MATLAB and Macroeconomics

Murat Üngör

Department of Economics
University of Southern California

September 18, 2009

BASIC OPERATIONS OF MATLAB

- MATLAB is a high-level software package with many built-in functions that make the learning of numerical methods much easier and more interesting.
- Once you start the MATLAB program, a Command window will open with the MATLAB prompt $>>$.
- On the command line, you can type MATLAB commands, functions together with their input/output arguments, and the names of script files containing a block of statements to be executed at a time or functions defined by users.
- The MATLAB program files must have the extension name **.m** to be executed in the MATLAB environment.
- If you want to create a new M-file or edit an existing file, you click File/New/M-file or File/Open in the top left corner of the main menu, find/select/load the file by double-clicking it, and then begin editing it in the Editor window.

BASIC OPERATIONS OF MATLAB

- If the path of the file you want to run is not listed in the MATLAB search path, the file name will not be recognized by MATLAB.
- In such cases, you need to add the path to the MATLAB-path list by clicking the menu File/Set Path in the Command window, clicking the Add Folder button, browsing/clicking the folder name, and finally clicking the SAVE button and the Close button.
- The **lookfor** command is available to help you find the MATLAB commands/functions which are related with a job you want to be done. The **help** command helps you know the usage of a particular command/function. You may type directly in the Command window

  ```
  >> lookfor norm                    >> help for
  ```

  to find the MATLAB commands in connection with norm or to obtain information about the for loop.

- **exit** or **quit** gets you out of MATLAB and back to the operating system.
- **help** displays several screens full of MATLAB commands. For help on a specific command, such as **norm**, say **help norm**
- The command **demo** runs a set of demonstrations.

To get MATLAB to work out $1 + 1$, type the following at the prompt:

```
1+1
```

MATLAB responds with

```
ans =
  2
```

The answer to the typed command is given the name **ans**. In fact **ans** is now a variable that you can use again. For example you can type

```
ans*ans
```

to check that $2 \times 2 = 4$:

```
ans*ans
ans =
  4
```

The Colon Operator and Linspace

To generate a vector of equally-spaced elements matlab provides the colon operator. Try the following commands:

```
1:5
0:2:10
0:.1:2*pi
```

The syntax x:y means roughly generate the ordered set of numbers from x to y with increment 1 between them. The syntax x:d:y means roughly generate the ordered set of numbers from x to y with increment d between them.

To generate a vector of evenly spaced points between two end points, you can use the function **linspace(start,stop,npoints)**:

```
x = linspace(0,1,10)
x =
0 0.1111 0.2222 0.3333 0.4444 0.5556 0.6667 0.7778 0.8889
1.0000
```

generates 10 evenly spaced points from 0 to 1.

Plotting Vectors

```
x = 0:.1:2*pi;
y = sin(x);
plot(x,y)
```

The first line uses the colon operator to generate a vector x of numbers running between 0 and 2 with increment 0.1.
The second line calculates the sine of this array of numbers, and calls the result y.
The third line produces a plot of y against x. Go ahead and produce the plot.
You should get a separate window displaying this plot.

x = 0:.1:2*pi;   y = sin(x);   plot(x,y)

In this case we used plot to plot one vector against another. The elements of the vectors were plotted in order and joined by straight line segments. There are many options for changing the appearance of a plot. For example:

plot(x,y,'r-.')

will join the points using a red dash-dotted line. Other colors you can use are: 'c', 'm', 'y', 'r', 'g', 'b', 'w', 'k', which correspond to cyan, magenta, yellow, red, green, blue, white, and black.
Possible line styles are: solid '-', dashed '–', dotted ':', and dash-dotted '-.'.
To plot the points themselves with symbols you can use: dots '.', circles 'o', plus signs '+', crosses 'x', or stars '*', and many others (type help plot for a list). For example:

plot(x,y,'bx')

plots the points using blue crosses without joining them with lines.

Clearing the Figure Window

You can clear the plot window by typing **clf**, which stands for clear figure.

