

# SciencesPo Computational Economics Spring 2017

Florian Oswald

April 20, 2017

## 1 Numerical Dynamic Programming

Florian Oswald, Sciences Po, 2017

### 1.1 Intro

- Numerical Dynamic Programming (DP) is widely used to solve dynamic models.
- You are familiar with the technique from your core macro course.
- We will illustrate some ways to solve dynamic programs.
  1. Models with one discrete or continuous choice variable
  2. Models with several choice variables
  3. Models with a discrete-continuous choice combination
- We will go through:
  1. Value Function Iteration (VFI)
  2. Policy function iteration (PFI)
  3. Projection Methods
  4. Endogenous Grid Method (EGM)
  5. Discrete Choice Endogenous Grid Method (DCEGM)

### 1.2 Dynamic Programming Theory

- Payoffs over time are

$$U = \sum_{t=1}^{\infty} \beta^t u(s_t, c_t)$$

where  $\beta < 1$  is a discount factor,  $s_t$  is the state,  $c_t$  is the control.

- The state (vector) evolves as  $s_{t+1} = h(s_t, c_t)$ .
- All past decisions are contained in  $s_t$ .

### 1.2.1 Assumptions

- Let  $c_t \in C(s_t)$ ,  $s_t \in S$  and assume  $u$  is bounded in  $(c, s) \in C \times S$ .
- Stationarity: neither payoff  $u$  nor transition  $h$  depend on time.
- Write the problem as

$$v(s) = \max_{s' \in \Gamma(s)} u(s, s') + \beta v(s')$$

- $\Gamma(s)$  is the constraint set (or feasible set) for  $s'$  when the current state is  $s$

### 1.2.2 Existence

**Theorem.** Assume that  $u(s, s')$  is real-valued, continuous, and bounded, that  $\beta \in (0, 1)$ , and that the constraint set  $\Gamma(s)$  is nonempty, compact, and continuous. Then there exists a unique function  $v(s)$  that solves the above functional equation.

**Proof.** [stokeylucas] [4] theorem 4.6.

## 2 Solution Methods

### 2.1 Value Function Iteration (VFI)

- Find the fix point of the functional equation by iterating on it until the distance between consecutive iterations becomes small.
- Motivated by the Bellman Operator, and it's characterization in the Continuous Mapping Theorem.

### 2.2 Discrete DP VFI

- Represents and solves the functional problem in  $\mathbb{R}$  on a finite set of grid points only.
- Widely used method.
  - Simple (+)
  - Robust (+)
  - Slow (-)
  - Imprecise (-)
- Precision depends on number of discretization points used.
- High-dimensional problems are difficult to tackle with this method because of the curse of dimensionality.

#### 2.2.1 Deterministic growth model with Discrete VFI

- We have this theoretical model:

$$V(k) = \max_{0 < k' < f(k)} u(f(k) - k') + \beta V(k')$$

$$f(k) = k^\alpha$$

$k_0$  given

- and we employ the following numerical approximation:

$$V(k_i) = \max_{i'=1,2,\dots,n} u(f(k_i) - k_{i'}) + \beta V(i')$$

- The iteration is then on successive iterates of  $V$ : The LHS gets updated in each iteration!

$$\begin{aligned} V^{r+1}(k_i) &= \max_{i'=1,2,\dots,n} u(f(k_i) - k_{i'}) + \beta V^r(i') \\ V^{r+2}(k_i) &= \max_{i'=1,2,\dots,n} u(f(k_i) - k_{i'}) + \beta V^{r+1}(i') \\ &\dots \end{aligned}$$

- And it stops at iteration  $r$  if  $d(V^r, V^{r-1}) < \text{tol}$
- You choose a measure of *distance*,  $d(\cdot, \cdot)$ , and a level of tolerance.
- $V^r$  is usually an *array*. So  $d$  will be some kind of *norm*.
- maximal absolute distance
- mean squared distance

### 2.2.2 Exercise 1: Implement discrete VFI

## 2.3 Checklist

1. Set parameter values
2. define a grid for state variable  $k \in [0, 2]$
3. initialize value function  $V$
4. start iteration, repeatedly computing a new version of  $V$ .
5. stop if  $d(V^r, V^{r-1}) < \text{tol}$ .
6. plot value and policy function
7. report the maximum error of both wrt to analytic solution

```
In [1]: alpha      = 0.65
        beta       = 0.95
        grid_max   = 2 # upper bound of capital grid
        n          = 150 # number of grid points
        N_iter     = 3000 # number of iterations
        kgrid      = 1e-6:(grid_max-1e-6)/(n-1):grid_max # equispaced grid
        f(x) = x^alpha # defines the production function f(k)
        tol = 1e-9
```

```
Out[1]: 1.0e-9
```

## 2.4 Analytic Solution

- If we choose  $u(x) = \ln(x)$ , the problem has a closed form solution.
- We can use this to check accuracy of our solution.

```
In [2]: ab          = alpha * beta
        c1          = (log(1 - ab) + log(ab) * ab / (1 - ab)) / (1 - beta)
        c2          = alpha / (1 - ab)
```

```

# optimal analytical values
v_star(k) = c1 .+ c2 .* log(k)
k_star(k) = ab * k.^alpha
c_star(k) = (1-ab) * k.^alpha
ufun(x) = log(x)

```

Out[2]: ufun (generic function with 1 method)

In [4]: kgrid[4]

Out[4]: 0.04026943624161074

In [5]: # Bellman Operator

```

# inputs
# `grid`: grid of values of state variable
# `v0`: current guess of value function

# output
# `v1`: next guess of value function
# `pol`: corresponding policy function

#takes a grid of state variables and computes the next iterate of the value function.
function bellman_operator(grid,v0)

    v1 = zeros(n)      # next guess
    pol = zeros(Int,n)  # policy function
    w = zeros(n)       # temporary vector

    # loop over current states
    # current capital
    for (i,k) in enumerate(grid)

        # loop over all possible kprime choices
        for (iprime,kprime) in enumerate(grid)
            if f(k) - kprime < 0    #check for negative consumption
                w[iprime] = -Inf
            else
                w[iprime] = ufun(f(k) - kprime) + beta * v0[iprime]
            end
        end
        # find maximal choice
        v1[i], pol[i] = findmax(w)    # stores Value und policy (index of optimal choice)
    end
    return (v1,pol)    # return both value and policy function
end

# VFI iterator

