Numerical Methods for Time Inconsistency, Private Information and Limited Commitment

Antonio Mele

University of Surrey

July 2017

NUMERICAL METHODS

Numerical methods

- The codes presented for VFI and PFI have many drawbacks!!!

NUMERICAL METHODS

- The codes presented for VFI and PFI have many drawbacks!!!
- Brief introduction to basic numerical tasks

NUMERICAL METHODS

- The codes presented for VFI and PFI have many drawbacks!!!
- Brief introduction to basic numerical tasks
- Not exhaustive

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)

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

COMPECON TOOLBOX AND LIBM

- CompEcon Toolbox: useful numerical routines for the economist
- LIBM: another library with efficiently coded routines (created by Michael Reiter)

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

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

Calculating integrals

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

CALCULATING INTEGRALS

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

- Different ways to choose nodes x_i and the weights w_i

CALCULATING INTEGRALS

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

- 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

CALCULATING INTEGRALS

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

- 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

CALCULATING INTEGRALS

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

- 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

CALCULATING INTEGRALS

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

- 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)

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
    - Simpson's rule: piecewise quadratic interpolants

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

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

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:

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

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:

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

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

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:

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

- 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 with 11 nodes.
2. Calculate the integral of $|x|^{\frac{1}{2}}$ on the interval $[-1,1]$ using Simpson's rule with 11 nodes.
3. Calculate the integral of $(1+25x^2)^{-1}$ on the interval $[-1,1]$ using Gauss-Legendre method with 11 nodes.

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 11, 21, 31, 101 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:

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

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

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

CALCULATING INTEGRALS

GAUSSIAN QUADRATURE FOR NORMAL DISTRIBUTION

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

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.

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_2 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 $f(a_1) < 0, f(b_1) > 0$

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 $[x_2, b_1]$.

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);
```

NONLINEAR EQUATIONS

BISECTION METHOD

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 $x = g(x)$

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 $x^{(1)} = g(x^{(0)})$

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

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);
```

NONLINEAR EQUATIONS

FUNCTION ITERATION

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 $x = x - f(x)$).

Experiment with the choice of the initial guess and see what happens.

Nonlinear equations

Newton's method

$$f(x) = 0$$

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

- First order Taylor approximation around $x^{(k)}$:

$$f(x) \approx f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}) = 0$$

NONLINEAR EQUATIONS

NEWTON'S METHOD

$$f(x) = 0$$

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

- First order Taylor approximation around $x^{(k)}$:

$$f(x) \approx f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}) = 0$$

- Guess $x^{(0)}$, calculate $x^{(1)} = x^{(0)} - [f'(x^{(0)})]^{-1} f(x^{(0)})$

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

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.

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.

Nonlinear equations

Newton's method

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**

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 $x^{(1)} = x^{(0)} - [A^{(0)}]^{-1} f(x^{(0)})$

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)

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);
```

NONLINEAR EQUATIONS

QUASI-NEWTON'S METHODS

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.

NONLINEAR EQUATIONS

QUASI-NEWTON'S METHODS

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.

OPTIMIZATION

DERIVATIVE-FREE METHODS

- **Golden search**: similar to bisection, find the maximum on interval $[a,b]$

Optimization

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 $f(x_i)$

OPTIMIZATION

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 $[x_1, b]$

OPTIMIZATION

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

OPTIMIZATION

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

OPTIMIZATION

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)
```

OPTIMIZATION

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)

