Class 5: Dynamic Programming and Estimation

Damian Clarke

February 21, 2020

Research Methods II
MRes. in Economics

UNIVERSITY OF EXETER

Outline

Outline

Choice over time

Dynamic problems have two aspects: stocks and flows.

- The <u>st</u>ate variable summarises <u>st</u>ocks
- The <u>c</u>ontrol variable is the variable being <u>c</u>hosen (ie flows)

$$U = \sum_{t=1}^{T} \beta^{t-1} u(c_t), \tag{1}$$

$$k_{t+1} = f(k_t, c_t). \tag{2}$$

A Dynamic Household

Attach functional forms to (1) and (2):

$$u(c_t) = \ln(c_t) \qquad\qquad k_{t+1} = k_t - c_t$$

Then ...

A Dynamic Household

$$\max_{\{c_t\}_1^T} \sum_{t=1}^{T} \beta^{t-1} \ln(c_t) \qquad \text{s.t.} \qquad \begin{gathered} \textstyle\sum_{t=1}^{T} c_t + k_{T+1} = k_1 \\ c_t \geq 0 \\ k_t \geq 0. \end{gathered} \tag{3}$$

MATLABbing it

We should be able to solve this problem by “direct attack” in MATLAB

- A function to maximise
- A vector of maximands
- A vector of upper and lower bounds
- A(n) (in)equality constraint
- our old friend `fmincon`

```
function V = flowutility(T,Beta,C)
% flowutility(T,Beta,C) takes T periods of
% consumption of size C (a Tx1 vector), and
% calculates the total utility of consumption
% assuming an additively separable utility
% function and discount rate β.

t  =  [1:1:T];
V  =  Beta.^(t-1)*log(C);
V  =  -V;

return
```

Trying this out...

```
>> Beta  = 0.9;
>> T     = 10;
>> k1    = 100;
>> lb    = eps*ones(10,1);
>> ub    = 100*ones(10,1);
>> guess = 10*ones(10,1);
>> A     = ones(1,10);
>> opt   =     optimset('TolFun', 1E-20, 'TolX', 1E-20, ...
                         'algorithm', 'sqp');
>> c  = fmincon(@(C) flowutility(T,Beta,C), guess, ...
                  A, k1, [], [], lb, ub, [], opt);
```

Sensitivity

We have assumed a particular functional form, and values for input parameters

- Here we are imposing these, rather than recovering them
- Of course, we can re-solve the model based on alternative assumptions...
  - Alternative values of β
  - Alternative utility functions
  - Alternative forms of the flow equation (see chapter)
- Let's have a look at `consump_graph.m`

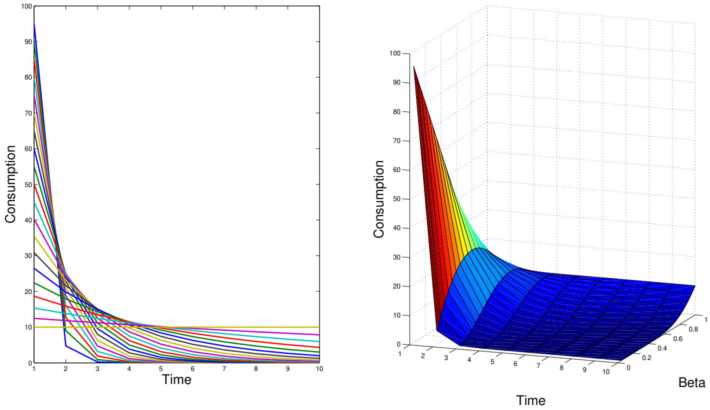

Figure: Sensitivity of Consumption to Discount Rate

A Small Firm

In the readings online we have introduced a more realistic example. Consider a small/family firm.

- This firm is both a producer and a consumer
- Unconsumed capital in period t can be used productively to generate additional capital in $t+1$
- Specifically, let's imagine production is captured by a Cobb-Douglas production function: $k_{t+1} = \theta(k_t - c_t)^{\alpha}$
- Now there is a joint optimization problem over capital and consumption
- For the first time this requires *non-linear* inequality contraints...
- We will not look at this now, but I would encourage you to work through section 6.2.2 of the notes

Outline

Outline

The Bellman Equation

Generally when people speak about 'dynamic programming' in economics, they refer to the class of models solved using value function iteration.

