Espen Henriksen
September 18, 2008

A rough guide to recursive methods and numerical dynamic programming

In its simplest form, the prototypical stochastic neoclassical growth model amounts to maximize expected discounted additive separable utility over consumption and leisure subject to the national income and product accounts, ie.

$$\max_{\{c_t, x_t, n_t\}_{t=0}^{\infty}} \mathrm{E}_0 \sum_{t=0}^{\infty} \beta^t u\left(c_t, 1-n_t\right)$$

subject to

$$\begin{aligned}
&\text{Product approach to output:} && y_t = z_t f\left(k_t, n_t\right). && (1)\\
&\text{Income approach to output:} && y_t = r_t k_t + w_t n_t. && (2)\\
&\text{Expenditure approach to output:} && y_t = c_t + x_t. && (3)\\
&\text{Law of motion for capital:} && k_{t+1} = (1-\delta)k_t + x_t. && (4)\\
&\text{Total factor productivity:} && z_t = \rho z_{t-1} + \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(0,\sigma^2), && (5)
\end{aligned}$$

where the exogenous stochastic process, specified in Equation (5), is computed, in final samples, from $\{\tilde{z}_t\}_{t=0}^{T}$ where each $\tilde{z}_t$ is given by

$$\ln \tilde{z}_t = \ln \tilde{y}_t - \alpha \ln \tilde{k}_t - (1-\alpha) \ln \tilde{n}_t.$$

1 Competitive equilibrium

A competitive equilibrium consists of quantities $\{c_t, n_t, x_t\}_{t=0}^{\infty}$ satisfying

1. Household optimization. They supply capital and labor, ie. choose quantities $\{x_t, n_t,\}_{t=0}^{\infty}$ given prices $\{r_t, w_t\}_{t=0}^{\infty}$.
2. Firm optimization. They rent capital and hire labor, ie. choose quantities $\{x_t, n_t,\}_{t=0}^{\infty}$ given prices $\{r_t, w_t\}_{t=0}^{\infty}$.
3. Market clearing. Prices, $\{r_t, w_t\}_{t=0}^{\infty}$, are set such that supply equals demand of capital and labor, $\{x_t, n_t,\}_{t=0}^{\infty}$. By Walras' law, since the markets for capital and labor clear, the market for consumption goods will also clear.

2 First welfare theorem

This model economy satisfies the first welfare theorem – that is any competitive equilibrium is Pareto efficient. That implies that we can solve the centralized planner's problem instead of the decentralized problem. The social planner's problem is much easier to solve since we get rid of the prices and the individuals' budget constraint and instead solve the optimization problem subject to the resource constraint. We can then find the prices that support these equilibrium allocations.

$$\max_{\{c_t,x_t,n_t\}_{t=0}^{\infty}} \mathrm{E}_0 \sum_{t=0}^{\infty} \beta^t u\left(c_t, 1-n_t\right)$$

subject to

$$y_t = z_t f\left(k_t, n_t\right). \tag{6}$$

$$y_t = c_t + x_t. \tag{7}$$

$$k_{t+1} = (1-\delta)k_t + x_t. \tag{8}$$

$$z_t = \rho z_{t-1} + \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(0,\sigma^2), \tag{9}$$

$$k_0 > 0, \quad \text{given.} \tag{10}$$

3 Simplifying the problem

For pedagogical purposes, let's simplify the problem and for a while ignore the labor-leisure choice and the stochastic process for total factor productivity. We set z_t equal to its unconditional mean $\bar{z}$ for all t, and, in addition, we eliminate y_t and x_t by combining Equations (6), (7) and (8). The problem simplifies to

$$\max_{\{c_t,x_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t u\left(c_t\right)$$

subject to

$$c_t + k_{t+1} = \bar{z} f\left(k_t\right) + (1-\delta)k_t.$$
$$k_0 > 0, \quad \text{given.}$$

4 The sequence problem, solving for *allocations*

The mathematical strategy is solve for a sequence of *allocations* by maximizing a function (total utility) of an infinity of variables (consumption and capital at each date) subject to technological constraints (production function and law of motion for capital) from a given initial capital stock k_0.

The Lagrangian equation for the problem is

$$\mathcal{L} = \sum_{t=0}^{\infty} \beta^t \left\{u\left(c_t\right) - \lambda_t \left[c_t + k_{t+1} - \bar{z} f\left(k_t\right) - \left(1-\delta\right) k_t\right]\right\},$$

with transversality condition

$$\lim_{t\to\infty} \beta^t \lambda_t k_{t+1} = 0.$$

Given standard assumptions on the functional forms of f and u, capital stock converge monotonically to the level that, if sustained, maximizes consumption per unit of time.

