Macroeconomics III
Lecture 2: Root Finding and Function Approximation

Tiago Cavalcanti

FGV/EESP
São Paulo

Road Map: Root Finding

1. Bisection Method;
2. Local approximation:
   - Newton-Raphson Method (one dimension and multidimension)
   - Secant

Reading: Judd (1998, ch 5)

Root-finding

- Two of the most common problems in economics:

$$f(x) = 0, \;\; f : R^n \to R^n.$$

$$g(x) = x, \;\; g : R^n \to R^n.$$

- The first is called a root-finding problem, and the second is a fixed-point problem.

- The two are equivalent in the following way:
  - For any f, define $g = f + x$. Then $f = 0 \Leftrightarrow g = x$.
  - For any g, define $f = g - x$. Then $g = x \Leftrightarrow f = 0$.

- Often it is infeasible to solve this analytically.

Root-finding in One Dimension

- We will begin with the simplest case, one-dimensional root-finding:

$$f(x) = 0, \;\; f : R \to R.$$

Bisection Method

- **Bisection:** A simple and robust method for finding the root of a **univariate** continuous function $f(x)$ on a closed interval $[a, b]$.
- Always converges to the solution - if one exists, and if the initial interval includes the solution.
- Does not rely on the derivatives of the function $\Rightarrow$ can be used to find roots of non-smooth functions
- Basic idea: If f is continuous and $f(a)$ and $f(b)$ are of opposite sign $\Rightarrow$

  $\exists x^* \in [a, b],$ for which $f(x^*) = 0$ (Intermediate Value Theorem).

Bisection Method

- Algorithm (for $f(a) < 0$ and $f(b) > 0$):

  (i) Define lower $\underline{x}$ and upper bound $\bar{x}$ of interval: Use $\underline{x} = a$ and $\bar{x} = b$.

  (ii) Compute midpoint of interval $c = \frac{\bar{x}+\underline{x}}{2}$.

  (iii) if $f(c) > 0$ set $\bar{x} = c$, otherwise if $f(c) < 0$ set $\underline{x} = c$.

  (iv) If $|f(c)| \leq \epsilon$, then stop and call c a root, otherwise go to step (ii).

Bisection Method to find the root of $f(x) = x^2 - 4$

```
% Bisection Method. Root of f(x)=x^2-4 for x in [a,b];
a=1; % Lower bound
b=10; % Upper bound
fa=a^1-4; % Value at the lower bound
fb=b^2-4; % Value at the upper bound
tol=0.001;
if fa*fb>0 % Condition of the IVT
    disp('Wrong choice for a and b')
else
    c=(a+b)/2; % Midpoint
    err=abs(c^2-4);
    while err>tol
        if c^2-4>0
            b=c;
        elseif c^2-4<0
            a=c;
        end
        c=(a+b)/2;
        err=abs(c^2-4);
    end
end
```

A More Elegant and Efficient Way

- Write a general bisection function in Matlab (script or m-file);
- Then call this function to find the root.

The script will be:

```
function c = bisection(f,a,b)
%
% Simple code to find a root of univariate function
% Give initial guesses for intervals
% Solves it by method of bisection.

if f(a)*f(b)>0
    disp('Wrong choice for a and b')
else
    c = (a + b)/2;
    err = abs(f(c));
    while err > 1e-7
        if f(a)*f(c)<0
            b = c;
        else
            a = c;
        end
        c = (a + b)/2;
        err = abs(f(c));
    end
end
```

How do you call the function (you need f, a, and b).

- In Matlab command window write:
  $>> f = @(x)x^2 - 4;$
  This is the function, you will calculate the root. You need to define the intervals.
  $>> a = 1;$
  $>> b = 10;$
  $>> yourroot = bisection(f, a, b)$
- *yourroot* is the root you are after.