To get rid of a figure window entirely, type **close**.

To get rid of all the figure windows, type **close all**.

New figure windows can be created by typing figure.

Subplots

To plot more than one set of axes in the same window, use the subplot command. You can type

```
subplot(m,n,p)
```

to break up the plotting window into m plots in the vertical direction and n plots in the horizontal direction, choosing the pth plot for drawing into.

```
t = 0:.1:2*pi;
subplot(2,2,1)
plot(cos(t),sin(t))
subplot(2,2,2)
plot(cos(t),sin(2*t))
subplot(2,2,3)
plot(cos(t),sin(3*t))
subplot(2,2,4)
plot(cos(t),sin(4*t))
```

Matrices

One of the many features of MATLAB is that it enables us to deal with the vectors/matrices in the same way as scalars. For instance, to input the matrices/ vectors:

A=[2 4 3;-5 0 12]	2×3 matrix named A
v=[9;8;-3]	column vector v
B=[2*3 π log(6);1/3 0 sqrt(-1)]	3×2 complex matrix
C=A'+B	matrix addition
M=A$\times$v	matrix-vector product
D=inv(A)	inverse of A

Matrices

- m=4; n=7; scalars,i.e., 1×1 matrix
- A=zeros(m,n) m×n matrix of zeros
- A=ones(m,n) m×n matrix of ones
- A=rand(m,n) m×n matrix of random numbers in $[0,1]$
- A=eye(m,n) m×n identity matrix

Accessing parts of a matrix

- $A(i,j)$ matrix element
- $A(i,:)$ *i*-th row
- $A(:,j)$ *j*-th column
- $A(p:q, r:s)$ submatrix comprising intersection of rows p-q and columns r-s
- $A(:)$ the columns of A strung out into one long vector

The growth rate between t and $t+1$ of a variable x is defined by

$$\frac{x_{t+1} - x_t}{x_t} = \frac{x_{t+1}}{x_t} - 1$$

Growth rates are often approximated by log-differences, defined by

$$\log\left(\frac{x_{t+1}}{x_t}\right) = \log(x_{t+1}) - \log(x_t)$$

The basis for approximating growth rates by log-differences is the result that

$$\log\left(\frac{x_{t+1}}{x_t}\right) = \log\left(1 + \frac{x_{t+1} - x_t}{x_t}\right) \simeq \frac{x_{t+1} - x_t}{x_t}$$

which is equivalent to

$$\log(1+z) \simeq z, \qquad \textit{for} \;\; \textit{small} \;\; z$$

This assertion is based on the linearization (first order Taylor series expansion)

$$\log(1+z) \simeq \log(1+z_0) + \frac{1}{1+z_0}(z - z_0)$$

around the point $z_0 = 0$.

```
% GROWTH RATE APPROXIMATIONS

% The next command produces a 1000-by-1 vector of evenly
% spaced "zs" between 0 and 1.

zs = linspace(0,1,1000)';

g_approximate      = log(1+zs);
g_actuals          = zs;

figure(1)
plot(100*zs,100*g_approximate,'k--',100*zs,100*g_actuals)
title('Accuracy␣of␣log-differences␣as␣aproximate␣growth␣rates
xlabel('zs␣(%)')
ylabel('growth␣rates␣(%)')
legend('log-approximate␣growth␣rate','actual␣growth␣rate',0)
```

A discrete-time version of the Solow model leads to a non-linear difference equation in a single state variable, k_t the capital stock per efficiency unit of labor

$$(1+g)(1+n)k_{t+1} = sk_t^{\alpha} + (1-\delta)k_t$$

The meaning of the parameters in this expression are listed in the following table. The far column lists some suggestive values of these parameters.

