$$\frac{1}{c_t} - r\beta\mathbb{E}_t\left[\frac{1}{c_{t+1}}\right] - \mu_t + \delta b_t = 0 \tag{39}$$

$$\mu_t\left(b_t - \underline{b}\right) = 0 \tag{40}$$

$$\mu_t, b_t - \underline{b} \geq 0 \tag{41}$$

The OBC 2/2

- The Kuhn-Tucker conditions are:

$$\frac{1}{c_t} - r\beta\mathbb{E}_t\left[\frac{1}{c_{t+1}}\right] - \mu_t + \delta b_t = 0 \tag{39}$$

$$\mu_t\left(b_t - \underline{b}\right) = 0 \tag{40}$$

$$\mu_t, b_t - \underline{b} \geq 0 \tag{41}$$

- As already discussed, we can verify that the condition

$$\min\left\{\mu_t, b_t - \underline{b}\right\} = 0 \tag{42}$$

is sufficient to ensure (40) and (41) are satisfied.

The OBC 2/2

- The Kuhn-Tucker conditions are:

$$\frac{1}{c_t} - r\beta\mathbb{E}_t\left[\frac{1}{c_{t+1}}\right] - \mu_t + \delta b_t = 0 \tag{39}$$

$$\mu_t\left(b_t - \underline{b}\right) = 0 \tag{40}$$

$$\mu_t, b_t - \underline{b} \geq 0 \tag{41}$$

- As already discussed, we can verify that the condition

$$\min\left\{\mu_t, b_t - \underline{b}\right\} = 0 \tag{42}$$

is sufficient to ensure (40) and (41) are satisfied.

- Into (42), we substitute (39) for μ_t and rearrange to find (38)

Projection implementation

See the code: **/borrowing_constraints/projection/soe_obc.m**

- Code begins with parameter values and initializations of the various options
- Start with grid of b and z. Each is a vector of possible values:

```
b = linspace( min_b , max_b , Nb );
z = linspace( min_z , max_z , Nz );
[ b_mesh , z_mesh ] = ndgrid( b , z );
```

  `ndgrid` creates `Nb`$\times$`Nz` matrices where element i, j representing a state-of-the-world.
  - As before, the grid should be large enough to cover plausible values of $b \geq \underline{b}$ and z.
  - Note: you could instead use non-uniform spacing
- Make an initial guess of policy function, perhaps: $b' = b$

```
  b_p = b_mesh;
```

Projection: the loop

Solve the right-hand side to give the next iteration policy function $\tilde{g}_{n+1}(b, z) = b'$

```
while ssr>err_tol
      b_p = max( exp(z_mesh)+r*b_mesh-(1+chi) ./ (muc_p - 2*delta*b_p) , b_limit );
      muc_p_new = ex_muc( b_mesh , z_mesh , b_p , params );
      error = muc_p_new- muc_p;
      ssr = sum( sum( error.^2 ) );
      muc_p = muc_p_new;
end;
```

- Again, uses Gauss-Hermite Quadrature for expectations
- The function `ex_muc` returns a value for the expected marginal utility of consumption $\mathbb{E}_t\left[u'\left(c_{t+1}\right)\right] \approx \sum_{z'} p(z', z)u'\left(c\left(z', b', b\right)\right)$ for every point in `b_mesh`.
- This time there is no optimization step. The max function just imposes the OBC.

Policy function

In this example, we interpolate between grid points using cubic splines. In matlab we can use:

```
b_p_p = interp2( z_mesh , b_mesh , b_p , z_p , b_p , 'spline' );
```

which, given b (`b_mesh`), z (`z_mesh`) and the computed $b' = \tilde{g}(z, b)$ (`b_p`) returns $b'' = (z', b')$ (`b_p_p`).

You can easily use different basis functions – try the CompEcon toolkit which is the companion to the Miranda & Fackler (2004) book.