The script will be:

```
% Matlab example to use the bisection.
f=@(x)x^2-4;   % Function f(x)=x^2-4
%
a=1;        % Lower bound of the the interval x\in[a,b]
%
b=10;       % Upper bound of the the interval x\in[a,b]
%
yourroot=bisection(f,a,b);
%
display('Root of the function')
yourroot
%
```

Application: Neoclassical Growth Model

- Suppose a consumer maximizes

$$\max_{\{c_t, k_{t+1}\}_{t=0}^{\infty}} U_0 = \sum_{t=0}^{\infty} \beta^t u(c_t), \; 0 < \beta < 1 \tag{1}$$

$$\text{s.t. } k_{t+1} + c_t \leq f(k_t) + (1-\delta)k_t$$

$$k_0 > 0 \text{ given,}$$
$$c_t, k_{t+1} \geq 0.$$

- **Note:** $u(.)$ is strictly concave and technology is $f(.)$ strictly concave and satisfying INADA conditions.

Concave Programming

Interior solution (Inada condition):

$$u'(c_t) = \beta(f'(k_{t+1}) + (1-\delta))u'(c_{t+1}), \ \forall t = 0, 1, \ldots$$

$$c_t = f(k_t) + (1-\delta)k_t - k_{t+1}, \ \forall t = 0, 1, \ldots$$

$$\lim_{T\to\infty} \beta^T u'(c_T)k_{T+1} = 0, \ \text{ given } \ k_0.$$

Let $u(c) = \frac{c^{1-\theta}-1}{1-\theta}$ with $\theta > 0$ and $f(k) = Ak^\alpha$ with $\alpha \in (0, 1)$. Then:

$$\frac{c_{t+1}}{c_t} = \beta(\alpha A k_{t+1}^{\alpha-1} + 1 - \delta)^{\frac{1}{\theta}},$$

$$c_t = Ak_t^\alpha + (1-\delta)k_t - k_{t+1},$$

$$\lim_{T\to\infty} \beta^T c_T^{-\theta} k_{T+1} = 0, \ \text{ given } \ k_0.$$

Steady-state

Since no growth in exogenous variables, in the steady-state, we have $c_{t+1} = c_t = c$, and $k_{t+1} = k_t = k$. Then

$$k = \left(\frac{\alpha A}{\left(\frac{1}{\beta}\right)^{\frac{1}{\theta}} + (1-\delta)} \right)^{\frac{1}{1-\alpha}} = k(A, \alpha, \beta, \delta, \theta),$$

$$c = Ak^{\alpha} - \delta k = c(A, \alpha, \beta, \delta, \theta).$$

Golden Rule:

$$k_{GR} = \left(\frac{\alpha A}{\delta} \right)^{\frac{1}{1-\alpha}}.$$

Then: $k < k_{GR}$, as long as $\beta \in (0, 1)$ and $\theta > 0$, which are necessary for Transversality Condition to be satisfied.

Transition dynamics

System of equations for all $t = 0, 1, \ldots$

$$k_{t+2} = Ak_{t+1}^{\alpha} + (1-\delta)k_t - \beta(\alpha Ak_{t+1}^{\alpha-1} + 1 - \delta)^{\frac{1}{\theta}}(Ak_t^{\alpha} + (1-\delta)k_t - k_{t+1}),$$

$$k_{t+2} = G(k_{t+1}, k_t). \text{ Then: } k_2 = G(k_1, k_0).$$

Therefore, given k_0 if we knew k_1, then we could find k_2, and $k_3 = G(k_2, k_1)$, and so on!

Use bisection!!!

(Shooting) Algorithm

Let $k_0 < k$. Define $[a = k_0, b = k]$ and define a large time period T.

1. Guess $k_1^1 = \frac{a+b}{2}$, then find k_2^1, and $k_3^1 = G(k_2^1, k_1^1)$, and so on!
    - If $k_T^1 > k$, then (over-accumulation), k_1^1 too high, define $b = k_1^1$;
    - If $k_T^1 < k$, then (under-accumulation), k_1^1 too low, define $a = k_1^1$;
    - If $|k_T^1 - k| < \epsilon$, then stop!
2. Guess $k_1^2 = \frac{a+b}{2}$, then find k_2^2, and $k_3^2 = G(k_2^2, k_1^2)$, and so on!
    - If $k_T^2 > k$, then (over-accumulation), k_1^2 too high, define $b = k_1^2$;
    - If $k_T^2 < k$, then (under-accumulation), k_1^2 too low, define $a = k_1^2$;
    - If $|k_T^2 - k| < \epsilon$, then stop!
3. And so on!

