Introduction to Numerical Methods

Diego Ascarza

RIEF

Introduction

We will study now:

- How to solve a non-linear system of equations (Newton-Raphson).
- How to calculate numerical derivatives for a function.
- How to solve the sequential Social Planner's problem.
- How to implement the value function iteration of the value function for the same problem.

Newton Raphson

- Let $x = (x_1, x_2, ..., x_n)$ be a vector of n components and $F : R^n \rightarrow R^n$.
- **Goal:** Find a vector $\hat{x}$ such that $F(\hat{x}) = 0$.
- Let's denote by $\overline{x}$ the numerical approximation to the solution of $\hat{x}$.
- By doing a Taylor Expansion for F around $\overline{x}$:

$$F(x) \approx F(\overline{x}) + J(\overline{x})(x - \overline{x})$$

where $J(\overline{x})$ is the Jacobian matrix of F evaluated at $\overline{x}$:

$$M= \begin{bmatrix} F_{11}(\overline{x}) & F_{12}(\overline{x}) & ... & F_{1n}(\overline{x}) \\ F_{21}(\overline{x}) & F_{22}(\overline{x}) & ... & F_{2n}(\overline{x}) \\ .... & .... & .... & .... \\ F_{n1}(\overline{x}) & F_{n2}(\overline{x}) & ... & F_{nn}(\overline{x}) \end{bmatrix}$$

Newton Raphson

- **Taylor's Theorem:** If the approximator $\overline{x}$ is close enough to the solution $\hat{x}$:

$$F(\hat{x}) \approx F(\overline{x}) + J(\overline{x})(\hat{x} - \overline{x})$$

Then:

$$\hat{x} \approx \overline{x} - J(\overline{x})^{-1} F(\overline{x})$$

This is the mathematical foundation of Newton-Raphson's method.

Algorithm

1. Propose a initial solution x^0, using as much information you have about F and initialize $s = 0$.
2. Calculate the vector $F(x^s)$ and the matrix $J(x^s)$.
3. Calculate x^{s+1} using the rule:

$$x^{s+1} = x^s - J(x^s)^{-1} F(x^s)$$

4. Evaluate the distance $|| \ x^{s+1} - x^s \ ||$. If the distance is greater than the tolerance criteria, go back to step 2 with $s = s + 1$. Otherwise, finish with $\overline{x} = x^{s+1}$.

Algorithm

1. Propose a initial solution x^0, using as much information you have about F and initialize $s = 0$.
2. Calculate the vector $F(x^s)$ and the matrix $J(x^s)$.
3. Calculate x^{s+1} using the rule:

$$x^{s+1} = x^s - J(x^s)^{-1} F(x^s)$$

4. Evaluate the distance $\| x^{s+1} - x^s \|$. If the distance is greater than the tolerance criteria, go back to step 2 with $s = s + 1$. Otherwise, finish with $\overline{x} = x^{s+1}$.

Norms

- There are different ways to measure the distance between two vectors of dimension n:
  1. Euclidean norm:

$$\| x - y \| = \left[(x_1 - y_1)^2 + ... + (x_n - y_n)^2 \right]^{0,5}$$

- Sup. norm:

$$\| x - y \| = max \{ | x_1 - y_1 |, ..., | x_n - y_n | \}$$

Be careful

- If x_0 starts close enough to $\hat{x}$, we can show that Newton-Raphson converges to $\hat{x}$.
- But, if x_0 is not close enough to $\hat{x}$, the method can:
  1. Converge to a different solution (if the solution is not unique), or
  2. Diverge, i.e., the distance $|| x^{s+1} - x^s ||$ grows with each iteration.
- We need to try different values for x_0 before we achieve a definite answer.
- Another disadvantage of Newton Raphson is that it requires analytical expressions for all the partial derivatives of F.

Secant Method

- The secant method is similar to Newton-Raphson, but it uses numerical derivatives.
- Let's write the Jacobian matrix as follows:

$$J(x) = [J_1(x), J_2(x), ..., J_n(x)]$$

  where $J_i(x)$ is a column vector with the n partial derivatives of F respect to x_i.
- **Problem:** Find a numerical approximation for each $J_i(x)$.
- Let h be a column vector of n components (steps).

Secant Method

- If the elements of h are small enough, we can use a Taylor expansion:

$$F(x_1 + h_1, x_2, ..., x_n) \approx F(x) + J_1(x)h_1$$

$$F(x_1, x_2 + h_2, ..., x_n) \approx F(x) + J_2(x)h_2$$

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$

$$F(x_1, x_2, ..., x_n + h_n) \approx F(x) + J_n(x)h_n$$

from where we obtain, for each $i = 1, ..., n$

$$J_i(x) \approx \frac{1}{h_i}\left[F(x_1, ..., x_i + h_i, ..., x_n) - F(x)\right]$$

Secant Method

- An alternative is to approximate the Jacobian matrix from the left:

$$J_i(x) \approx \frac{1}{h_i}\left[F(x) - F(x_1, ..., x_i - h_i, ..., x_n)\right]$$

It is recommendable to take an average of both:

$$J_i(x) \approx \frac{1}{2h_i}\left[F(x_1, ..., x_i + h_i, ..., x_n) - F(x_1, ..., x_i - h_i, ..., x_n)\right]$$

Unless there is a discontinuity of F at x.