```

```

#
## input
# `n`: number of grid points
# output
# `v_next`: tuple with value and policy functions after `n` iterations.
function VFI()
    v_init = zeros(n)      # initial guess
    for iter in 1:N_iter
        v_next = bellman_operator(kgrid,v_init) # returns a tuple: (v1,pol)
        # check convergence
        if maxabs(v_init.-v_next[1]) < tol
            errors = maxabs(v_next[1].-v_star(kgrid))
            perrors = maxabs(kgrid[v_next[2]].-k_star(kgrid))
            println("Found solution after $iter iterations")
            println("maximal value function error = $errors")
            println("maximal policy function error = $perrors")
            return v_next
        elseif iter==N_iter
            warn("No solution found after $iter iterations")
            return v_next
        end
        v_init = v_next[1] # update guess
    end
end

# plot
function plotVFI()
    v = VFI()
    figure("discrete VFI",figsize=(10,5))
    subplot(131)
    plot(kgrid,v[1],color="blue")
    plot(kgrid,v_star(kgrid),color="black")
    xlim(-0.1,grid_max)
    ylim(-50,-30)
    xlabel("k")
    ylabel("value")
    title("value function")

    subplot(132)
    plot(kgrid,kgrid[v[2]])
    plot(kgrid,k_star(kgrid),color="black")
    xlabel("k")
    title("policy function")

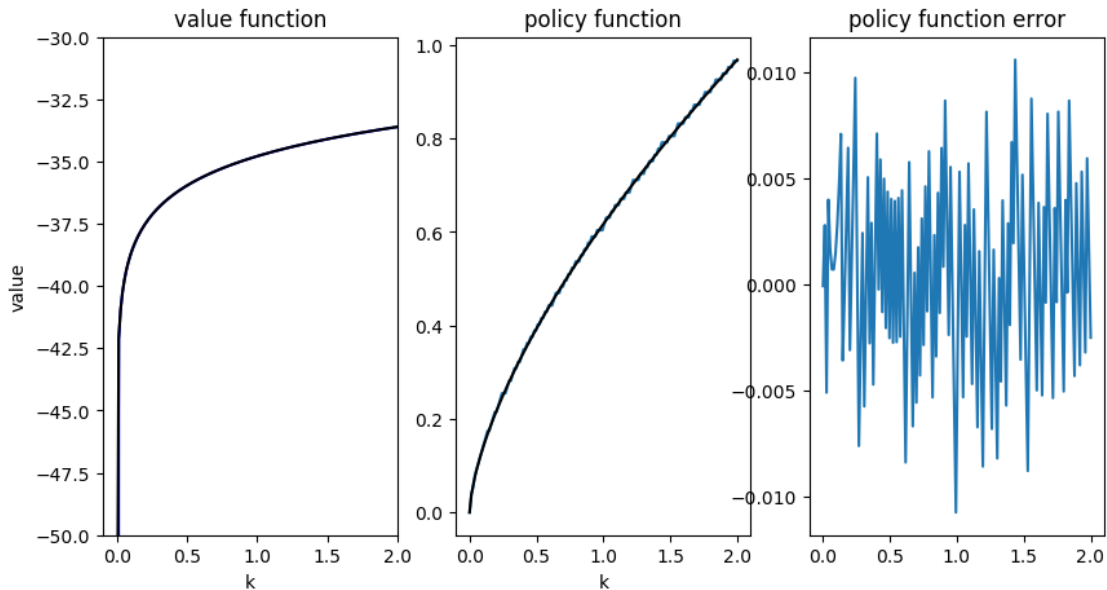
    subplot(133)
    plot(kgrid,kgrid[v[2]].-k_star(kgrid))
    title("policy function error")

```

```

end
using PyPlot
plotVFI()

```



```

Found solution after 448 iterations
maximal value function error = 121.49819145170268
maximal policy function error = 0.010775693497948935

```

Out [5]: PyObject <matplotlib.text.Text object at 0x328494ad0>

## 2.4.1 Exercise 2: Discretizing only the state space (not control space)

- Same exercise, but now use a continuous solver for choice of  $k'$ .
- in other words, employ the following numerical approximation:

$$V(k_i) = \max_{k' \in [0, \bar{k}]} \ln(f(k_i) - k') + \beta V(k')$$

- To do this, you need to be able to evaluate  $V(k')$  where  $k'$  is potentially off the  $k$ grid.
- use `Interpolations.jl` to linearly interpolate  $V$ .
  - the relevant object is setup with function `interpolate((grid,), v, Gridded{Linear}())`
- use `Optim::optimize()` to perform the maximization.
  - you have to define an objective function for each  $k_i$
  - do something like `optimize(objective, lb, ub)`

```

In [6]: using Interpolations
using Optim
function bellman_operator2(grid,v0)

    v1 = zeros(n)      # next guess
    pol = zeros(n)      # consumption policy function

    Interp = interpolate((collect(grid),), v0, Gridded{Linear}())

    # loop over current states
    # of current capital
    for (i,k) in enumerate(grid)

        objective(c) = - (log(c) + beta * Interp[f(k) - c])
        # find max of objective between [0,k^alpha]
        res = optimize(objective, 1e-6, f(k)) # Optim.jl
        pol[i] = f(k) - res.minimizer # k'
        v1[i] = -res.minimum
    end
    return (v1,pol) # return both value and policy function
end

function VFI2()
    v_init = zeros(n) # initial guess
    for iter in 1:N_iter
        v_next = bellman_operator2(kgrid,v_init) # returns a tuple: (v1,pol)
        # check convergence
        if maxabs(v_init.-v_next[1]) < tol
            verrors = maxabs(v_next[1].-v_star(kgrid))
            perrors = maxabs(v_next[2].-k_star(kgrid))
            println("continuous VFI:")
            println("Found solution after $iter iterations")
            println("maximal value function error = $verrors")
            println("maximal policy function error = $perrors")
            return v_next
        elseif iter==N_iter
            warn("No solution found after $iter iterations")
            return v_next
        end
        v_init = v_next[1] # update guess
    end
    return nothing
end

function plotVFI2()
    v = VFI2()
    figure("discrete VFI - continuous control",figsize=(10,5))
    subplot(131)