Bisection Method

- **Advantages**:
  - Finds a zero of any C^0 function.
  - Extremely simple.
  - Frequently used.
- **Disadvantages**
  - Convergence is slow relative to other methods. It does not exploit information about function curvature. It is a linear convergence.
  - Have to find initial bracket (true for all these methods).

Function approximation

Classifications

- Local approximation methods
  - Taylor approximations
- Global approximation methods
  - Ordinary regression + interpolation
  - Orthogonal polynomials (Chebyshev polynomials)
  - Chebyshev regression in $\mathbb{R}$ and $\mathbb{R}^2$
  - Splines
  - Shape preserving approximation

Local versus global approximations

- Solving macro models computationally always boils down to essentially one problem:
- Getting **good approximations for functions** (e.g. value functions, policy functions, etc.) that are not known to the modeller and cannot be derived in closed form (analytically).
- Function approximation is broadly split into two types of methods:
  - **Local approximations**: focus on one point of interest (typically the deterministic steady state of an economy) and approximate locally around that point
  - **Global approximations**: approximate an unknown function f with 'nice' functions g that are close to f, where 'close' is in some well specified sense.

Local versus global approximations

- Starting point for each method are the following two powerful theorems:
- **Taylor's Theorem**: any sufficiently smooth function can be locally approximated by polynomials
  - *Advantage*: Gives us the exact approximation
  - *Disadvantage*: Holds locally; e.g. as we move away from the steady state, approximation may deteriorate a lot
- **Weierstrass' Theorem**: any continuous function defined on an interval [a,b] can be uniformly approximated as closely as desired by a polynomial function
  - *Disadvantage*: Does not tell us which polynomial to pick for approximating the function (tricky)
  - *Advantage*: Generates a global approximation

Local approximations

- Let $f : \mathbb{R} \to \mathbb{R}$ and $x^* \in \mathbb{R}$.
- The information may include knowing the actual value $f(x^*)$ and also derivatives of the function at that point.
- **Taylor series approximation for univariate functions**: For $f : \mathbb{R} \to \mathbb{R}$ where $f \in C^{n+1}$ and $x^* \in \mathbb{R}$, the $n-$th order Taylor approximation is given by

$$f(x) \approx f(x^*) + (x - x^*) f^{(1)}(x^*) + \frac{1}{2}(x - x^*)^2 f^{(2)}(x^*) + \ldots + \frac{1}{n!}(x - x^*)^n f^{(n)}(x^*) + e_{n+1}$$

where the error term $e_{n+1} \to 0$ as $n \to \infty$.

Local approximations

- **Taylor series approximation for multivariate functions**: For $f : \mathbb{R}^m \to \mathbb{R}$ where $f \in C^{n+1}$ and $x^* \in \mathbb{R}^m$, the $n-$th order Taylor approximation is given by