- While solvers like `fmincon` are useful as a general outline, often we need more flexible methods of attack
- This is where the Bellman equation comes in handy
- Essentially, breaks down the problem into sequentially much smaller problems

The Bellman Equation

$$V(k_t) = \max_{c_t}\{u(c_t) + \beta V(k_{t+1})\} \tag{4}$$

Iteration...

So, we can break this down into sub-problems:

$$
\begin{aligned}
V(k_T) &= \max_{c_T} \{u(c_T) + \beta V(k_{T+1})\} \\
V(k_{T-1}) &= \max_{c_{T-1}} \{u(c_{T-1}) + \beta V(k_T)\} \\
V(k_{T-2}) &= \max_{c_{T-2}} \{u(c_{T-2}) + \beta V(k_{T-1})\} \\
&\vdots \\
V(k_2) &= \max_{c_2} \{u(c_2) + \beta V(k_3)\} \\
V(k_1) &= \max_{c_1} \{u(c_1) + \beta V(k_2)\}
\end{aligned} \tag{5}
$$

Iteration II

Now, all we need is a place to start...

$$V(k_{T+1}) = 0 \quad \forall \quad k \tag{6}$$

MATLABbing it

We'll solve Bellman equations numerically with MATLAB

- Essentially, 'brute force' grid search
- Requires 'gridding' state variables (if not binary)
- Let's check out `backwardsInduc.m`

An Example...

```
>> backwards_induc
Input Beta:0.9
Input time: 10
Input initial capital:100
Input fineness of grid:0.25
```

‘Memoization’

A brief final point here: this is computationally intense, but we can avoid a lot of repeated heavy lifting

- ‘Memoization’ (aka computer programming in ‘Nature’)
- This is something that comes in very handy when simulating and solving these problems

Outline

Outline

Uncertainty

What we've seen so far is actually remarkably flexible.

- Generalises quite simply (in theory) to multiple state and control variables, alternative functional forms
- Though in practice, curse of dimensionality
- Perhaps the only major thing we're missing is stochastic elements
- Consider the case where the capital flow equation is now: $k_{t+1} = f(k_t - c_t, \theta, \varepsilon_{t+1}) = \theta(k_t - c_t)^\alpha + \varepsilon_{t+1}$

The Bellman Equation

$$V(k_t) = \max_{c_t}\{u(c_t) + \beta\mathbb{E}[V(k_{t+1})]\} \tag{7}$$

Decisions Under Uncertainty

So, now the decision must framed in terms of consumption now and *expected* consumption in the future.

- In this case, the backwards iteration step is similar
- However, the iterating forwards to solve the model depends upon progressive realisations of shocks
- If time: `finiteStochastic.m`, `simulateStochastic.m`

Simulations

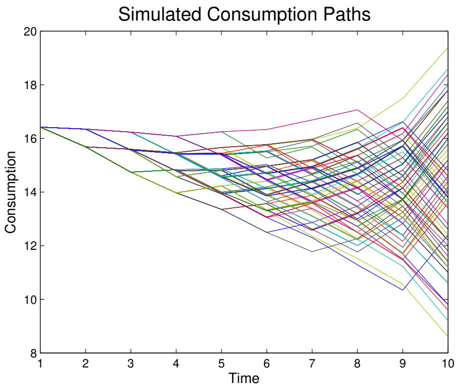

Figure: Simulated Consumption in a Stochastic Model

Summary

Part I (Finite Horizon):

- Finite horizon dynamic optimsation
- Bellman equations
- A little bit of model simulation

Part II (Infinite Horizon):

- Infinte horizons
- Using Bellman again
- Estimation!!

Outline

Outline

Infinite Horizons

With *finite* horizon problems, we could use V_T to seed solution

- We could iterate backwards from the terminal value function, and use this to recursively solve models
- But, in many cases there won't be some obvious end point...
- Fortunately we can still use Bellman's technique if we can find some optimal policy 'forever'
- What's more, the way we solve this is also iterative

Solving Infinite Problems...

$$V(k) = \max_{c}\{u(c) + \beta V(\tilde{k})\} \tag{8}$$

cf:

$$V_t(k_t) = \max_{c_t}\{u(c_t) + \beta V_{t+1}(k_{t+1})\}$$

Finding an Optimal Policy Forever

In (8), $u(c)$ is quite clear, but calculating $V(k)$ is the tricky part

- Value function is the same on both sides of the equation
- So, solving this involves finding a fixed point
- Bellman shows that quite conveniently, starting with any $V_j(k)$, we will eventually iterate onto our stationary $V(k)$