For the deterministic problem, it is quick and easy to solve the sequence problem. However, if uncertainty is introduced the problem blows up and most likely becomes impossible to solve.

5 The recursive problem, solving for *functions*

The mathematical strategy is to seek the optimal savings/investment *function* directly and then to use this function to compute the optimal sequence of investments from any initial capital stock. This way of looking at the problem – decide on the immediate action to take as a function of the current situation – is called a *recursive formulation.* It exploits the observation that a decision problem of the same structure *recurs* every period.

For the deterministic problem, this might seem like an unnecessarily hard way to solve the problem. However, as we shall see, if uncertainty is introduced the structure of the problem hardly changes and we can still solve it with relative ease.

Our problem is still

$$\max_{\{c_t, k_t+1\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t u\left(c_t\right)$$

subject to

$$c_t + k_{t+1} = \bar{z} f\left(k_t\right) + (1-\delta) k_t.$$
$$k_0 > 0, \quad \text{given.}$$

Substituting in for c_t we have the following problem

$$\max_{\{k_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t u\left[\bar{z} f\left(k_t\right) + (1-\delta) k_t - k_{t+1}\right],$$

or

$$\max_{\{k_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t u\left(k_t, k_{t+1}\right),$$

both given

$$k_0 > 0.$$

Starting from an arbitrary time t, the problem is

$$\max_{\{k_{s+1}\}_{s=t}^{\infty}} \sum_{s=t}^{\infty} \beta^{s-t} u\left(k_s, k_{s+1}\right). \tag{11}$$

given

$$c_s + k_{s+1} = \bar{z} f\left(k_s\right) + (1-\delta) k_s \qquad \forall s \geq t \tag{12}$$
$$k_t > 0 \qquad \text{given} \tag{13}$$

5.1 Value function

Let us define $v(k_t)$ as the indirect utility function, ie. function that attains the value of the optimal program from period t and onwards given an initial condition k_t

$$v(k_t) \equiv \max_{\{k_{s+1}\}_{s=t}^{\infty}} \sum_{s=t}^{\infty} \beta^{s-t} u\left(k_s, k_{s+1}\right).$$

Using the maximization-by-steps idea, we can write

$$\begin{aligned}
v(k_t) &\equiv \max_{k_{t+1}} \left\{ u\left(k_t, k_{t+1}\right) + \max_{\{k_{s+1}\}_{s=t+1}^{\infty}} \sum_{s=t+1}^{\infty} \beta^{s-t} u\left(k_s, k_{s+1}\right) \right\} \\
&= \max_{k_{t+1}} \left\{ u\left(k_t, k_{t+1}\right) + \beta \max_{\{k_{s+1}\}_{s=t+1}^{\infty}} \sum_{s=t+1}^{\infty} \beta^{s-(t+1)} u\left(k_s, k_{s+1}\right) \right\} \\
&\equiv \max_{k_{t+1}} \left\{ u\left(k_t, k_{t+1}\right) + \beta v\left(k_{t+1}\right) \right\}. \\
&= \max_{k_{t+1}} \left\{ u\left(c_t\right) + \beta v\left(k_{t+1}\right) \right\}
\end{aligned}$$

We have derived the Bellman equation

$$v(k_t) = \max_{k_{t+1}} \left\{ u\left(c_t\right) + \beta v\left(k_{t+1}\right) \right\}, \tag{14}$$

where

$$c_t = \bar{z} f\left(k_t\right) + (1-\delta)k_t - k_{t+1}. \tag{15}$$

The Bellman equation captures a key element of most intertemporal decision, which most of our decisions are, namely that they are a trade-off between instantaneous utility $u(c_t)$ and discounted continuation utility $\beta v\left(k_{t+1}\right)$.

The only *state variable* of our recursive problem is k_t. The variable k_t captures all relevant information for making a decision. Since k_t depends on past decision, we classify it as an *endogenous* state variable.

The only *decision* or *control variable* of our recursive problem is k_{t+1}. Equivalently, we could instead have defined c_t or x_t as our control/decision variable.