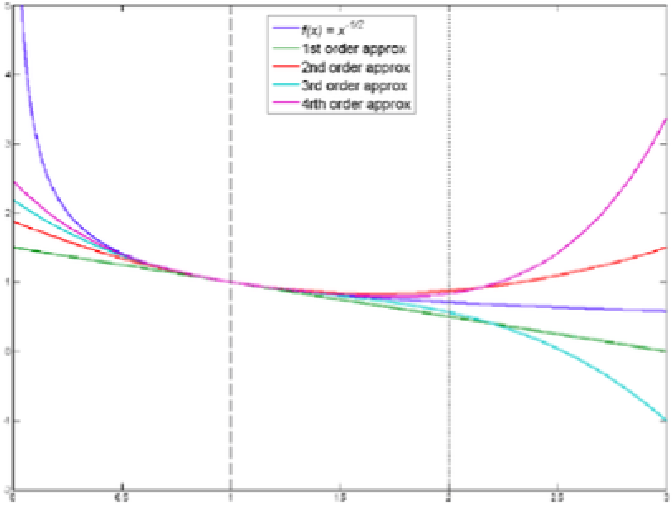
$$
\begin{aligned}
f(x) \quad \approx \quad & f(x^*) + \sum_{i=1}^{m} \left(\frac{\partial f(x^*)}{\partial x_i} (x_i - x_i^*) \right) + \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \left(\frac{\partial^2 f(x^*)}{\partial x_i \partial x_j} (x_i - x_i^*) \left(x_j - x_j^* \right) \right) + \\
& \ldots + \frac{1}{n!} \sum_{i_1=1}^{m} \ldots \sum_{i_n=1}^{m} \left(\frac{\partial^n f(x^*)}{\partial x_{i_1} \ldots \partial x_{i_n}} \prod_{k=1}^{n} \left(x_{i_k} - x_{i_k}^* \right) \right) + e_{n+1}
\end{aligned}
$$

where the error term $e_{n+1} \to 0$ as $n \to \infty$.

Local approximations

- A local approximation is designed to give good approximations near a point x^* but its accuracy may deteriorate rapidly once we are away from that point.
- A useful theorem: Suppose that we know that the function f (or some derivative of it) has a singularity at a point y. Then, the Taylor approximation around x^* is reliable only within an interval $[x^* - d, x^* + d]$, where d is the distance (in absolute terms) of the two points, x^* and y (see [Judd, 1998, ch. 6]).
- **Example**: Take $f(x) = x^{-1/2}$ that has a singularity at $x = 0$, and approximate around $x^* = 1$. We expect that the approximations will deteriorate close to the singularity, but also according to the theorem for any $x > 2$.

Taylor approximations with singularity ($f(x) = x^{-\frac{1}{2}}$)



Two-Period Saving Problem

- As an example take the following problem:

$$\max_{a_1,c_1,c_2} \{u(c_1) + \beta u(c_2)\} \text{ subjet to}$$

$$c_1 + a_1 = m,$$

$$c_2 = (1+r)\,a_1.$$

- Or: $\max_{a_1} \{u(m - a_1) + \beta u((1+r)\,a_1)\}$.
- The first-order conditions are

$$-u'(m - a_1) + \beta(1+r)\,u'((1+r)\,a_1) = 0,$$

$$c_1 + a_1 = m,$$

$$c_2 = (1+r)\,a_1.$$

- We wish to find $a_1 = g(m)$ or $c_1 = h(m) = m - g(m)$
- Our first step to find the policy function is to solve the **one dimensional problem** given by the Euler equation.

Newton-Raphson (one dimension)

- Suppose you need to solve $f(x) = 0$ and $f : \mathbb{R} \rightarrow \mathbb{R}$

- Start at x^0 ($f(x^0)$ must be defined)

- Take the first-order Taylor approximation of f around x^0

$$f(x) \approx v(x) = f\left(x^0\right) + f'\left(x^0\right)\left(x - x^0\right)$$

- Find x^1, which solves the zero of v implying:

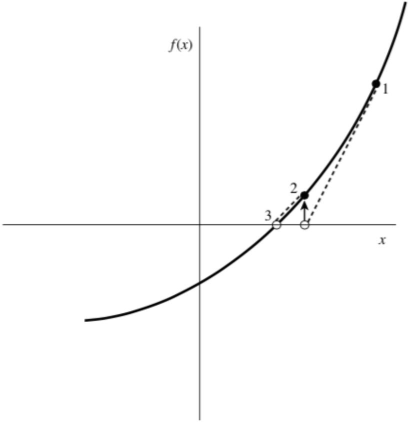
$$v\left(x^1\right) = 0 \quad \Longrightarrow \quad x^1 = x^0 - \frac{f\left(x^0\right)}{f'\left(x^0\right)}$$