```

```

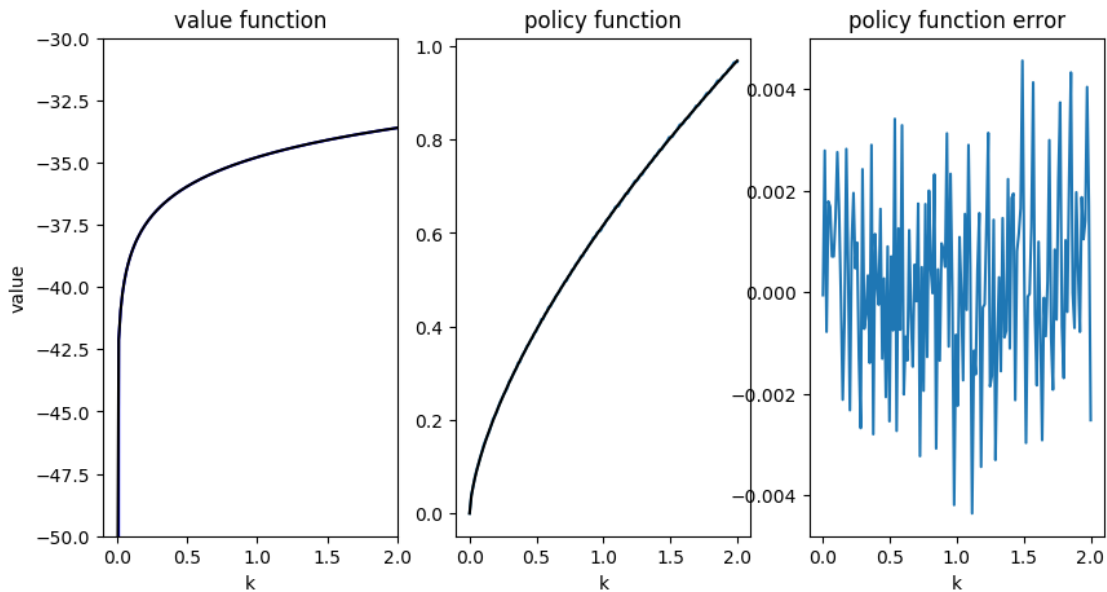
plot(kgrid,v[1],color="blue")
plot(kgrid,v_star(kgrid),color="black")
xlim(-0.1,grid_max)
ylim(-50,-30)
xlabel("k")
ylabel("value")
title("value function")

subplot(132)
plot(kgrid,v[2])
plot(kgrid,k_star(kgrid),color="black")
xlabel("k")
title("policy function")

subplot(133)
plot(kgrid,v[2].-k_star(kgrid))
xlabel("k")
title("policy function error")
end

```

```
plotVFI2()
```



continuous VFI:

Found solution after 439 iterations

maximal value function error = 121.10003370383764

maximal policy function error = 0.004564649814724597

Out[6]: PyObject <matplotlib.text.Text object at 0x32881c9d0>



## 2.5 Policy Function Iteration

- This is similar to VFI but we now guess successive *policy* functions
- The idea is to choose a new policy  $p^*$  in each iteration so as to satisfy an optimality condition. In our example, that would be the Euler Equation.
- We know that the solution to the above problem is a function  $c^*(k)$  such that

$$c^*(k) = \arg \max_z u(z) + \beta V(f(k) - z) \quad \forall k \in [0, \infty]$$

- We **don't** directly solve the maximization problem outlined above, but it's first order condition:

$$\begin{aligned} u'(c^*(k_t)) &= \beta u'(c^*(k_{t+1})) f'(k_{t+1}) \\ &= \beta u'[c^*(f(k_t) - c^*(k_t))] f'(f(k_t) - c^*(k_t)) \end{aligned}$$

- In practice, we have to find the zeros of

$$g(k_t) = u'(c^*(k_t)) - \beta u'[c^*(f(k_t) - c^*(k_t))] f'(f(k_t) - c^*(k_t))$$

In [9]: *# Your turn!*

```
using Roots
function policy_iter(grid,c0,u_prime,f_prime)

    c1 = zeros(length(grid))      # next guess
    pol_fun = interpolate((collect(grid),), c0, Gridded{Linear}())

    # loop over current states
    # of current capital
    for (i,k) in enumerate(grid)
        objective(c) = u_prime(c) - beta * u_prime(pol_fun[f(k)-c]) * f_prime(f(k)-c)
        c1[i] = fzero(objective, 1e-10, f(k)-1e-10)
    end
    return c1
end

uprime(x) = 1.0 ./ x
fprime(x) = alpha * x.^(alpha-1)

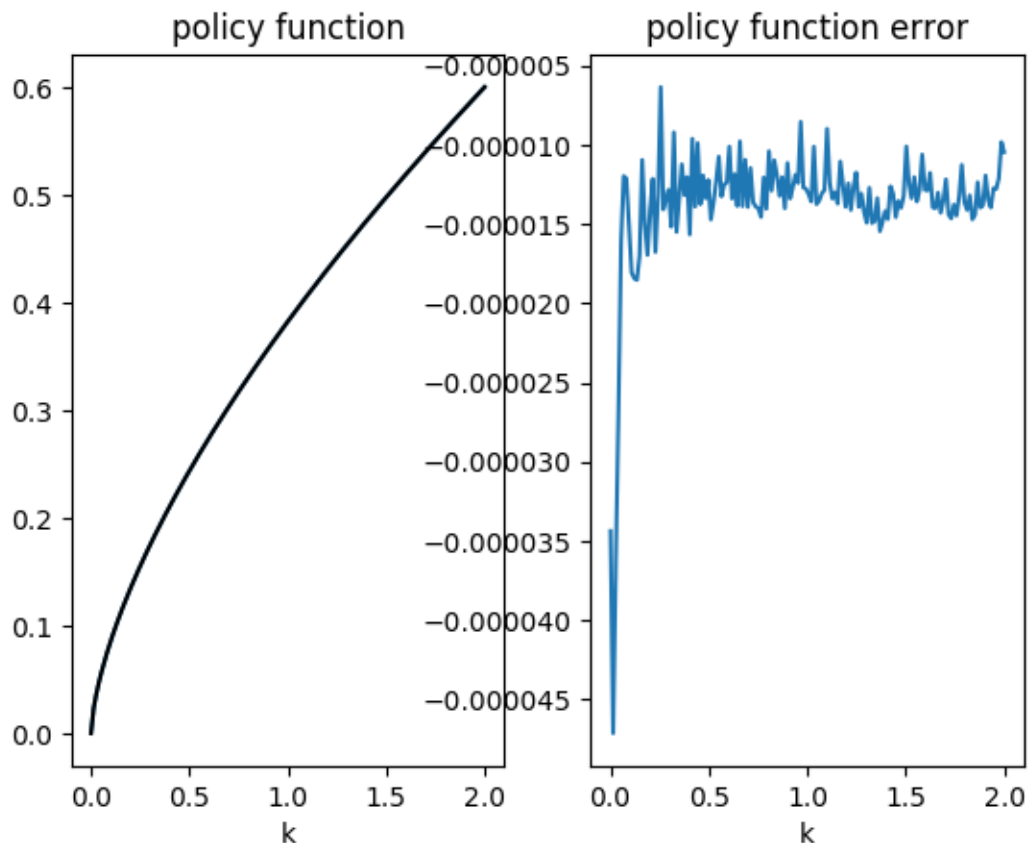
function PFI()
    c_init = kgrid
    for iter in 1:N_iter
        c_next = policy_iter(kgrid,c_init,uprime,fprime)
        # check convergence
        if maxabs(c_init.-c_next) < tol
```

```

        perrors = maxabs(c_next.-c_star(kgrid))
        println("PFI:")
        println("Found solution after $iter iterations")
        println("max policy function error = $perrors")
        return c_next
    elseif iter==N_iter
        warn("No solution found after $iter iterations")
        return c_next
    end
    c_init = c_next # update guess
end
end
function plotPFI()
    v = PFI()
    figure("PFI")
    subplot(121)
    plot(kgrid,v)
    plot(kgrid,c_star(kgrid),color="black")
    xlabel("k")
    title("policy function")

    subplot(122)
    plot(kgrid,v.-c_star(kgrid))
    xlabel("k")
    title("policy function error")
end
plotPFI()