5.2 Recursive notation

We will call a problem *stationary* whenever the structure of the choice problem that a decision maker faces is identical at every point in time, ie. only relative time, not absolute time, is relevant. In order to make sure we are not using notation that might indicate otherwise we write the Bellman equation using recursive notation

$$v(k) = \max_{k'} \left\{ u\left(c\right) + \beta v\left(k'\right) \right\}, \tag{16}$$

where

$$c = \bar{z} f\left(k\right) + (1-\delta)k - k'. \tag{17}$$

5.3 Properties of the decision rule

Given the true (but potentially unknown) value function $v\left(\cdot\right)$, we can study the optimality conditions of the problem. The unique optimal time-invariant decision rule is defined by the first-order condition of the maximization problem, defined by the right-hand-side of the Bellman equation (16), *with respect to the control variable*, in this case k'

$$\frac{\partial u\left(c\right)}{\partial c} \frac{\partial c}{\partial k'} + \beta \frac{\partial v\left(k'\right)}{\partial k'} = 0,$$

and since $\partial c/\partial k' = -1$

$$\frac{\partial u(c)}{\partial c} = \beta \frac{\partial v(k')}{\partial k'}. \tag{18}$$

Application of the envelope theorem, also known as the Benveniste and Scheinkman (1979) condition, which generally holds off corners, amounts to taking the derivative of the problem *with respect to the endogenous state variable*, in this case k

$$\frac{\partial v(k)}{\partial k} = \frac{\partial u(c)}{\partial c}\frac{\partial c}{\partial k}.$$

Since $\partial c/\partial k = \bar{z}\partial f(k)/\partial k + (1-\delta)$

$$\frac{\partial v(k)}{\partial k} = \frac{\partial u(c)}{\partial c}\left(\bar{z}\frac{\partial f(k)}{\partial k} + 1 - \delta\right). \tag{19}$$

Since we have a stationary, recursive problem, given Equation (20) we also have

$$\frac{\partial v(k')}{\partial k'} = \frac{\partial u(c')}{\partial c'}\left(\bar{z}\frac{\partial f(k')}{\partial k'} + 1 - \delta\right). \tag{20}$$

Combining Equations (18) and (7.4) and eliminating $\partial v(k')/\partial k'$, we have

$$\frac{1}{\beta}\frac{\partial u(c)}{\partial c} = \frac{\partial u(c')}{\partial c'}\left(\bar{z}\frac{\partial f(k')}{\partial k'} + 1 - \delta\right),$$

or equivalently

$$\beta\frac{\partial u(c')/\partial c'}{\partial u(c)/\partial c}\left(\bar{z}\frac{\partial f(k')}{\partial k'} + 1 - \delta\right) = 1.$$

This is the intertemporal optimality condition, also known as the Euler equation.

5.4 Steady state

In steady state, $k = k' = \bar{k}$, $c = c' = \bar{c}$, etc. The Euler equation becomes

$$\beta\left(\bar{z}\frac{\partial f(\bar{k})}{\partial \bar{k}} + 1 - \delta\right) = 1,$$

from which we can solve for $\bar{k}$ as a function of the structural parameters and $\bar{z}$.

5.5 Contraction mapping theorem

In general, the function $v(\cdot)$ is unknown, but the Bellman equation can be used to find it. In most of the cases we will deal with, the Bellman equation satisfies a contraction mapping theorem, which implies that

1. There is a unique function $v(\cdot)$ which satisfies the Bellman equation.
2. If we begin with any initial *guess* for the *function* v, let's call it $v_0(k)$ and define

   $$v_{n+1}(k) = \max_{c,k'}\left[u(c) + \beta v_n(k')\right]$$

   subject to

   $$c + k' = f(k) + (1-\delta)k$$

for $n = 0, 1, 2, ...$ then $\lim_{n\to\infty} v_{n+1}(k) = v(k)$

In other words, if we iterate on the Bellman equation and for each iteration update the guess for the function v, we will eventually converge to the true *function* v.

The above two implications give us two alternative means of uncovering the value function.

First, given implication 1 above, if we are fortunate enough to correctly guess the value function $v(\cdot)$ then we can simply plug $v(k')$ into the right side and then verify that $v(k)$ solves the Bellman equation. This procedure only works in a very few and very special cases.

Second, implication 2 above is useful for doing numerical work. One approach is to find an approximation to the value function and iterate. It is almost like it begs to be programmed.