- Check if $f(x^1)$ is defined (if not, choose a point between x^0 and x^1). If yes, repeat the procedure at x^1, such that:

$$x_{s+1} = x_s - \frac{f(x_s)}{f'(x_s)}.$$

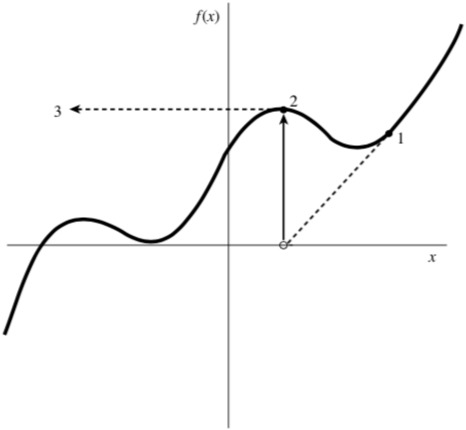
- The solution x^* is found if $f(x_{s+1}) \approx 0$.

Newton-Raphson Method Using Derivative



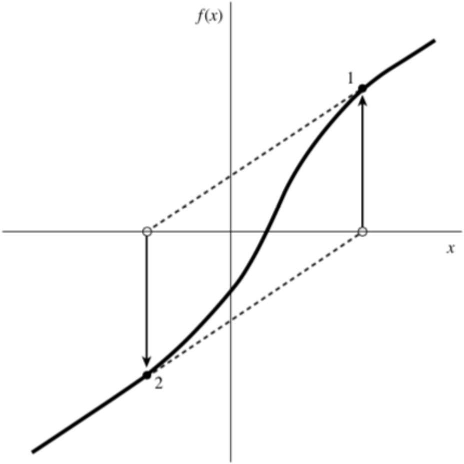
Newton's method extrapolates the local derivative to find the next estimate of the root.

Newton-Raphson Method Using Derivative



Newtons method encounters a local extremum and shoots off...
Bracketing bounds would save the day.

Newton-Raphson Method Using Derivative



Newton's method enters a non-convergent cycle. A better initial guess can solve the problem.

Newton-Raphson Method (one dimension)

- Suppose we apply it to our two period saving problem.

- We need:

$$
\begin{aligned}
f(x) &= -u'(m - a_1) + \beta(1+r)\,u'((1+r)\,a_1)\,, \\
f'(x) &= u''(m - a_1) + \beta(1+r)^2\,u''((1+r)\,a_1)\,.
\end{aligned}
$$

- For a given m choose an initial condition $a_{1,0}$ and compute:

$$
a_{1,s+1} = a_{1,s} - \frac{f(a_{1,s})}{f'(a_{1,s})} \quad \text{for } s = 0, 1, \ldots, S
$$

until $|f(a_{1,s+1})| < \delta$

- *Example 1: Find the zeros of*

$$y = (x-4)(x+4).$$

```
function x = newton(func,x0,param,crit,maxit);

% Newton.m Program to solve a system of equations
% x=newton(func, x0, param, crit, maxit)

for i=1:maxit
    [f,J]=feval(func,x0,param);
    x=x0-inv(J)*f;
    if norm(x-x0)<crit;
            break
    end
    x0=x;
end

if i>=maxit
   sprintf('WARNING: Maximum number of %g iterations
      reached',maxit)
end
```

Before, we have to provide the inputs of the code:

```
%
% This program solves by the Newton-Raphson method
% the following equation: y=(x-4)*(x+4)
clear all
% Seed
x0=1;
% Maximum number of iterations
maxit=1000;
% Tolerance value
crit=1e-3;
% Parameters of the system
param=4;
% Call the newton.m program and specify the file with
    the
% equations and Jacobian.

sol=newton('nexp1', x0, param, crit, maxit);

sprintf('x=%g', sol(1))
```

Program with the function and Jacobian:

```
function [f,J]=nexp1(z,p)
%here.m
x=z(1);
a=p(1);
f=(x-a)*(x+a);
J=2*x;
```

If $x0 = 1$, the root is 4; If $x0 = -1$, the root is -4.