```



PFI:

Found solution after 39 iterations

max policy function error = 4.71194965185294e-5

WARNING: Method definition policy\_iter(Any, Any, Any, Any) in module Main at In[8]:8 overwritten

WARNING: Method definition uprime(Any) in module Main at In[8]:24 overwritten at In[9]:19.

WARNING: Method definition fprime(Any) in module Main at In[8]:25 overwritten at In[9]:20.

WARNING: Method definition PFI() in module Main at In[8]:29 overwritten at In[9]:23.

WARNING: Method definition plotPFI() in module Main at In[8]:50 overwritten at In[9]:41.

Out[9]: PyObject <matplotlib.text.Text object at 0x32da2b690>

### 3 Projection Methods

- Many applications require us to solve for an *unknown function*
  - ODEs, PDEs
  - Pricing functions in asset pricing models

- Consumption/Investment policy functions
- Projection methods find approximations to those functions that set an error function close to zero.

### 3.1 Example: Growth, again

- We stick to our working example from above.
- We encountered the Euler Equation  $g$  for optimality.
- At the true consumption function  $c^*$ ,  $g(k) = 0$ .
- We define the following function operator:

$$\begin{aligned} 0 &= u'(c^*(k)) - \beta u'[c^*(f(k) - c^*(k))]f'(f(k) - c^*(k)) \\ &\equiv (\mathcal{N}(\cdot^*))(k) \end{aligned}$$

- The Equilibrium solves the operator equation

$$0 = \mathcal{N}(c^*)$$

#### 3.1.1 Projection Method example

1. create an approximation to  $c^*$ :

- find

$$\bar{c} \equiv \sum_{i=0}^n a_i k^i$$

which nearly solves

$$\mathcal{N}(\bar{c}) = 0$$

2. Compute Euler equation error function:

$$g(k; a) = u'(\bar{c}(k)) - \beta u'[\bar{c}(f(k) - \bar{c}(k))]f'(f(k) - \bar{c}(k))$$

3. Choose  $a$  to make  $g(k; a)$  small in some sense

- *small in some sense*:
  - Least-squares: minimize sum of squared errors

$$\min_a \int g(k; a)^2 dk$$

- Galerkin: zero out weighted averages of Euler errors
- Collocation: zero out Euler equation errors at grid  $k \in \{k_1, \dots, k_n\}$ :

$$P_i(a) \equiv g(k_i; a) = 0, i = 1, \dots, n$$

### 3.1.2 General Projection Method

1. Express solution in terms of unknown function

$$\mathcal{N}(h) = 0$$

where  $h(x)$  is the equilibrium function at state  $x$

2. Choose a space for approximation
3. Find  $\bar{h}$  which nearly solves

$$\mathcal{N}(\bar{h}) = 0$$

### 3.1.3 Projection method exercise

- suppose we want to find effective supply of an oligopolistic firm in cournot competition.
- We want to know  $q = S(p)$ , how much is supplied at each price  $p$ .
- This function is characterized as

$$p + \frac{S(p)}{D'(p)} - MC(S(p)) = 0, \forall p > 0$$

- Take  $D(p) = p^{-\eta}$  and  $MC(q) = \alpha\sqrt{q} + q^2$ .
- Our task is to solve for  $S(p)$  in

$$p - \frac{S(p)p^{\eta+1}}{\eta} - \alpha\sqrt{S(p)} - S(p)^2 = 0, \forall p > 0$$

- No closed form solution. But collocation works!

### TASK

1. solve for  $S(p)$  by collocation
2. Plot residual function
3. Plot resulting  $mS(p)$  together with market demand and  $m = 1, 10, 20$  for market size.

```
In [ ]: using CompEcon
        using NLSolve
        function proj(n=25)
            alpha = 1.0
            eta   = 1.5
            a     = 0.1
            b     = 3.0
            basis = fundefn(:cheb,n,a,b) # cheby basis
            p     = funnode(basis)[1]   # collocation points
            c0 = ones(n)*0.3 # starting value is crucial!

            # your turn!
            function resid(c::Vector,result::Vector,p,basis,alpha,eta)
                q = funeval(c,basis,p)[1]
```

```

    q2 = zeros(q)
    for i in eachindex(q2)
        if q[i] < 0
            q2[i] = -20.0
        else
            q2[i] = sqrt(q[i])
        end
    end
    result[:] = p.+ q .*((-1/eta)*p.^(eta+1)) .- alpha*q2 .- q.^2
end

# need a closure around it
f_closure(x::Vector,r::Vector) = resid!(x,r,p,basis,alpha,eta)
res = nlsolve(f_closure,c0)

# plot residual function
x = collect(linspace(a,b,501))
y = similar(x)
resid(res.zero,y,x,basis,alpha,eta);
figure()
subplot(121)
plot(x,y)
xlabel("price")
ylabel("residual")
title("residual function")

subplot(122)
y = funeval(res.zero,basis,x)[1]
plot(y,x,label="supply 1")
plot(10*y,x,label="supply 10")
plot(20*y,x,label="supply 20")
d = x.^(-eta)
plot(d,x,label="demand")
legend()
ylabel("price")
xlabel("quantity")
title("Equilibrium")

# plot mS(p) and D(p) for m=1,10,20

return res
end

```

## 4 Endogenous Grid Method (EGM)

- Fast, elegant and precise method to solve consumption/savings problems

- One continuous state variable
- One continuous control variable

$$V(M_t) = \max_{0 < c < M_t} u(c) + \beta EV_{t+1}(R(M_t - c) + y_{t+1})$$

- Here,  $M_t$  is cash in hand, all available resources at the start of period  $t$ 
  - For example, assets plus income.
- $A_t = M_t - c_t$  is end of period assets
- $y_{t+1}$  is stochastic next period income.
- $R$  is the gross return on savings, i.e.  $R = 1 + r$ .
- utility function can be of many forms, we only require twice differentiable and concave.