6 Discrete deterministic value function iteration

Numerical deterministic value function iteration amounts to two choices of numerical methods: how to *approximate* the value function and how to *optimize* the object inside the curly brackets:

$$\underbrace{v(k)}_{\text{approximation}} = \underbrace{\max_{k'}\left\{u(c) + \beta \underbrace{v(k')}_{\text{approximation}}\right\}}_{\text{optimization}}$$

With discrete deterministic value function iteration, the value function is numerically approximated by *a vector of discrete points*, ie. both the state variable k and the decision variable k' can just take a set of discrete values.

Numerical optimization is then almost trivial and will simply amount to for each k pick the k' which gives the highest value for $\left\{u(c) + \beta v(k')\right\}$

6.1 Computational implementation

1. Initialize the algorithm
   (a) Define the *calibrated* structural parameters of the model economy.
   (b) Compute the steady state value of the capital stock k^*
   (c) Set g_k (number of grid points), $\underline{k}$ (lower bound of the state space), $\bar{k}$ (upper bound of the state space), and ε (tolerance of error).
   g_k is determined weighting the tradeoff between speed and precision.

   Pseudo-code:

```
$\alpha \leftarrow .35$
$\beta \leftarrow .98$
$\delta \leftarrow .025$
$\sigma \leftarrow 2$
$\bar{z} \leftarrow 5$
$k^* \leftarrow \left(\frac{\frac{1}{\beta}-(1-\delta)}{\alpha\bar{z}}\right)^{\frac{1}{\alpha-1}}$
```

```
$\underline{k} \leftarrow .95k^*$
$\overline{k} \leftarrow 1.05k^*$
$g_k \leftarrow 101$
$\varepsilon \leftarrow 10^{-8}$
```

Matlab implementation:

```
clear all;
close all;
alpha = .35;
beta  = .98;
delta = .025;
sigma = 2;
zbar = 5;
kstar = ((1/beta - 1 + delta)/(alpha*zbar))^(1/(alpha-1));
kmin = 0.95*kstar;
kmax = 1.05*kstar;
gk = 101;
epsilon = 1E-8;
```

Note that the values assigned to the variables in this example are *arbitrary*. In this example, they are *not* calibrated, but are simply added to illustrate the algorithm.

2. Given the bounds of the state space, set the grid points $\{k_1, k_2, \ldots, k_{g_k}\}$. Default is equidistanced grid points. The value function will be approximated as a $(g_k \times 1)$-dimensional vector v.

   Pseudo-code

```
for $i = 1$ to $g_k$ do
    $k_i \leftarrow \underline{k} + \frac{\overline{k}-\underline{k}}{g_k - 1}(i-1)$
end for
```

   Matlab implementation:

```
for i = 1 : gk
    k(i) = kmin + (kmax - kmin) / (gk - 1) * (i - 1);
end
```

   An alternative Matlab implementation:

```
k = linspace(kmin,kmax,gk);
```

3. Construct consumption and welfare matrices. Compute a $(g_k \times g_k)$ dimensional consumption matrix c with the value of consumption for all the $(g_k \times g_k)$ combinations of k and k'. Then compute a $(g_k \times g_k)$-dimensional welfare matrix u with the utility of consumption for all the $(g_k \times g_k)$ combinations of k and k'.

   Pseudo-code:

```
for $i = 1$ to $g_k$ do
    for $j = 1$ to $g_k$ do
        $c_{i,j} \leftarrow \bar{z}\,(k_i)^\alpha + (1-\delta)k_i - k'_j$
        if $c_{i,j} < 0$ then
            $c_{i,j} \leftarrow 0$
        end if
      end for
    end for
    for $i = 1$ to $g_k$ do
      for $j = 1$ to $g_k$ do
        if $\sigma = 1$ then
          $u(c_{i,j}) \leftarrow \ln(c_{i,j})$
        else
          $u(c_{i,j}) \leftarrow \frac{(c_{i,j})^{1-\sigma}-1}{1-\sigma}$
        end if
      end for
    end for
```

Matlab implementation:

```
for i = 1 : gk
    for j = 1 : gk
        c(i,j) = zbar*k(i)^alpha + (1-delta)*k(i) - k(j);
        if c(i,j) < 0
            c(i,j) = 0;
        end
   end
end
clear i j

for i = 1 : gk
   for j = 1 : gk
        if sigma == 1
            u(i,j) = log(c(i,j));
        else
            u(i,j) = (c(i,j)^(1-sigma) - 1)/(1-sigma);
        end
   end
end
clear i j
```

4. Set an initial value of $v^0 = \left\{v_i^0\right\}_{i=1}^{g_k}$. A trivial initial condition is $v^0 = 0$. Also initialize Tv