The choice of h is arbitrary. It is recommendable to try also with values progressively lower until the numerical value of the derivative is stable.

Solving the Social Planner's Problem

- Solving the deterministic problem of the Social Planner's, we obtain a system of equations in difference of first order:

$$\frac{u'(c_t}{\beta u'(c_{t+1})} = f'(k_{t+1}) + (1-\delta)$$

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

we can write this in general terms as:

$$\Psi_K(k_t, k_{t+1}, c_t, c_{t+1}) = 0$$

$$\Psi_C(k_t, k_{t+1}, c_t, c_{t+1}) = 0$$

Solving the Social Planner's Problem

- We can also combine both conditions to obtain:

$$\frac{u'[f(k_t) - k_{t+1} + (1-\delta)k_t]}{\beta u'[f(k_{t+1}) - k_{t+2} + (1-\delta)k_{t+1}]} = f'(k_{t+1}) + (1-\delta)$$

A equation of differences of second order that we can write as:

$$\Psi(k_t, k_{t+1}, k_{t+2}) = 0$$

Finally, we know that k_t converges monotonically to its steady state value:

$$k^* = (f')^{-1}\left[\frac{1}{\beta} - (1-\delta)\right]$$

Solving the Social Planner's Problem

Problem: Given the functional forms for u, f, and the value for the parameters β and δ,

1. Find sequences of values for k_t, c_t that solve the system of equations in differences $\Psi_K = 0$, $\Psi_C = 0$ or
2. Find a sequence of values for k_t that solve the equation in differences $\Psi(.) = 0$

... with initial condition $k_0 > 0$ and final $\text{lím}_{t \to \infty} k_t = k^*$

Using directly Newton-Raphson

Assuming that the model reaches the steady state in a finite number of periods (T). The approximated solution must satisfy the system of equations:

$$\Psi_K(k_0, k_1, c_0, c_1) = 0$$
$$\Psi_C(k_0, k_1, c_0, c_1) = 0$$
$$\Psi_K(k_1, k_2, c_1, c_2) = 0$$
$$\Psi_C(k_1, k_2, c_1, c_2) = 0$$

....

$$\Psi_K(k_{T-1}, k_T, c_{T-1}, c_T) = 0$$
$$\Psi_C(k_{T-1}, k_T, c_{T-1}, c_T) = 0$$

with 2T equations and $2(T+1)$ unknowns (including k_0,c_0,k_T and c_T)

Using directly Newton-Raphson

- The first missing equation is $k_0 = ...$ (whatever it is its initial value).
- The other missing equation can be k_T^* or $k_T = k_{T-1}$.
- We can the solve the system of equations using the Newton-Raphson method (or the secant method).
- There are so many equations (T is at least 100), but it usually works.
- We need to propose initial sequences for k_0^0, ...k_T^0 and c_0^0,...,c_0^T. For example, a straight line between k_0 and $k_T = k^*$.

Using directly Newton-Raphson

- The method can also be applied to the equation in differences of second order in k:

$$\Psi(k_0, k_1, k_2) = 0$$

$$\Psi(k_1, k_2, k_3) = 0$$

$$.........$$

$$\Psi(k_{T-2}, k_{T-1}, k_T) = 0$$

  This time we have $T-1$ equations and $T+1$ unknowns, the missing equations are $k_0 = ...$ and some terminal condition $k_T = k^*$ or $k_T = k_{T-1}$.

- An algorithm for this problem that does not require to solve so many equations simultaneously is the one of Gauss-Seidel.

Gauss-Seidel

The algorithm is the following:

- Propose an initial sequence $k_2^0, ..., k_{T-1}^0$ and initialize $s = 0$. For example, a straight line between k_0 and $k_T = k^*$.
- Given k_0 and k_2^s, find k_1^{s+1} by solving:

$$\Psi(k_0, k_1^{s+1}, k_2) = 0$$

  using Newton-Raphson or other method.
- Find $k_2^{s+1}, ..., k_{T-1}^{s+1}$ solving and iterating:

$$\Psi(k_1^{s+1}, k_2^{s+1}, k_3^s) = 0$$

$$..........$$

$$\Psi(k_{T-2}^{s+1}, k_{T-1}^{s+1}, k^*) = 0$$

- Calculate $\mid\mid (k_2^{s+1}, ..., k_{T-1}^{s+1}) - (k_2^s, ..., k_{T-1}^s) \mid\mid$. If the distance is greater than the tolerance criteria then go back to the second step with $s = s + 1$. Otherwise stop with $k_t = k_t^{s+1}$.