Trying One Out

$$\max_{\{c_t\}_{t=1}^{\infty},\{k_t\}_{t=2}^{\infty}} \sum_{t=1}^{\infty} \beta^{t-1} \ln(c_t) \qquad \text{subject to} \qquad \begin{aligned} k_{t+1} &= \theta k_t^{\alpha} - c_t \\ c_t &\geq 0 \\ k_t &\geq 0. \end{aligned} \tag{9}$$

Trying One Out

Fortunately this *also* gives an analytical solution!

$$V(k) = \frac{\alpha}{1-\beta\alpha}\ln k + F \tag{10}$$

$$c(k) = \theta k^{\alpha}(1-\beta\alpha). \tag{11}$$

If you need convincing, check out appendix to notes...

Trying This Out in MATLAB

The function `Iterate_VF.m` provides a way to make a single iteration on a value function...

- This is all numerical and 'grid search' methods, so is somewhat demanding
- The script `ConvergeGraph.m` runs 10 iterations of the search for a fixed point
- Clearly, 10 iterations is not enough to 'converge' to the fixed point (graph next slide)
- The script `IterateGraph.m` finds 'convergence' (using a `while` loop)
- This is not true convergence. We ask MATLAB to ensure $||V_{j+1}(k) - V_j(k)|| < \varepsilon \ \forall \ k$

Examining a Number of Value Function Iterations

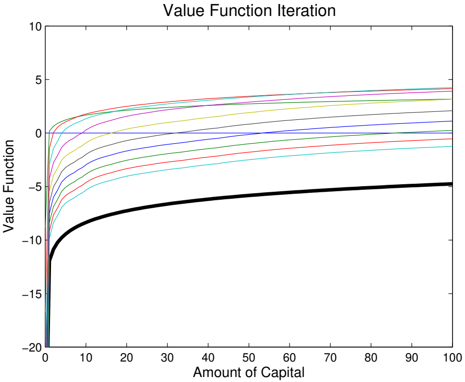

Figure: 10 Iterations of the Numerical Value Function

Iterating to the Value Function

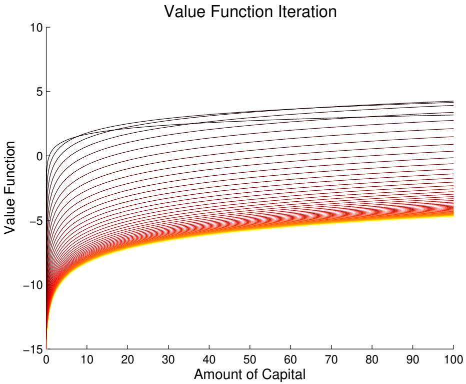

Figure: Convergence to the Numerical Value Function

The Solution

And then, with value function $V(k)$ in hand it's just a matter of determining the optimal policy— the "policy function"—$c(k)$.

```
>> aB = 0.65*0.9; theta = 1.2; alpha = 0.65;
>> plot(K,K(opt),K,aB*theta*K.^alpha, '--r', 'LineWidth', 3)
>> xlabel('Amount of Capital', 'FontSize', 12)
>> ylabel('Optimal k_{t+1}', 'FontSize', 12)
>> title('Policy Function for Capital Consumption', 'FontSize', 14)
```

The Solution

And then, with value function $V(k)$ in hand it's just a matter of determining the optimal policy— the "policy function"—$c(k)$.

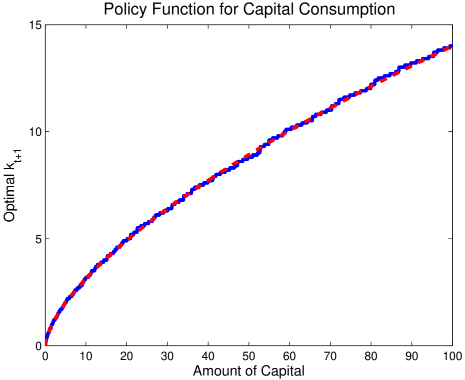

Figure: The Numerical and Analytical Polocy Function

Outline

Outline

The Howard Improvement Algorithm

Value function iteration can be quite slow and computationally intensive

- There are a number of ways to speed up these problems, including 'policy function iteration'
- Our value function iteration takes between 66-135 iterations depending upon ε
- In policy function iteration we follow each proposed solution forever, rather than just for one period