   Pseudo-code:

```
    for $i = 1$ to $g_k$ do
      $v_i \leftarrow 0$
      $Tv_i \leftarrow 0$
    end for
```

   Matlab implementation:

```
v  = zeros(gk,1);
Tv = zeros(gk,1);
```

5. Update the value function and obtain Tv. More specifically, do the following steps for each of $i = 1, \ldots, g_k$.

   (a) Solve the following problem

$$d_i^k = \underset{k' \in \{k_j\}_{j=1}^{g_k}}{\operatorname{argmax}} \left\{u(k_i, k') + \beta v(k')\right\}$$

For later use, also store $d_i^g \leftarrow \text{argmax}_{j\in\{1,\dots,g_k\}}\left\{u\left(k_i,k_j\right)+\beta v\left(k_j\right)\right\}$

(b) Once d_i is obtained, use it to update value function. Specifically:

$$Tv_i = u\left(k_i, d_i^k\right) + \beta v\left(d_i^k\right)$$

After implementing the procedure above for $i = 1, \dots, g_k$, we have constructed a new (discretized) approximation for the value function as $Tv = \{Tv_i\}_{i=1}^{g_k}$.

6. Compare $\{v_i\}_{i=1}^{g_k}$ and $\{Tv_i\}_{i=1}^{g_k}$ and compute the distance w. One way to define the error is to use maximum distance, as follows:

$$w = \max_i |v_i - Tv_i|$$

If $w > \varepsilon$, the error is not small enough. Update the value function using: $v = Tv$ and go back to Step 5.

Pseudo-code:

```
while $w > \varepsilon$ do
  for $i = 1$ to $g_k$ do
    $d_i^k \leftarrow \text{argmax}_{k'\in\{k_j\}_{j=1}^{g_k}}\left\{u\left(k_i,k'\right)+\beta v\left(k'\right)\right\}$
    $d_i^g \leftarrow \text{argmax}_{j\in\{1,\dots,g_k\}}\left\{u\left(k_i,k_j\right)+\beta v\left(k_j\right)\right\}$
    $Tv_i \leftarrow u\left(k_i,d_i^k\right)+\beta v\left(d_i^k\right)$
  end for
  $w = \max_i |v_i - Tv_i|$
  $v = Tv$
end while
```

Matlab implementation:

```
while w > epsilon
    for i = 1 : gk
        [y,j] = max( u(i,:) + beta*v');
        dk(i) = k(j);
        dg(i) = j;
        Tv(i) = u(i,dg(i)) + beta*v(dg(i));
    end
    w = max(abs(Tv - v));
    v = Tv;
end
```

7. If $w < \varepsilon$, then we find our optimal value *function*. The value function is approximated by the vector $v = \{v_i\}_{i=1}^{g_k}$. The optimal decision rule is approximated by the vector $d^k = \left\{d_i^k\right\}_{i=1}^{g_k}$.

Given d^k we can compute the optimal decision rule for consumption $d^c = \{d_i^c\}_{i=1}^{g_k}$.

Pseudo code:

```
for $i = 1$ to $g_k$ do
  $d_i^c \leftarrow \bar{z}\left(k_i\right)^{\alpha} + (1-\delta)k_i - d_i^k$
end for
```

Matlab implementation:

```
for i = 1 : gk
    dc(i) = zbar*k(i)^alpha + (1-delta)*k(i) - dk(i);
end
```

6.2 Simulate the model

If we want to simulate the model for T periods from any given initial capital stock $k_s \in \{k_j\}_{j=1}^{g_k}$, the easiest is to use the grid-representation of the optimal decision rule for next-period capital; d^g.

Pseudo code:

```
Require: $\{k_s, i\}$, $d^g$
Ensure: $\{\hat{k}_t\}_{t=1}^{T+1}$, $\{\hat{c}_t\}_{t=1}^{T}$
  for $t = 1$ to $T$ do
    $j \leftarrow d_i^g$
    $\hat{k}_{t+1} \leftarrow k(j)$
    $\hat{c}_t \leftarrow f(\hat{k}_t) + (1-\delta)\hat{k} - \hat{k}_{t+1}$
    $i \leftarrow j$
  end for
```

7 Reintroducing uncertainty

An AR(1) process like the one specifying the exogenous law of motion for z_{t+1} in Equation (5) is an example of a continuous process which has the Markov property. Loosely speaking, having the Markov property means that the probability distribution of the current state is conditionally independent of the path of past states – ie. no memory beyond the present. A discrete two-state Markov chain is an example of a simpler process with the Markov property.