- Download https://github.com/PaulFackler/CompEcon

Reporting results

In either case (VFI or projection) we can plot the policy function for given productivities:

```
plot( b , b_p( : , low_z ) ); hold on;
plot( b , b_p( : , high_z ) );
```

Reporting results

In either case (VFI or projection) we can plot the policy function for given productivities:

```
plot( b , b_p( : , low_z ) ); hold on;
plot( b , b_p( : , high_z ) );
```

We can the simulate time-series

```
for t=2:time_horizon
  prod(t) = rho * prod(t-1) + sigma * eps(t);
  bonds(t) =  interp2( z_mesh , b_mesh , b_p , prod(t) , bonds(t-1) , 'spline' );
end
cons(2:end) = ( exp( prod(2:end) ) + r .* bonds(1:end-1) - bonds(2:end) ) ./ (1 + chi
    );
hours = 1 - chi .* cons ./ exp( prod );
```

report moments, and compute generalized impulse response function

Projection methods: results

	Mean	Standard deviation	Skewness	
	Relative to no constraint	Relative to no constraint	Baseline	No constraint
Consumption	+0.03%	+15%	-0.22	0.09
Hours	-0.01%	-43%	-0.09	-0.04
Bonds / $\bar{c}$	0.3% $\longrightarrow$ 5%	-59%	1.18	0.007

- Households unable to smooth consumption at the constraint – more volatile and negatively skewed
- precautionary saving, and large skewness in bond holding.

Projection methods: results

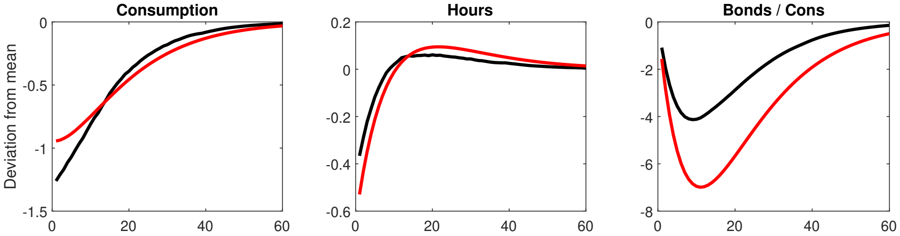
	Mean	Standard deviation	Skewness	
	Relative to no constraint	Relative to no constraint	Baseline	No constraint
Consumption	+0.03%	+15%	-0.22	0.09
Hours	-0.01%	-43%	-0.09	-0.04
Bonds / $\bar{c}$	0.3% $\longrightarrow$ 5%	-59%	1.18	0.007

- Households unable to smooth consumption at the constraint – more volatile and negatively skewed
- precautionary saving, and large skewness in bond holding.

	Stationary point
	Relative to no constraint
Consumption	+0.014%
Hours	-0.007%
Bonds / $\bar{c}$	-0.05% $\longrightarrow$ 2.1%

- Found solving $\bar{b}^* = g\left(\bar{b}^*, 0\right)$
- $\bar{b}^*$ is the risky steady state

Projection methods: generalized IRFs



- GIRF to large technology shock
- % deviation c and h, ppt deviation b/c.
- Red line = no constraint, black line = borrowing constraints model.

Alternatives to global methods

We've demonstrated it is (relatively) straightforward and fast to use projection

- nonlinear approximation to policy function $g(\cdot)$ over a large state space
- naturally captures OBCs
- approximation error depends on accuracy/speed trade off

However, these methods do not scale well so we look to methods suited to larger models.

Alternatives to global methods

We've demonstrated it is (relatively) straightforward and fast to use projection

- nonlinear approximation to policy function $g(\cdot)$ over a large state space
- naturally captures OBCs
- approximation error depends on accuracy/speed trade off

However, these methods do not scale well so we look to methods suited to larger models.

Local approximation (perturbation) will miss OBCs

- Either constraint *always* binds or *never* binds
- Depends on the fixed point the policy function is approximated around

Alternatives to global methods

We've demonstrated it is (relatively) straightforward and fast to use projection

- nonlinear approximation to policy function $g(\cdot)$ over a large state space
- naturally captures OBCs
- approximation error depends on accuracy/speed trade off

However, these methods do not scale well so we look to methods suited to larger models.