#### 4.1 EGM after [carroll2006method]

- [carroll2006method] [1] introduced this method.
- The idea is as follows:
  - Instead of using non-linear root finding for optimal  $c$  (see above)
  - fix a grid of possible end-of-period asset levels  $A_t$
  - use structure of model to find implied beginning of period cash in hand.
  - We use euler equation and envelope condition to connect  $M_{t+1}$  with  $c_t$

##### 4.1.1 Recall Traditional Methods: VFI and Euler Equation

- Just to be clear, let us repeat what we did in the beginning of this lecture, using the  $M_t$  notation.

$$V(M_t) = \max_{0 < c < M_t} u(c) + \beta EV_{t+1}(R(M_t - c) + y_{t+1})$$

$$M_{t+1} = R(M_t - c) + y_{t+1}$$

##### 4.1.2 VFI

1. Define a grid over  $M_t$ .
2. In the final period, compute

$$V_T(M_T) = \max_{0 < c < M_t} u(c)$$

3. In all preceding periods  $t$ , do

$$V_t(M_t) = \max_{0 < c_t < M_t} u(c_t) + \beta EV_{t+1}(R(M_t - c_t) + y_{t+1})$$

4. where optimal consumption is

$$c_t^*(M_t) = \arg \max_{0 < c_t < M_t} u(c_t) + \beta EV_{t+1}(R(M_t - c_t) + y_{t+1})$$

### 4.1.3 Euler Equation

- The first order condition of the Bellman Equation is

$$\frac{\partial V_t}{\partial c_t} = 0$$

$$u'(c_t) = \beta E \left[ \frac{\partial V_{t+1}(M_{t+1})}{\partial M_{t+1}} \right] \quad (FOC)$$

- By the Envelope Theorem, we have that

$$\frac{\partial V_t}{\partial M_t} = \beta E \left[ \frac{\partial V_{t+1}(M_{t+1})}{\partial M_{t+1}} \right]$$

by FOC

$$\frac{\partial V_t}{\partial M_t} = u'(c_t)$$

true in every period:

$$\frac{\partial V_{t+1}}{\partial M_{t+1}} = u'(c_{t+1})$$

- Summing up, we get the Euler Equation:

$$u'(c_t) = \beta E [u'(c_{t+1})R]$$

### 4.1.4 Euler Equation Algorithm

1. Fix grid over  $M_t$
2. In the final period, compute

$$c_T^*(M_T) = \arg \max_{0 < c_T < M_T} u(c_T)$$

3. With optimal  $c_{t+1}^*(M_{t+1})$  in hand, backward recurse to find  $c_t$  from

$$u'(c_t) = \beta E [u'(c_{t+1}^*(R(M_t - c_t) + y_{t+1}))R]$$

4. Notice that if  $M_t$  is small, the euler equation does not hold.
  - In fact, the euler equation would prescribe to *borrow*, i.e. set  $M_t < 0$ . This is ruled out.
  - So, one needs to tweak this algorithm to check for this possibility
5. Homework.

## 4.2 The EGM Algorithm

Starts in period  $T$  with  $c_T^* = M_T$ . For all preceding periods:

1. Fix a grid of *end-of-period* assets  $A_t$
2. Compute all possible next period cash-in-hand holdings  $M_{t+1}$

$$M_{t+1} = R * A_t + y_{t+1}$$



- for example, if there are  $n$  values in  $A_t$  and  $m$  values for  $y_{t+1}$ , we have  $\dim(M_{t+1}) = (n, m)$
3. Given that we know optimal policy in  $t + 1$ , use it to get consumption at each  $M_{t+1}$

$$c_{t+1}^*(M_{t+1})$$

4. Invert the Euler Equation to get current consumption compliant with an expected level of cash-on-hand, given  $A_t$

$$c_t = (u')^{-1} (\beta E [u'(c_{t+1}^*(M_{t+1}))R|A_t])$$

5. Current period *endogenous* cash on hand just obeys the accounting relation

$$M_t = c_t + A_t$$

*#Core of a simple implementation*

```
type iidModel <: Model
```

```
  # computation grids
```

```
  avec::Vector{Float64}
```

```
  yvec::Vector{Float64}    # income support
```

```
  ywgt::Vector{Float64}    # income weights
```

```
  # intermediate objects (na,ny)
```

```
  m1::Array{Float64,2}     # next period cash on hand (na,ny)
```

```
  c1::Array{Float64,2}     # next period consumption
```

```
  ev::Array{Float64,2}
```

```
  # result objects
```

```
  C::Array{Float64,2}      # consumption function on (na,nT)
```

```
  S::Array{Float64,2}      # savings function on (na,nT)
```

```
  M::Array{Float64,2}      # endogenous cash on hand on (na,nT)
```

```
  V::Array{Float64,2}      # value function on (na,nT). Optional.
```

```
  Vzero::Array{Float64,1}  # value of saving zero
```

```
end
```

```
function EGM!(m::iidModel,p::Param)
```

```
  # final period: consume everything.
```

```
  m.M[:,p.nT] = m.avec
```

```
  m.C[:,p.nT] = m.avec
```

```
  m.C[m.C[:,p.nT].<p.cfloop,p.nT] = p.cfloop
```

```
  # preceding periods
```

```
  for it in (p.nT-1):-1:1
```

```
    # interpolate optimal consumption from next period on all cash-on-hand states
```

```

# using C[:,it+1] and M[:,it+1], find c(m,it)

tmpx = [0.0; m.M[:,it+1] ]
tmpy = [0.0; m.C[:,it+1] ]
for ia in 1:p.na
    for iy in 1:p.ny
        m.c1[ia+p.na*(iy-1)] = linearapprox(tmpx,tmpy,m.m1[ia+p.na*(iy-1)],1,p.na)
        # m.c1[ia,iy] = linearapprox(tmpx,tmpy,m.m1[ia,iy],1,p.na) # equivalent
    end
end

# get expected marginal value of saving: RHS of euler equation
# beta * R * E[ u'(c_{t+1}) ]
Eu = p.R * p.beta .* up(m.c1,p) * m.ywgt

# get optimal consumption today from euler equation: invert marginal utility
m.C[:,it] = iup(Eu,p)

# floor consumption
m.C[m.C[:,it].<p.cfloor,it] = p.cfloor

# get endogenous grid today
m.M[:,it] = m.C[:,it] .+ m.avec
end
end