Summary

- With any of these methods, once we find a sequence for k_t we can easily calculate sequences for c_t, Y_t, w_t, r_t and any other variable of interest:

$$Y_t = f(k_t)$$

$$i_t = k_{t+1} - (1-\delta)k_t$$

$$c_t = Y_t - i_t$$

$$K_t = k_t$$

$$r_t = f'(K_t)$$

$$w_t = f(K_t) - f'(K_t)K_t$$

Value Function Iteration

- Using dynamic programming and the contraction mapping theorem, departing from any function v^0 (for example $v^0 = 0$, the sequence v^n defined by:

$$v^{n+1}(k) = max_{k'} \left\{ u[f(k) + (1-\delta)k - k'] + \beta v^n(k') \right\}$$

$$s.t.$$

$$k' \in [0, f(k) + (1-\delta)k]$$

converges to the solution of the social planner v, when $n \to \infty$. Let's see how to implement numerically this method to approximate the value function v.

Value Function Iteration

Initial setting:

- Define a grid of capital for k, this is a vector:

$$K = (K_1, K_2, ..., K_p)$$

  with $K_1 = k_{min}$ and $K_p = k_{max}$. For simplicity we can use points that are equally distanced:

$$K_2 = k_{min} + \eta \quad K_3 = k_{min} + 2\eta, \quad etc$$

  with $\eta = \frac{K_p - K_1}{p-1}$

If p is bigger (a broader grid), the approximation is more accurate but the algorithm is slower.

Value Function Iteration

- Define the matrix M as:

$$M = \begin{bmatrix} F(K_1, K_1) & F(K_1, K_2) & .... & F(K_1, K_p) \\ F(K_2, K_1) & F(K_2, K_2) & .... & F(K_2, K_p) \\ .... & .... & .... & .... \\ F(K_p, K_1) & F(K_p, K_2) & .... & F(K_p, K_p) \end{bmatrix}$$

  M saves any possible value for $F(k, k')$ for each possible combination (k, k') in our grid.

- Eliminate all the entries that are not feasible by doing:

$$M_{ij} = -1000000 \quad \text{if} \quad K_j > f(K_i) + (1 - \delta)K_i$$

Value Function Iteration

- Propose an initial column vector $V^0 \in \mathbb{R}^p$ an initialize $s = 0$ (for example, propose $V^0 = 0$.
- Given V^s and M, calculate V^{s+1} as:

$$V^{s+1} = max\left\{M + \beta(V^s xe)^T\right\}$$

  where T denotes the transpose of a matrix, $e = [1, 1, 1..., 1]$ is a row vector of size p with ones. The max is calculated row by row.
- Compute $\| V^{s+1} - V^s \|$. If the distance is greater than the tolerance criteria, go back to step 2 with $s = s + 1$. If the tolerance criteria is satisfied, finish with $V = V^{s+1}$.

Value Function Iteration

The result will be an approximation to the value function in each entry of the grid:

$$V = \begin{bmatrix} V(K_1) \\ V(K_2) \\ ... \\ V(K_p) \end{bmatrix} = \begin{bmatrix} v(K_1) \\ v(K_2) \\ ... \\ v(K_p) \end{bmatrix}$$

- The algorithm stores the optimal decision rule G as well:

$$G = argmax \left\{ M + \beta (Vxe)^T \right\}$$

  G is a column vector of n components, where $G_i \in \{1, ..., p\}$ indicates the number of the column that maximizes the row i.

Value Function Iteration

Therefore, departing from any $k_0 = K_i$, we can recover the optimal sequence for capital:

$$k_1 = K_j \quad with \quad j = G_j$$

$$k_2 = K_l \quad with \quad l = G_j$$

$$..........$$

Solving the Recursive Equilibrium Directly

The value function iteration is not ideal to solve directly the recursive competitive equilibrium since it requires:

- Two state variables (individual capital and aggregate capital) (not that important).
- The law of motion Γ is an unknown object when the consumer decides to solve her Bellman equation.

We will have to follow then an algorithm of double iteration.

We suppose that the law of motion follows a polynomial of degree n:

$$K' = \Gamma(K) = \alpha_0 + \alpha_1 K + \alpha_2 K^2 + ... + \alpha_n K^n$$

Algorithm

1. Propose a initial vector of parameters $(\alpha_0, \alpha_1, ..., \alpha_n)$.
2. Given Γ, solve the Bellman equation of the consumer iterating the value function and obtain the optimal sequence $k_0, k_1, ..., k_T$.
3. Using the time series $k_0, k_1, ..., k_T$, run the regression:

$$k_{t+1} = a_0 + a_1 k_t + a_2 k_t^2 + ... + a_n k_t^n$$

   and estimate a vector of parameters $(\hat{a}_0, ..., \hat{a}_n)$
4. Compare $(\hat{a}_0, ..., \hat{a}_n)$ and $(\alpha_0, \alpha_1, ..., \alpha_n)$. If the distance is greater than the tolerance criteria, go back to step 2 with the new law of motion. In other case, the algorithm converges.

Algorithm

- This method will be more accurate with a higher degree of the polynomial.
- Even with n large, the convergence is not guaranteed.