OPTIMIZATION

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)
```

OPTIMIZATION

DERIVATIVE-FREE METHODS

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.

Optimization

Derivative-free methods

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

OPTIMIZATION

NEWTON-RAPHSON METHOD

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

$$f(x) \approx f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T f''(x^{(k)})(x - x^{(k)})$$

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: $f'(x^{(k)}) + f''(x^{(k)})(x - x^{(k)}) = 0$

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: $x^{(n+1)} = x^{(n)} - [f''(x^{(n)})]^{-1} f'(x^{(n)})$

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

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)

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-SHIT" PROBLEMS

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

- Global methods: line search, pattern search

OPTIMIZATION

"I-AM-IN-DEEP-SHIT" PROBLEMS

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

- Global methods: line search, pattern search
- Genetic algorithms

OPTIMIZATION

"I-AM-IN-DEEP-SHIT" PROBLEMS

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

- Global methods: line search, pattern search
- Genetic algorithms
- Simulated annealing

OPTIMIZATION

"I-AM-IN-DEEP-SHIT" 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

Optimization

"I-am-in-deep-shit" 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)

OPTIMIZATION

”I-AM-IN-DEEP-SHIT” 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

Curse of dimensionality

- VFI relies on setting up a grid for the state variables
- Need many points to get a good approximation,

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

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*.

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:

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

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

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

Projection methods

- How do we solve functional equations in general?

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

MAIN IDEA

We want to solve a functional equation of the type:

$$\mathscr{N}\left(g\left(x\right)\right)=0 \tag{2}$$

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

MAIN IDEA

We want to solve a functional equation of the type:

$$\mathscr{N}\left(g\left(x\right)\right) = 0 \tag{2}$$

- 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 $\phi_i(x)$: $\widehat{g}(x) = \sum_{i=1}^{n} a_i \phi_i(x)$

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

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 $p_i(x)$: $P_i(\cdot) \equiv \langle R(\cdot; \mathbf{a}), p_i(\cdot)\rangle, i = 1, \ldots, n$

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 $p_i(x)$: $P_i(\cdot) \equiv \langle R(\cdot; \mathbf{a}), p_i(\cdot) \rangle, i = 1, \ldots, n$
4. Find the vector of coefficients **a** that solves $P_i(\cdot) = 0$, $i = 1, \ldots, n$.

FIRST STEP

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

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)

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

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

FOURTH STEP

- Calculate the projections

$$P_i(\cdot) \equiv \langle R(\cdot;\mathbf{a}), p_i(\cdot)\rangle, i = 1, \ldots, n \tag{3}$$

FOURTH STEP

- Calculate the projections

$$P_i(\cdot) \equiv \langle R(\cdot;\mathbf{a}), p_i(\cdot)\rangle, i = 1, \ldots, n \tag{3}$$

- 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

FIFTH STEP

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

FIFTH STEP

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

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

CHOICE OF BASIS FUNCTIONS

CHOICE OF BASIS FUNCTIONS

- Ordinary polynomials $1, x, x^2, x^3, \ldots$ However, problematic
- Two broad categories: spectral methods and finite element methods.

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)

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

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:

CHEBYCHEV POLYNOMIALS (CONT.)

Let $z_l^n \equiv \cos\left(\frac{(2l-1)\pi}{2n}\right), l = 1, \dots, n$ 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 $i=0,1,...,n$:

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

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

MULTIDIMENSIONAL STATE SPACE

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

Choice of projections

- The choice of functions p_i characterizes different approaches

CHOICE OF PROJECTIONS

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

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

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**:

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

CHOICE OF PROJECTIONS (CONT.)

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

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

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

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

- This is equivalent to solve

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

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

SPEED CONCERNS

- Main computational burden is calculating projections
- Collocation is very fast; other methods would require the computation of an integral

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

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'`)

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'`)

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

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

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`

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);
```

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 $\{\phi_i(x)\}_{i=1}^{n}$

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`:

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);
```

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`

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);
```

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):

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

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

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

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

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

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

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

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

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

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

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

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

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`)

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`)

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`)

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.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);
```

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

```
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
```

focsSGM.m, PART 3

```
% equation to be solved: Euler equation
equ = (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

TRICKS AND DIFFICULTIES

CONVERGENCE

- There is no theorem guaranteeing convergence
- The researcher has to "guide" the convergence process with ad-hoc solutions

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

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

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

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

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

TRICKS AND DIFFICULTIES

ILL-CONDITIONING

- When using nonlinear solvers: Jacobians/Hessians must be computed
- Ill-conditioning is a very frequent problem, no easy fixes

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

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

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

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

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

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.

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.

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

1001
KEEP
CALM
AND
CODE
ON