Local approximation (perturbation) will miss OBCs

- Either constraint *always* binds or *never* binds
- Depends on the fixed point the policy function is approximated around

Some solutions:

- Perfect-foresight simulations – abstract from role of uncertainty
- Perturbation with penalty function to mimic effect of OBC
- Perturbation with regime switching
- Perturbation with shocks or 'add factors' to impose the OBC

Perturbation

Perturbation

Recall the general form of the model:

$$\mathbb{E}_t f\left(x_{t+1}, x_t, x_{t-1}, u_t\right) = 0 \tag{43}$$

A perturbation solution begins with an easier problem with a known solution, e.g., the deterministic steady state:

$$f\left(\bar{x}, \bar{x}, \bar{x}, 0\right) = 0 \tag{44}$$

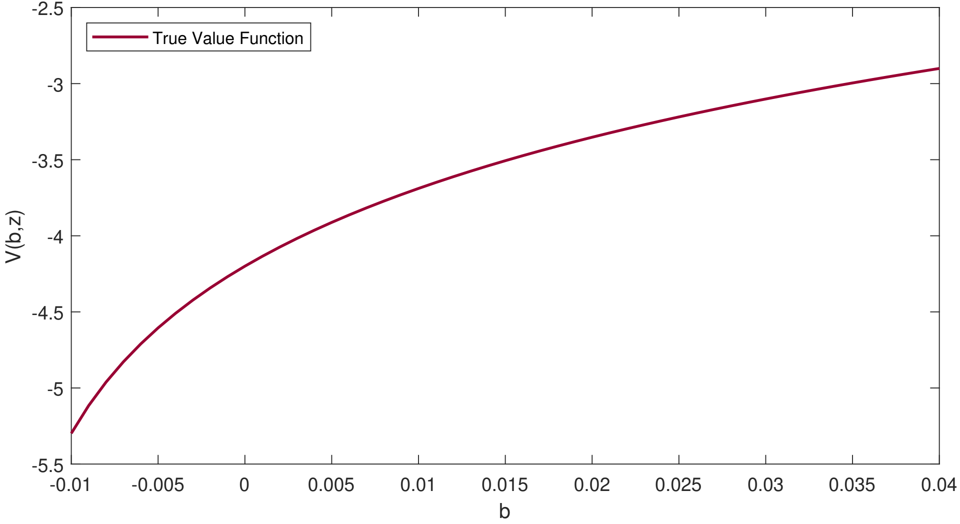
A Taylor approximation around this point can be taken up to the *n*th order and solved using standard methods (see Fernández-Villaverde et al. (2016) for a review).

- There are several matlab-based programs to do this (dynare, IRIS)
- Leads to policy function of the form (at first order):