Newton-Raphton Method (multidimension)

- Suppose now:

$$f(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{pmatrix} \quad \text{and } x \in \mathbb{R}^n.$$

- The Taylor expansion becomes

$$f(x) \approx g(x) = f\left(x^0\right) + J_{x^0}\left(x - x^0\right)$$

where

$$J_x = \begin{pmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \frac{\partial f_2(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \cdots & \frac{\partial f_2(x)}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n(x)}{\partial x_1} & \frac{\partial f_n(x)}{\partial x_2} & \cdots & \frac{\partial f_n(x)}{\partial x_n} \end{pmatrix}$$

- The solution then becomes:

$$x^1 = x^0 - J_{x^0}^{-1} f\left(x^0\right)$$

- It generally works well and converges fast but it is important to have good initial guesses.

- *Example 2: Find the zeros of*

$$\begin{aligned} x^2 - 4 + y &= 0 \\ y + 5 &= 0 \end{aligned}$$

Before, we have to provide the inputs of the code:

```
% This program solves by the Newton-Raphson
% x^2-4+y=0
% y+5=0
clear all
% Seed
x0=[1; 1];
% Maximum number of iterations
maxit=1000;
% Tolerance value
crit=1e-3;
% Parameters of the system
param=[2; 4; 5];
% Call the newton.m program and specify the file with the
% equations and Jacobian.
sol=newton('nexp2', x0, param, crit, maxit);
sprintf('x=%g', sol(1))
sprintf('y=%g', sol(2))
```

Program with the function and Jacobian:

```
function [f,J]=nexp1(z,p)
%here.m
x=z(1);
y=z(2);
a=p(1);
b=p(2);
c=p(3);
f=[x^a-b+y; y+c]
J=[a*x 1; 0 1];
```

In this case, there is a unique solution $(x, y) = (3, 5)$.

Some Remarks on the Newton-Raphson Method

1. Advantages:
   - Faster convergence than the bisection (quadratic convergence)
   - It can be used to multivariable systems

2. Disadvantages
   - Choice of initial conditions
   - Function has to be differentiable
   - $f(x_{s+1})$ may not be defined and the procedure does not ensure convergence
   - Computation of the derivative of 'f' can be costly or impossible

Using the Secant instead of the Tangent

- What if we cannot compute the derivative?
- Recall that:

$$
\begin{aligned}
J_1(x) &= \lim_{h_1 \to 0} \frac{F(x) - F(x_1 + h_1, x_2, ..., x_n)}{h_1}, \\
J_2(x) &= \lim_{h_2 \to 0} \frac{F(x) - F(x_1, x_2 + h_2, ..., x_n)}{h_2}, \\
\vdots &= \vdots, \\
J_n(x) &= \lim_{h_n \to 0} \frac{F(x) - F(x_1, x_2, ..., x_n + h_n)}{h_n}.
\end{aligned}
$$

- Notice that $J_i(x)$ is a column vector with the n partial derivatives with respect to x_i. Therefore, $J(x) = [J_1(x), J_2(x), ..., J_n(x)]$. We can define a small value h, such that

$$
\begin{aligned}
J_1(x) &\approx \lim_{h \to 0} \frac{F(x) - F(x_1 + h, x_2, ..., x_n)}{h}, \\
J_2(x) &\approx \lim_{h \to 0} \frac{F(x) - F(x_1, x_2 + h, ..., x_n)}{h}, \\
\vdots &\approx \vdots, \\
J_n(x) &\approx \lim_{h \to 0} \frac{F(x) - F(x_1, x_2, ..., x_n + h)}{h}.
\end{aligned}
$$

```
function x=secant(func, x0, param, crit, maxit)
%   secant.m solves a system of equations
%   f(z1, z2,...,zn)=0 with the secant method
%   where x=[z1, z2,...,zn] is the solution vector.
%   function 'func', which is a string.
%   param corresponds to additonal parameters of the function 'func'.
%   x0, crit, and maxit
del=diag(max(abs(x0)*1e-4, 1e-8));
n=length(x0);
    for i=1:maxit
        f=feval(func,x0,param);
        for j=1:n
            J(:,j)=(f-feval(func,x0-del(:,j),param))/del(j,j);
        end
        x=x0-inv(J)*f;
        if norm(x-x0)<crit; break;  end
        x0=x;
    end
    if i>=maxit
        sprintf('maximum number of iterations was reached')
```