The Algorithm

1. Based upon V_j, determine optimal consumption for each k, giving a proposed 'policy function', $c_j(k)$
2. Calculate the payoff associated with this policy function, $u(c_j(k))$
3. Calculate the value of following this policy function forever, V_{j+1}
4. If $||V_{j+1} - V_j|| < \varepsilon$ stop, or else return to step (i) for another iteration

Remaining Bottlenecks

In step (3):

$$
\begin{aligned}
V_j &= u(c_j(k)) + \beta Q_j V_j \\
\Rightarrow V_j &= (I - \beta Q_j)^{-1} u(c_j(k)), \qquad (12)
\end{aligned}
$$

Policy Function Iteration

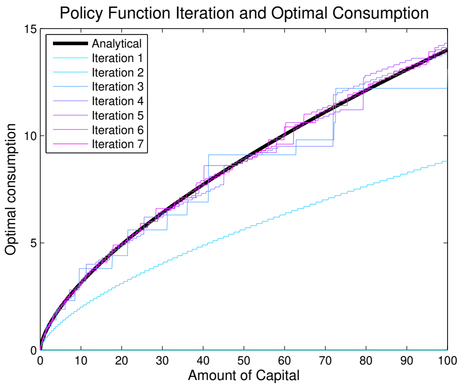

Figure: Convergence to the True Policy Function

Outline

Outline

Changing Viewpoints Entirely...

Forward versus **Inverse** Problems

Generalised Method of Moments

Linear regression one more time...

Population moments:

$$E[\boldsymbol{X}\boldsymbol{\varepsilon}] = E[\boldsymbol{X}(y - \boldsymbol{X}\boldsymbol{\beta})] = \mathbf{0}. \tag{13}$$

And by analogy:

$$\boldsymbol{m} = \frac{1}{N}\left[\sum_{i=1}^{N} \boldsymbol{X_i}(y_i - \boldsymbol{X_i}\boldsymbol{\beta})\right] = \mathbf{0}. \tag{14}$$

Generalised Method of Moments with Linear Regression

We will return one more time to the auto.csv example as a 'sanity check' on GMM coding.

- The idea in GMM is to drive the weighted quadratic distance $\boldsymbol{mWm}'$ to as close as zero as possible
- For consistency, $\boldsymbol{W}$ needs just be semi-definite-positive
- Let's have a look at `objective.m` for moments in this case

Solving the System

```
>> DataIn = dlmread('auto.csv');
>> X      = [ones(74,1) DataIn(:,2:3)];
>> y      = DataIn(:,1);
>> [beta,Q] = fminsearch(@(B) objective(B,y,X),[10,0,0]', ...
          optimset('TolX',1e-9));
```

Outline

Outline

Estimating Dynamic Models

So, going from moments to MATLAB isn't too difficult...

- The challenge is in knowing which moments to fit!
- Let's return to the example from dynamic setting with stochastic elements
- Our moments can be based on $\mathbb{E}[\varepsilon_t] = 0$.

Stochastic Dynamic Model

Remember:

$$\max_{\{c_t\}_{t=1}^{T}} \sum_{t=1}^{T} \beta^{t-1} u(c_t) \qquad \text{subject to} \qquad k_{t+1} = f(k_t, c_t) + \varepsilon_{t+1}. \tag{15}$$

Moments

$$\mathbb{E}[k_{t+1} - f(k_t, c_t)] = 0 \tag{16}$$

$$\mathbb{E}\left[\frac{u'(c_t)}{\beta u'(c_{t+1})} - f'(k_t)\right] = 0. \tag{17}$$

Moments II

$$\mathbb{E}[k_{t+1} - \theta(k_t - c_t)^\alpha] = 0 \tag{18}$$

$$\mathbb{E}\left[\frac{c_{t+1}}{\beta c_t} - \alpha\theta(k_t - c_t)^{\alpha-1}\right] = 0. \tag{19}$$

Estimation

We have set-up these moments in `dynamicMoments.m`. This is a just-identified system (for now). Let's estimate the parameters α and θ based on our simulated example from the end of Finite Horizon estimation:

```
>> finiteStochastic;
>> simulateStochastic;
>> opt = optimset('TolFun', 1E-20, 'TolX', 1E-20);
>> [Omega, Q] = fminunc(@(p) dynamicMoments(con(:,4),con(:,5),...
                kap(:,4),kap(:,5),p),[1, 1], opt);
```