$$x_t = \bar{x} + g_x\left(x_{t-1} - \bar{x}\right) + g_u u_t \tag{45}$$

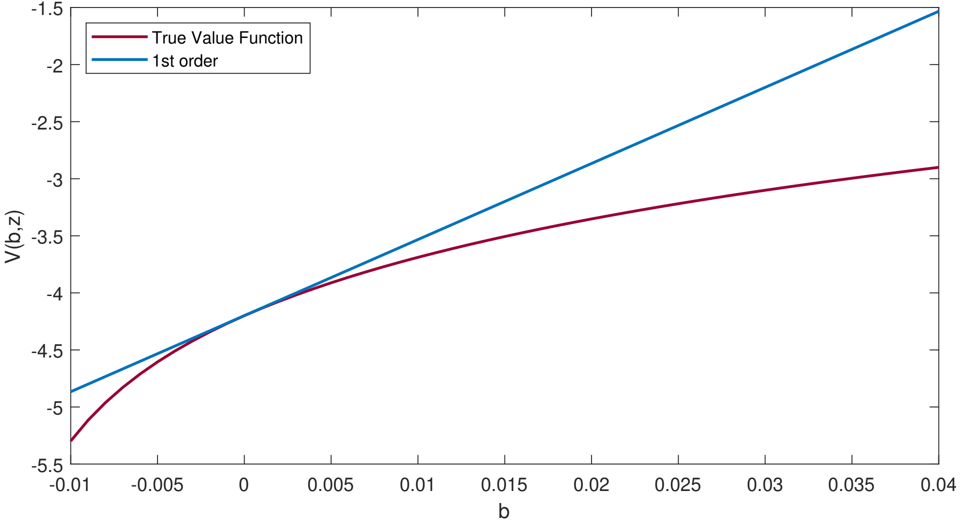
Perturbation approximation

Example $V(b, z) \equiv \log(b + 0.015)$



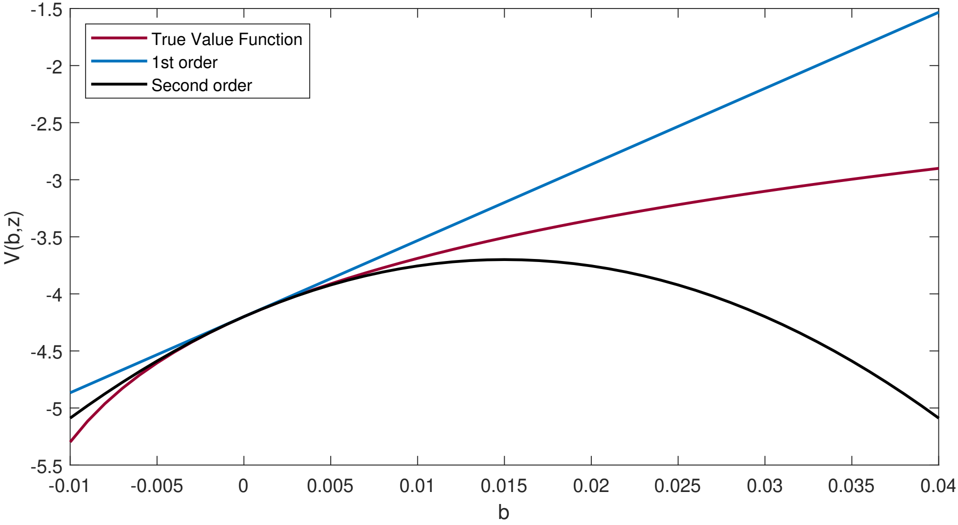
Perturbation approximation

Example $V(b,z) \equiv \log(b + 0.015)$



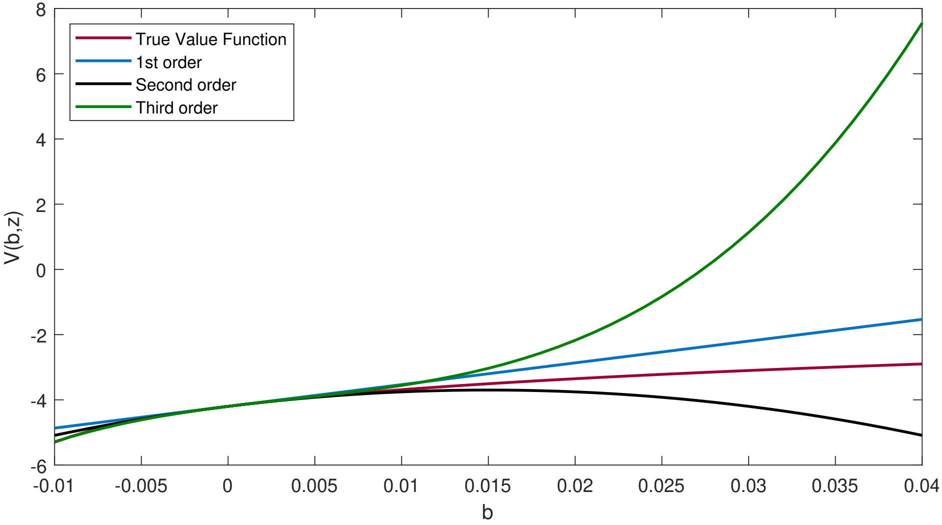
Perturbation approximation

Example $V(b,z) \equiv \log(b + 0.015)$



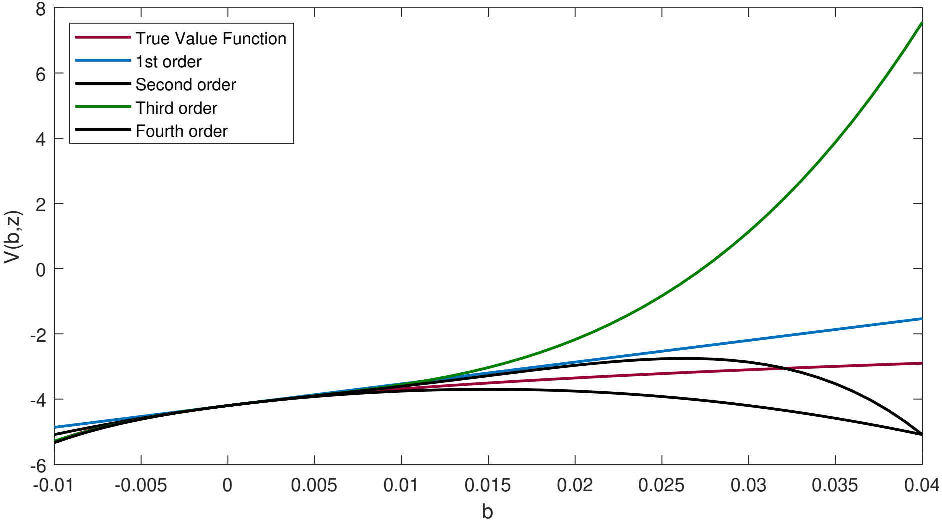
Perturbation approximation

Example $V(b,z) \equiv \log(b + 0.015)$



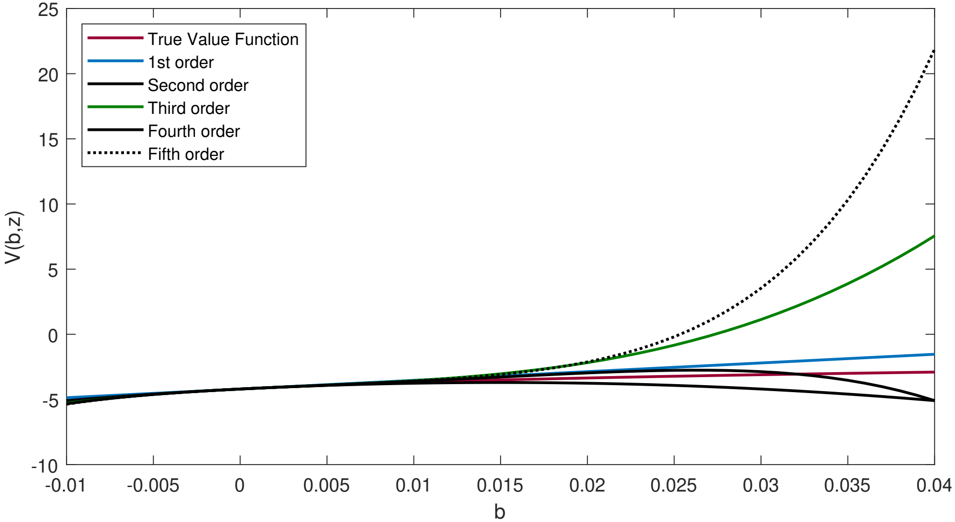
Perturbation approximation

Example $V(b,z) \equiv \log(b + 0.015)$



Perturbation approximation

Example $V(b, z) \equiv \log(b + 0.015)$



Perturbation: Some Considerations 1/2

- Higher-order polynomials have weird shapes
- The accuracy can be severely undermined away from the steady state
  - How far away can vary a lot
- Higher-orders can be explosive – pruning can help:
  1. Split higher-order approximation into linear and non-linear parts
  2. Use linear approximation in higher order terms, e.g. for 2nd-order:

$$x_t(1) = g_x x_{t-1}(1) + g_u u_t \tag{46}$$

$$x_t(2) = g_x x_{t-1}(2) + g_u u_t + g_{xx}\left(x_{t-1}(1)\right)^2 + g_{xu} x_{t-1}(1) u_t^2 + g_{uu} u_t^2 \tag{47}$$