We take the problem from Equations ..., but let z_s be stochastic:

$$\max_{\{k_{s+1}\}_{s=t}^{\infty}} \mathrm{E}_t \sum_{s=t}^{\infty} \beta^{s-t} u(c_s). \tag{21}$$

subject to

$$c_s + k_{s+1} = z_s f(k_s) + (1-\delta)k_s \qquad \forall s \geq t \tag{22}$$

$$k_t > 0 \qquad \text{given.} \tag{23}$$

where z_s can attain two values z_H (high productivity) and z_L (low productivity). z_t follows a two-state Markov chain with $\Pr(z_{s+1} = z_H \mid z_s = z_H) = p$ and $\Pr(z_{s+1} = z_L \mid z_s = z_L) = q$.

Denoting the Markov transition matrix Π, we have

$$\Pi = \begin{pmatrix} p & 1-p \\ 1-q & q \end{pmatrix}.$$

7.1 The sequence problem, solving for *allocations*

Introduction of a very simple exogenous stochastic process – the entire problem "blows up".

7.2 The recursive problem, solving for *functions*

Introduction of exogenous uncertainty hardly changes the problem

$$v(k_t, z_t) = \max_{k_{t+1}} \left\{ u(c_t) + \beta \mathrm{E}_{z_{t+1}|z_t} v(k_{t+1}, z_{t+1}) \right\},$$

where

$$c_t = z_t f(k_t) + (1-\delta)k_t - k_{t+1}.$$

Our recursive problem has now two state variables: k_t and z_t. Together these two variables capture all relevant information for making a decision. Since z_t does *not* depend on past decision, we classify it as an *exogenous* state variable. k_t is still an endogenous state variable.

The only *decision* or *control variable* of our recursive problem is still k_{t+1} (or equivalently c_t or x_t).

7.3 Recursive notation

$$v(k, z) = \max_{k'} \left\{ u(c) + \beta \mathrm{E}_{z'|z} \left[v(k', z') \right] \right\},$$

where

$$c = zf(k) + (1-\delta)k - k'.$$

7.4 Properties of the decision rule

As before, we can take *the first order condition with respect to the control variable* (in this case k')

$$\frac{\partial u(c)}{\partial c} \cdot (-1) + \beta \mathrm{E}_{z'|z} \left[\frac{\partial v(k', z')}{\partial k'} \right] = 0,$$

and the envelope condition *with respect to the endogenous state variable* (in this case k)

$$\frac{\partial v(k, z)}{\partial k} = \frac{\partial u(c)}{\partial c} \left(z \frac{\partial f(k)}{\partial k} + 1 - \delta \right).$$

Updating the latter, combining these two equations and eliminating $\partial v(k', z') / \partial k'$, we have

$$\frac{1}{\beta} \frac{\partial u(c)}{\partial c} = \mathrm{E}_{z'|z} \left[\frac{\partial u(c')}{\partial c'} \left(z' \frac{\partial f(k', z')}{\partial k'} + 1 - \delta \right) \right],$$

or equivalently

$$\beta \mathrm{E}_{z'|z} \left[\frac{\partial u(c') / \partial c'}{\partial u(c) / \partial c} \left(z' \frac{\partial f(k', z')}{\partial k'} + 1 - \delta \right) \right] = 1.$$

7.5 Discrete stochastic value function iteration

Numerical stochastic value function iteration amounts to three choices of numerical methods: how to *approximate* the value function, how to *integrate* in order to compute the expectation, and how to *optimize* the object inside the curly brackets:

$$\underbrace{v\left(k,z\right)}_{\text{approximation}} = \underbrace{\max_{k'}\left\{u\left(c\right)+\beta\,\underbrace{\mathrm{E}_{z'|z}\left[\ \underbrace{v\left(k',z'\right)}_{\text{approximation}}\ \right]}_{\text{integration}}\right\}}_{\text{optimization}}$$

With discrete stochastic value function iteration, the value function is numerically approximated by *a vector of discrete points*, ie. both the state variables k and z and the decision variable k' can only take a set of discrete values.

Both numerical integration and numerical optimization are then almost trivial. Integration will simply amount to weighting discrete events by discrete probabilities and optimization is simply for each combination (k,z) to pick the k' which gives the highest value for the expression within the curly brackets.