Before, we have to provide the inputs of the code:

```
%
% This program solves by the Secant method
% the following equation: y=(x-4)*(x+4)
clear all
% Seed
x0=1;
% Maximum number of iterations
maxit=1000;
% Tolerance value
crit=1e-3;
% Parameters of the system
param=4;
% Call the newton.m program and specify the file with the
% equations and Jacobian.

sol=secant('sexp1', x0, param, crit, maxit);

sprintf('x=%g', sol(1))
```

Program with the function and Jacobian:

```
function [f]=sexp1(z,p)
%here.m
x=z(1);
a=p(1);
f=(x-a)*(x+a);
```

If $x0 = 1$, the root is 4; If $x0 = -1$, the root is -4.

Optimal Growth Problem in Discrete Time:

- Suppose consumer maximizes

$$\max_{\{c_t, k_{t+1}\}_{t=0}^{\infty}} U_0 = \sum_{t=0}^{\infty} \beta^t u(c_t), \; 0 < \beta < 1 \tag{2}$$

$$\text{s.t. } k_{t+1} + c_t \leq f(k_t) + (1-\delta)k_t$$

$$k_0 > 0 \text{ given,}$$

$$c_t \geq 0.$$

- **Note:** $u(.)$ is strictly concave and technology is $f(.)$ strictly concave and satisfying INADA conditions.

Solution:

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

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

$$k_0 \text{ given, and } \lim_{T\to\infty} \beta^T u'(c_T)k_{T+1} = 0.$$

Let $u(c) = \frac{c^{1-\eta}}{1-\eta}$ and $f(k) = Ak^\alpha$. Then solution can be written as:

$$\beta(Ak_{t+1}^\alpha + (1-\delta)k_{t+1} - k_{t+2})^{-\eta}(\alpha Ak_{t+1}^{\alpha-1} + (1-\delta)) - (Ak_t^\alpha + (1-\delta)k_t - k_{t+1})^{-\eta} = 0.$$

Plus terminal conditions. This is a second-order difference equation $\varphi(k_{t+2}, k_{t+1}, k_t) = 0$.
We know that the model converges to a steady state:

$$k_{ss} = [\frac{\alpha A}{\frac{1}{\beta} - (1-\delta)}]^{\frac{1}{1-\alpha}}.$$

- Problem: k_t goes to k_{ss} only when T goes to ∞.
- However, the speed of convergence to the steady-state value is usually very high in this growth model
- In practice for a given k_0, we know that k_t is very close to k_{SS}, when, for instance, $t = 30$.
- Therefore, it is possible to find an approximate trajectory very close to the true one. The idea is to construct a vector $k = [k_0, k_1, k_2, ..., k_{T+1}]$ corresponding to the approximated trajectory that solves:

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

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

$$\vdots$$

$$\varphi(k_{T+1}, k_T, k_{T-1}) = 0.$$

Notice that we know k_0 and $k_{T+1} = k_{SS}$ and therefore we have T equations and T unknown variables.