```

### 4.3 Discrete Choice EGM

- This is a method developed by Fedor Iskhakov, Thomas Jorgensen, John Rust and Bertel Schjerning.
- Reference: [IskhakovRust2014] [3]
- Suppose we have several discrete choices (like "work/retire"), combined with a continuous choice in each case (like "how much to consume given work/retire").
- Let  $d = 0$  mean to retire.
- Write the problem of a worker as

$$\begin{aligned}
 V_t(M_t) &= \max [v_t(M_t|d_t = 0), v_t(M_t|d_t = 1)] \\
 &\text{with} \\
 v_t(M_t|d_t = 0) &= \max_{0 < c_t < M_t} u(c_t) + \beta EW_{t+1}(R(M_t - c_t)) \\
 v_t(M_t|d_t = 1) &= \max_{0 < c_t < M_t} u(c_t) - 1 + \beta EV_{t+1}(R(M_t - c_t) + y_{t+1})
 \end{aligned}$$

- The problem of a retiree is

$$W_t(M_t) = \max_{0 < c_t < M_t} u(c_t) + \beta EW_{t+1}(R(M_t - c_t))$$

- Our task is to compute the optimal consumption functions  $c_t^*(M_t|d_t = 0)$ ,  $c_t^*(M_t|d_t = 1)$

#### 4.3.1 Problems with Discrete-Continuous Choice

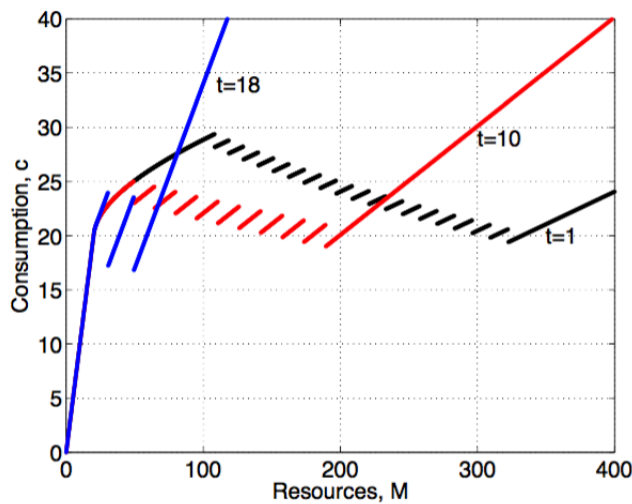
- Even if all conditional value functions  $v$  are concave, the *envelope* over them,  $V$ , is in general not.
- [clausenenvelope] [2] show that there will be a kink point  $\bar{M}$  such that

$$v_t(\bar{M}|d_t = 0) = v_t(\bar{M}|d_t = 1)$$

- We call any such point a **primary kink** (because it refers to a discrete choice in the **current period**)
- $V$  is not differentiable at  $\bar{M}$ .
- However, it can be shown that both left and right derivatives exist, with

$$V^-(\bar{M}) < V^+(\bar{M})$$

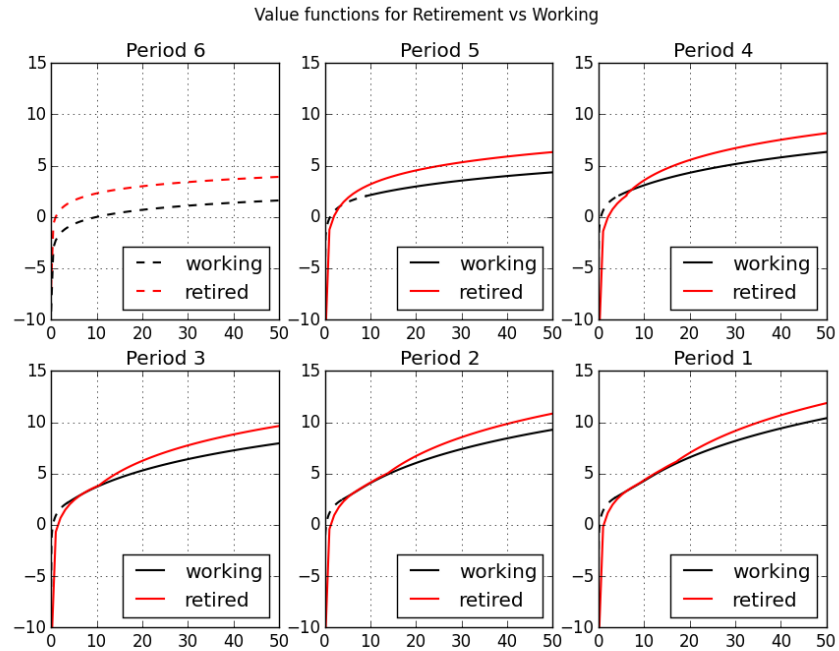
- Given that the value of the derivative changes discretely at  $\bar{M}_t$ , the value function in  $t - 1$  will exhibit a discontinuity as well:
  - $v_{t-1}$  depends on  $V_t$ .
  - Tracing out the optimal choice of  $c_{t-1}$  implies next period cash on hand  $M_t$ , and as that hits  $\bar{M}_t$ , the derivative jumps.
  - The derivative of the value function determines optimal behaviour via the Euler Equation.
  - We call a discontinuity in  $v_{t-1}$  arising from a kink in  $V_t$  a **secondary kink**.
- The kinks propagate backwards.
- [iskhakovRust2014] [3] provide an analytic example where one can compute the actual number of kinks in period 1 of  $T$ .
- Figure 1 in [clausenenvelope]:



[iskhakovRust2014] figure 1

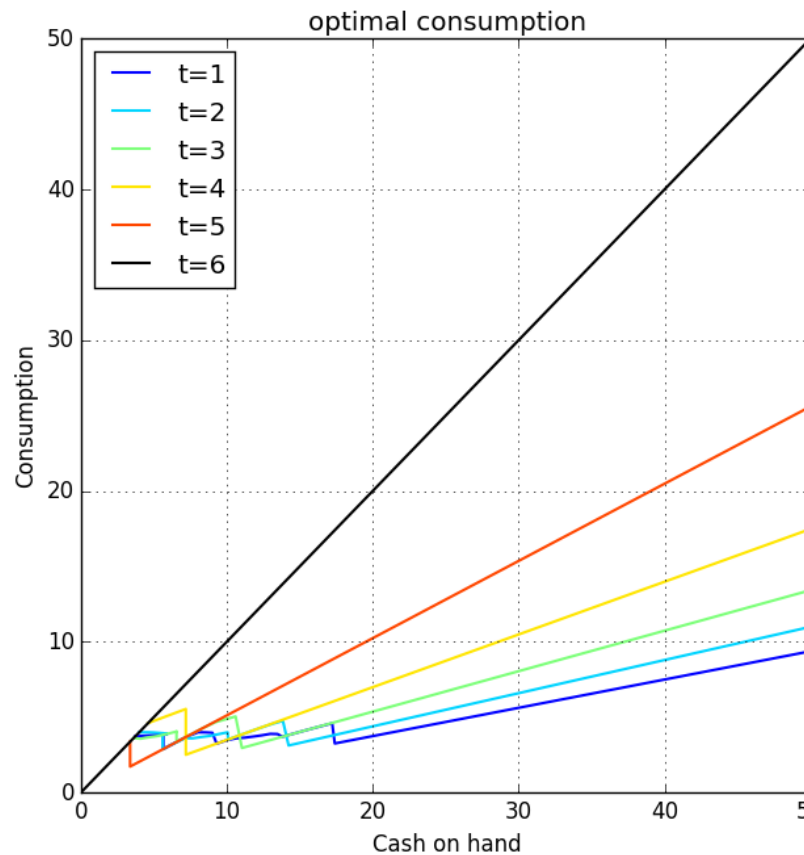
### 4.3.2 Kinks

- Refer back to the work/retirement model from before.
- 6 period implementation of the DC-EGM method:



github/floswald

- Iskhakov @ cemmap 2015: Value functions in T-1
- Iskhakov @ cemmap 2015: Value functions in T-2
- Iskhakov @ cemmap 2015: Consumption function in T-2



- Optimal consumption in 6 period model:

#### 4.3.3 The Problem with Kinks

- Relying on fast methods that rely on first order conditions (like euler equation) will fail.
- There are multiple zeros in the Euler Equation, and a standard Euler Equation approach is not guaranteed to find the right one.
- picture from Fedor Iskhakov's master class at [cemmap 2015](#):

#### 4.3.4 DC-EGM Algorithm

1. Do the EGM step for each discrete choice  $d$
2. Compute  $d$ -specific consumption and value functions
3. compare  $d$ -specific value functions to find optimal switch points
4. Build envelope over  $d$ -specific consumption functions with knowledge of which optimal  $d$  applies where.

#### 4.3.5 But EGM relies on the Euler Equation?!

- Yes.
- An important result in [clausenenvelope] is that the Euler Equation is still the necessary condition for optimal consumption
  - Intuition: marginal utility differs greatly at  $\epsilon + \bar{M}$ .

- No economic agent would ever locate at  $\bar{M}$ .
- This is different from saying that a procedure that tries to find the zeros of the Euler Equation would still work.
  - this will pick the wrong solution some times.
- EGM finds **all** solutions.
  - There is a procedure to discard the "wrong ones". Proof in [iskhakovRust2014]

#### 4.3.6 Adding Shocks

- This problem is hard to solve with standard methods.
- It is hard, because the only reliable method is VFI, and this is not feasible in large problems.
- Adding shocks to non-smooth problems is a widely used remedy.
  - think of "convexifying" in game theoretic models
  - (Add a lottery)
  - Also used a lot in macro
- Adding shocks does indeed help in the current model.
  - We add idiosyncratic taste shocks: Type 1 EV.
  - Income uncertainty:
  - In general, the more shocks, the more smoothing.
- The problem becomes

$$V_t(M_t) = \max [v_t(M_t|d_t = 0) + \sigma_\epsilon \epsilon_t(0), v_t(M_t|d_t = 1) + \sigma_\epsilon \epsilon_t(1)]$$

$$v_t(M_t|d_t = 1) = \max_{0 < c_t < M_t} \log(c_t) - 1 + \beta \int EV_{t+1}(R(M_t - c_t) + y\eta_{t+1})f(d\eta_{t+1})$$

where the value for retirees stays the same.

#### 4.3.7 Adding Shocks

#### 4.3.8 Full DC-EGM

- Needs to discard *false* solutions.
- Criterion:
  - grid in  $A_t$  is **increasing**
  - Assuming concave utility function, the function

$$A(M|d) = M - c(M|d)$$

is **monotone non-decreasing**

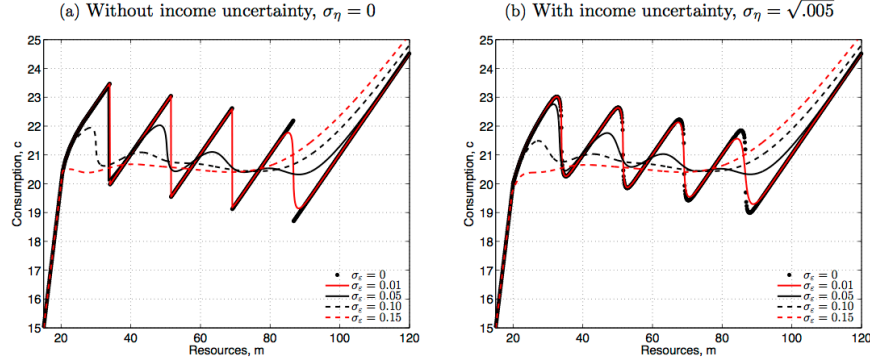
- This means that, if you go through  $A_i$ , and find that

$$M_t(A^j) < M_t(A^{j-1})$$

you know you entered a non-concave region

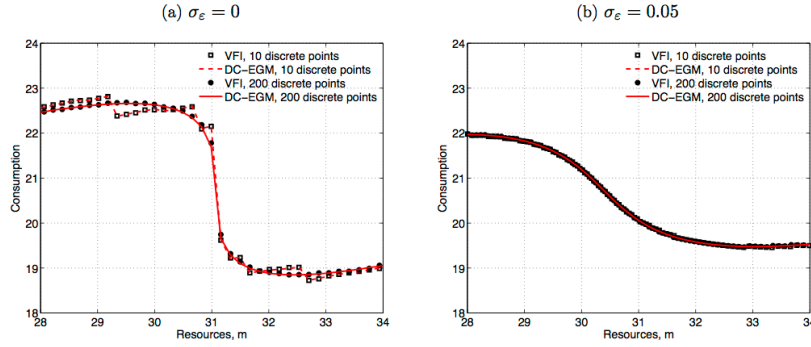
- The Algorithm goes through the upper envelope and *prunes* the *inferior* points  $M$  from the endogenous grids.
- Precise details of Algorithm in paper.
- Julia implementation on [floswald/ConsProb.jl](#)

Figure 2: Optimal Consumption Rules for Agent Working Today ( $d_{t-1} = 1$ ).