8 Reintroducing labor-leisure choice

$$\max_{\{k_{t+1},n_t\}_{t=s}^{\infty}} \mathrm{E}_s \sum_{t=s}^{\infty} \beta^{t-s} u\left(c_t, 1-n_t\right). \tag{24}$$

subject to

$$c_t + k_{t+1} = z_t f\left(k_t,n_t\right) + (1-\delta)k_t \qquad \forall t \geq s \tag{25}$$

$$k_s > 0 \qquad \text{given.} \tag{26}$$

where z_t follows a two-state Markov chain with $\Pr\left(z_{t+1}=z_H \mid z_t=z_H\right)=p$ and $\Pr\left(z_{t+1}=z_L \mid z_t=z_L\right)=q$.

8.1 The recursive problem, solving for ***functions***

Again, the problem is hardly changed

$$v(k_t,z_t) = \max_{k_{t+1},n_t} \left\{u\left(c_t,1-n_t\right)+\beta \mathrm{E}_{z_{t+1}|z_t} v\left(k_{t+1},z_{t+1}\right)\right\},$$

where

$$c_t = z_t f\left(k_t,n_t\right)+(1-\delta)k_t - k_{t+1}.$$

Our recursive problem has still two state variables: k_t and z_t. No additional variables are necessary to capture all relevant information for making a decision. However, we have now two *decision* or *control variables*: k_{t+1} and n_t.

8.2 Recursive notation

$$v(k,z) = \max_{k',n}\left\{u\left(c,1-n\right)+\beta \mathrm{E}_{z'|z} v\left(k',z'\right)\right\},$$

where

$$c = zf\left(k,n\right)+(1-\delta)k-k'.$$

8.3 Properties of the decision rule

As before, we can take the first order condition with respect to the control variables k'

$$\frac{\partial u(c, 1-n)}{\partial c} \cdot (-1) + \beta \mathrm{E}_{z'|z}\left[\frac{\partial v(k', z')}{\partial k'}\right] = 0, \tag{27}$$

and n,

$$\frac{\partial u(c, 1-n)}{\partial c}\frac{\partial c}{\partial n} + \frac{\partial u(c, 1-n)}{\partial n} = 0, \tag{28}$$

and the envelope condition with respect to the endogenous state variable k

$$\frac{\partial v(k, z)}{\partial k} = \frac{\partial u(c)}{\partial c}\left(z\frac{\partial f(k)}{\partial k} + 1 - \delta\right). \tag{29}$$

Equation (28) is our *intratemporal optimality condition.* Reorganizing it gives us

$$\frac{\partial u(c, 1-n)/\partial n}{\partial u(c, 1-n)/\partial c} = z\frac{\partial f(k, n)}{\partial n}$$

Updating Equation (29), combining it with Equation (27) and eliminating $\partial v(k', z')/\partial k'$, gives us (as before) *the intertemporal optimality condition*

$$\beta \mathrm{E}_{z'|z}\left[\frac{\partial u(c')/\partial c'}{\partial u(c)/\partial c}\left(z'\frac{\partial f(k', n')}{\partial k'} + 1 - \delta\right)\right] = 1.$$

8.4 Discrete stochastic value function iteration

Again everything follows through almost exactly as in the deterministic case without labor-leisure choice... to be continued ...

9 Piecewise-linear interpolation of the value fn

As we saw in the previous section, numerical deterministic value function iteration amounts to two choices of numerical methods: how to *approximate* the value function and how to *optimize* the sum of instantaneous return (utility) and the discounted continuation value.

In this case, the value function is numerically approximated by *piecewise linear interpolation*, ie. whereas we will still evaluate the function at a set of discreet notes for k, the decision variable k' can now take any value within the permissable range.

For the numerical optimization part, we need a routine that ... a continuous, *but not* continuously differentiable function. A natural choice is golden section search.

9.1 Computational implementation

1. Initialize the algorithm

   (a) Define the *calibrated* structural parameters of the model economy.

   (b) Compute the steady state value of the capital stock k^*

   (c) Set g_k (number of grid points), $\underline{k}$ (lower bound of the state space), $\bar{k}$ (upper bound of the state space), and ε (tolerance of error).
   g_k is determined weighting the tradeoff between speed and precision.