- No theory to say 2nd-order better than 1st, 3rd-order better than 2nd

Perturbation: Some Considerations 2/2

On risk:

- First order: no risk premia
- Second order: constant risk premia
- Third order+: time varying risk premia

Models with interesting frictions or of financial crises require order ...?

- If the non-linearity is mostly coming from OBCs, perhaps 2nd/3rd order is sufficient.
- The key might be to capture the precautionary effects stemming from the OBC which would be present in an otherwise linear model
- Precautionary behaviour arises because of asymmetries even with linearized preferences

References I

Fernández-Villaverde, J., Rubio-Ramírez, J. & Schorfheide, F. (2016), Chapter 9 - solution and estimation methods for dsge models, Vol. 2 of *Handbook of Macroeconomics*, Elsevier, pp. 527 – 724. **URL:** *http://www.sciencedirect.com/science/article/pii/S1574004816000070*

Judd, K. L. (1998), *Numerical Methods in Economics*, Vol. 1 of *MIT Press Books*, The MIT Press.

Ljungqvist, L. & Sargent, T. J. (2004), *Recursive Macroeconomic Theory, 2nd Edition*, Vol. 1 of *MIT Press Books*, The MIT Press.

Miranda, M. J. & Fackler, P. L. (2004), *Applied Computational Economics and Finance*, Vol. 1 of *MIT Press Books*, The MIT Press.