% growth.m
% This program solves the basic growth model %
% Time period utility: $u(c) = c^{1-\eta} - 1/(1-\eta)$
%
% Technology: $c(t) + k(t+1) - (1-\delta)k(t) = Ak(t)^{\alpha}$
%
% given $k(0)$
%
% PARAMETER (MODEL) VALUES
$A = 10;$
$\alpha = 0.36;$
$\delta = 0.06;$
$\eta = 0.99;$
$\beta = 0.96;$
%

% PARAMETERS OF THE PROGRAM (secant.m)

$maxit = 1000;$

$crit = 1e - 3;$

%

% INITIAL CAPITAL STOCK AND STEADY-STATE

%

$kss = ((A * \beta * \alpha)/(1 - (1 - \delta) * \beta))^{(1/(1-\alpha))};$

$k0 = 0.8 * kss;$

$T = 30;$

%

% SEED %

$x0 = [k0\ k0 * ones(size(1 : T - 1))]';$

%

% CALL THE PROGRAM secant.m (it has to be in the same directory)

%

$param = [A\ \alpha\ \delta\ \eta\ \beta\ T\ k0\ kss];$

$sol = secant('foc', x0, param, crit, maxit)$

%

% SOLUTION

%

$k = [k0; sol; kss];$

$y = A * k.^{\alpha};$

$i = k(2 : T + 1) - (1 - \delta) * k(1 : T);$

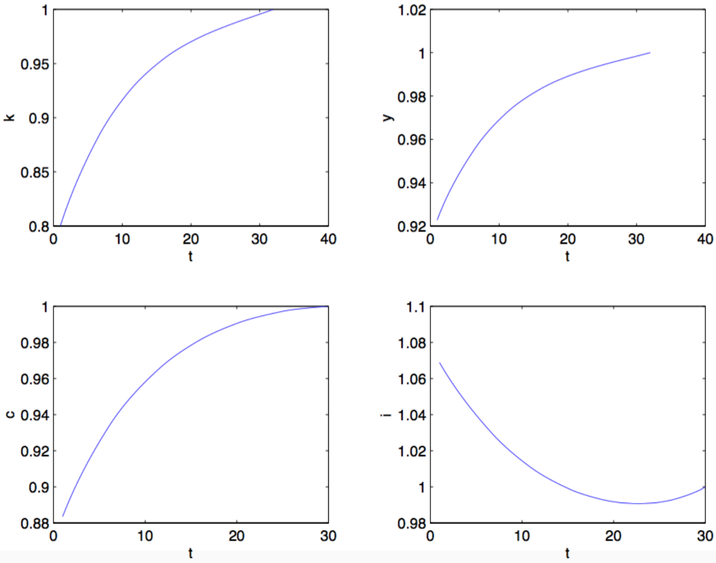
$c = y(1 : T) - i;$

The program *foc.m* contains the first-order conditions of the growth model.

function f=foc(z,p)
% Function foc.m contains the first order conditions
% of the basic neoclassical growth model
%
% VECTOR p
%
$A = p(1);$
$\alpha = p(2);$
$\delta = p(3);$
$\eta = p(4);$
$\beta = p(5);$
$T = p(6);$
$k0 = p(7);$
$kss = p(8);$

% ENDOGENOUS VARIABLE z
%
for $t = 1 : T$
$k(t) = z(t);$
end
$k(T+1) = kss;$
$f(1) = \beta * (A * k(1)^{\alpha} + (1-\delta) * k(1) - k(2))^{(-\eta)} * (\alpha * A * k(1)^{(\alpha-1)} + (1-\delta)) - (A * k0^{\alpha} + (1-\delta) * k0 - k(1))^{(-\eta)};$
for $t = 2 : T$
$f(t) = \beta * (A * k(t)^{\alpha} + (1-\delta) * k(t) - k(t+1))^{(-\eta)} * (\alpha * A * k(t)^{(\alpha-1)} + (1-\delta)) - (A * k(t-1)^{\alpha+}(1-\delta) * k(t-1) - k(t))^{(-\eta)};$
end
$f = f'$**;**

Transition Dynamics of a Neoclassical Growth Model



Issues on Numerical Differentiation

- The secant method above is based on numerical differentiation.
- Taking h very small, for instance much less than ϵ where ϵ is machine precision, means that $x = x + h$! So ensure that $h > \epsilon$.
- It is worse than this because $f(x + h) - f(x)$ for h close to 0 means many digits of accuracy are being lost.
- However, taking h large means you are probably not near the limit.