   Pseudo-code:

   ```
   $\alpha \leftarrow .35$
   $\beta \leftarrow .98$
   $\delta \leftarrow .025$
   $\sigma \leftarrow 2$
   $\bar{z} \leftarrow 5$
   $k^* \leftarrow \left(\frac{\frac{1}{\beta}-(1-\delta)}{\alpha\gamma}\right)^{\frac{1}{\alpha-1}}$
   $\underline{k} \leftarrow .95k^*$
   $\bar{k} \leftarrow 1.05k^*$
   $g_k \leftarrow 11$
   $\varepsilon \leftarrow 10^{-8}$
   ```

   Note that the values assigned to the variables in this example are *arbitrary*. In this example, they are *not* calibrated, but are simply added to illustrate the algorithm.

2. Given the bounds of the state space, set the *knots* $\{k_i\}_{i=1}^{g_k}$. Default is to set equidistance grid points. Given the knots, the value function, which is approximated using piecewise-linear interpolation, can be stored as an array of length g_k. Let's denote the value function as $\{v_i\}_{i=1}^{g_k}$

3. (a) Initialize the algorithm to evaluate d given the piecewise-linear interpolation defined by $\{k_i, v_i\}_{i=1}^{g_k}$
   Pseudo-code:

```
Require: $d$ and $\{k_i, v_i\}_{i=1}^{g_k}$
Ensure: $y$
  for $i = 1$ to $g_k - 1$ do
    if $d \geq k_i$ and $d < k_{i+1}$ then
      $y = v_i + (d - k_i) \frac{v_{i+1} - v_i}{k_{i+1} - k_i}$
      break loops
    end if
  end for
```

(b) Initialize the algorithm to evaluate $u(k, k') + v(k')$ given $k' = d$

Pseudo-code:

```
Require: $k$, $d$ and $\{k_i, v_i\}_{i=1}^{g_k}$
Ensure: $v(k \mid k' = d)$
  $v(x) \leftarrow$ evaluate the interpolation at $d$ given $\{k_i, v_i\}_{i=1}^{g_k}$
  if $\sigma = 1$ then
    $v(k \mid k' = d) \leftarrow \ln [\bar{z}(k)^{\alpha} + (1 - \delta)k - d] + v(d)$
  else
    $v(k \mid k' = d) \leftarrow \frac{[\bar{z}(k)^{\alpha} + (1-\delta)k - d]^{1-\sigma} - 1}{1-\sigma} + v(d)$
  end if
```

(c) Initialize Golden-section search algorithm, for given k, to find

$$d = \underset{k' \in [\underline{k}, \bar{k}]}{\operatorname{argmax}} \{u(k, k') + \beta v(k')\}.$$

Start of pseudo:

```
Require: $k$ and $\{k_i, v_i\}_{i=1}^{g_k}$
Ensure: $d$
  ...
```

4. Set an initial value of $v^0 = \left\{v_i^0\right\}_{i=1}^{g_k}$. A trivial initial condition is $v^0 = 0$. Also initialize Tv

   Pseudo-code:

```
for $i = 1$ to $g_k$ do
  $v_i \leftarrow 0$
  $Tv_i \leftarrow 0$
end for
```

5. Update the value function and obtain Tv. More specifically, do the following steps for each knot in the state space $\{k_i\}_{i=1}^{g_k}$.

   (a) Solve the following problem

$$d_i^k = \underset{k' \in [\underline{k}, \bar{k}]}{\operatorname{argmax}} \{u(k_i, k') + \beta v(k')\}$$

   using golden section search.

   (b) Once d_i^k is obtained, use it to update value function. Specifically:

$$Tv_i = u\left(k_i, d_i^k\right) + \beta v\left(d_i^k\right)$$

After implementing the procedure above for $i = 1, \dots, g_k$, we have constructed a new piece-wise linear interpolation for the value function as $Tv = \{Tv_i\}_{i=1}^{g_k}$.

6. Compare $\{v_i\}_{i=1}^{g_k}$ and $\{Tv_i\}_{i=1}^{g_k}$ and compute the distance w. One way to define the error is to use maximum distance, as follows:

   $$w = \max_i |v_i - Tv_i|$$

   If $w > \varepsilon$, the error is not small enough. Update the value function using: $v = Tv$ and go back to Step 5.

7. If $w < \varepsilon$, then we find our optimal value *function*. The value function is approximated by the vector $v = \{v_i\}_{i=1}^{g_k}$. The optimal decision rule is approximated by the vector $d^k = \left\{d_i^k\right\}_{i=1}^{g_k}$.

   Given d^k we can compute the optimal decision rule for consumption $d^c = \{d_i^c\}_{i=1}^{g_k}$.