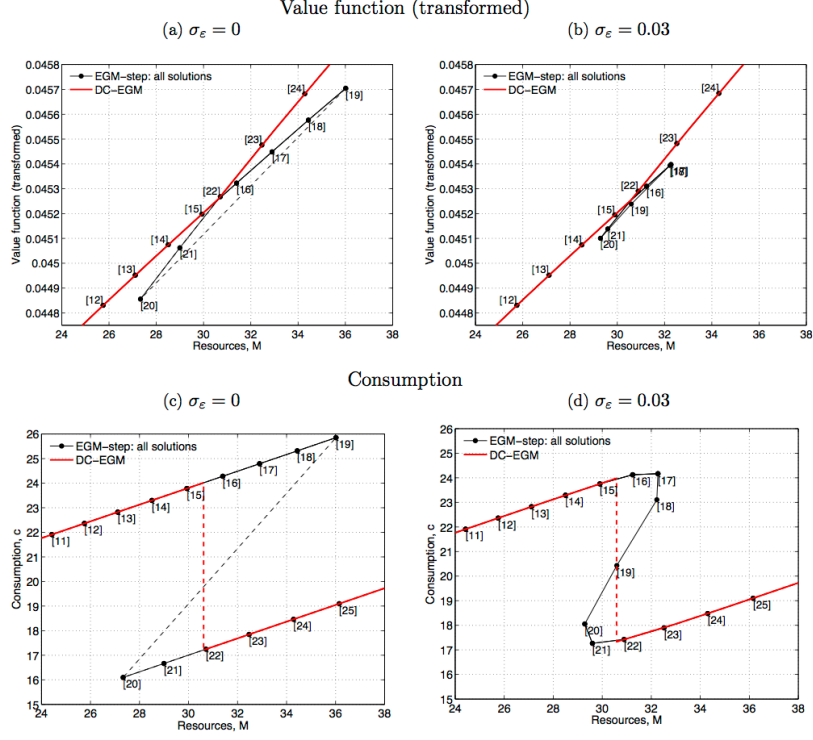
Notes: The plots show optimal consumption rules of the worker who decides to continue working in the consumption-savings model with retirement in period  $t = T - 5$  for a set of taste shock scales  $\sigma_\varepsilon$  in the absence of income uncertainty,  $\sigma_\eta = 0$ , (left panel) and in presence of income uncertainty,  $\sigma_\eta = \sqrt{.005}$ , (right panel). The rest of the model parameters are  $R = 1$ ,  $\beta = 0.98$ ,  $y = 20$ .

Figure 3: Artificial Discontinuities in Consumption Functions,  $\sigma_\eta^2 = 0.01$ ,  $t = T - 3$ .



Notes: Figure 3 illustrates how the number of discrete points used to approximate expectations regarding future income affects the consumption functions from value function iteration (VFI) and the DC-EGM. Panel (a) illustrates how using few (10) discrete equiprobable points to approximate expectations produce severe approximation error when there is *no* taste shocks. Panel (b) illustrates how moderate smoothing ( $\sigma_\varepsilon = .05$ ) significantly reduces this approximation error.

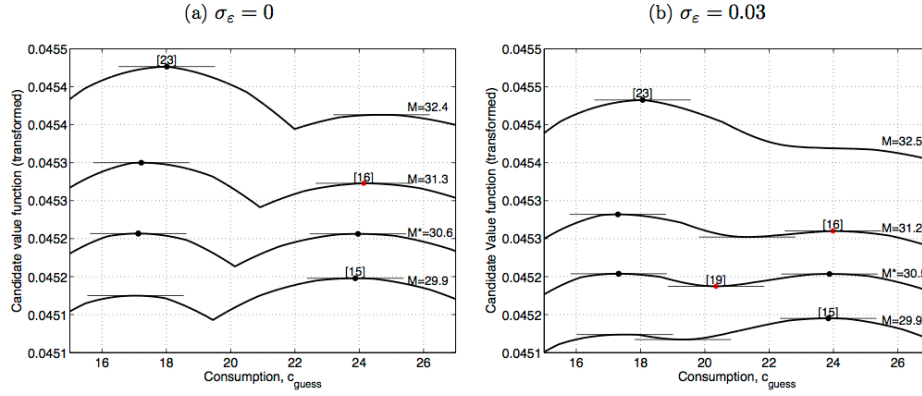
Figure 4: Non-concave regions and the elimination of the secondary kinks in DC-EGM.



Notes: The plots illustrate the output from the EGM-step of the DC-EGM algorithm (Algorithm 1) in a non-concave region. The dots are indexed by the index  $j$  of the ascending grid over the end-of-period wealth  $\vec{A} = \{A^1, \dots, A^G\}$  where  $A^j > A^{j-1}, \forall j \in \{2, \dots, G\}$ . The connecting lines show the  $d_t$ -specific value functions  $v_t(\vec{M}_t|d_t)$  and the consumption function  $c_t(\vec{M}_t|d_t)$  linearly interpolated on the endogenous grid  $\vec{M}_t$ . computed on this grid are the outputs. The left panels illustrate the deterministic case without taste shocks, while in the right panels  $\sigma_\varepsilon = 0.03$ . The “true” solution, after applying the DC-EGM algorithm is illustrated with a solid red line. Dashed lines illustrate discontinuities. The solution is based on  $G = 70$  grid points in  $\vec{A}$ ,  $R = 1$ ,  $\beta = 0.98$ ,  $y = 20$ ,  $\sigma_\eta = 0$ .



Figure 5: Local maxima and multiple solutions of the Euler equation.



Notes: The figure plots the maximand of the equation (10), which defines the discrete choice specific value function  $v_t(M_t|d_t = 1)$ , for the case of  $\sigma_\epsilon = 0$  (panel a) and  $\sigma_\epsilon = 0.03$  (panel b). Horizontal lines indicate the critical points found or approximated by the EGM step of DC-EGM algorithm. The points are indexed with the same indexes as in Figure 4 and the black dots represent global maxima. Model parameters are identical to those of Figure 4.

[@iskhakovRust2014] figure 4

## References

- [1] Christopher D Carroll. The method of endogenous gridpoints for solving dynamic stochastic optimization problems. *Economics letters*, 91(3):312–320, 2006.
- [2] A. Clausen and C. Strub. Envelope theorems for non-smooth and non-concave optimization. <https://andrewclausen.net/research.html>, 2013.
- [3] Fedor Iskhakov, John Rust, Bertel Schjerning, and Thomas Jorgensen. Estimating Discrete-Continuous Choice Models: Endogenous Grid Method with Taste Shocks. *SSRN working paper*, 2014.
- [4] Nancy Stokey and R Lucas. *Recursive Methods in Economic Dynamics (with E. Prescott)*. Harvard University Press, 1989.