Table: Parameters

Symbol	Meaning	Value
g	annual growth rate of productivity	0.03
n	annual growth rate of working age population	0.01
s	national savings rate	0.2
α	capitals share in national output	0.33
δ	annual depreciation rate of physical capital	0.04
k_0	initial capital per effective working age person	varies

Write the non-linear difference equation as

$$k_{t+1} = \frac{s}{(1+g)(1+n)} k_t^{\alpha} + \frac{(1-\delta)}{(1+g)(1+n)} k_t \equiv \Psi(k_t)$$

Verify that there are two-steady states, points that solve $\bar{k} = \Psi(\bar{k})$, and that the non-trivial steady-state can be expressed as

$$\log \bar{k} = \frac{1}{1-\alpha} \log \left(\frac{s}{g+n+gn+\delta} \right)$$

Clearly, steady state effective capital per worker is increasing in capitals share α, increasing in the savings rate s, and decreasing in the growth rates of productivity g and work force n and in the physical depreciation rate. With the parameter values given above, verify that

$$\bar{k} = 3.9040$$

Also verify that the steady state capital/output ratio is about 2.5 and that the marginal product of capital (gross of depreciation) is about 13%.

Now compute a sequence $\{k_t\}$ until

$$\mid k - \bar{k} \mid < 10^{-6}$$

Plot the trajectory of $\{k_t\}$ against time t, and plot transitions $\{k_t\}$ against $\{k_{t+1}\}$ using $log(k_0/\bar{k}) = -0.5$, so that initially the capital stock is approximately 50% below its steady state level.

Figure 1: Time path of the capital stock

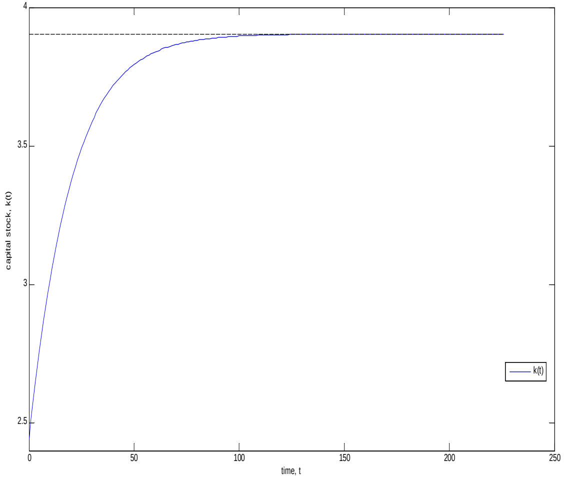
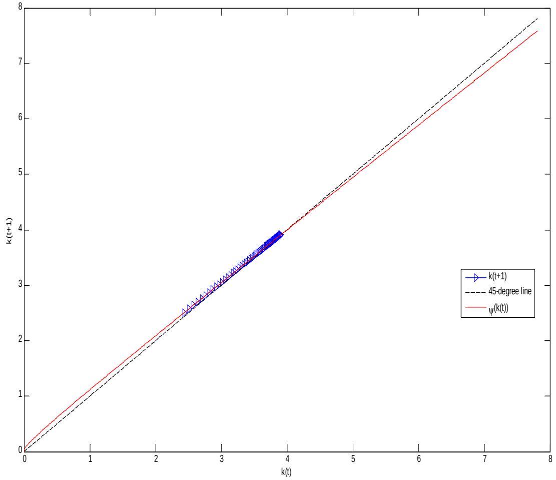


Figure 2: Phase diagram



Solving Ordinary differential equations (ODEs)

- Ordinary differential equations (ODEs) are used throughout engineering, mathematics, and science to describe how physical quantities change, so an introductory course on elementary ODEs and their solutions is a standard part of the curriculum in these fields.
- An ODE represents a relationship between a function and its derivatives. One such relation taken up early in calculus courses is the linear ordinary differential equation

  $$y^{'}(t) = y(t) \tag{1}$$

  Often solutions are specified by means of an initial value. For example, there is a unique solution of $y^{'}(t) = y(t))$ for which $y(0) = 1$, namely $y(t) = e^{t}$

Solving Ordinary differential equations (ODEs)

MATLAB has several built-in functions for solving initial-value problems. Two of them are ODE23 and ODE45.

The former calculates the solution by second- and third-order Runge-Kutta methods. The latter uses fourth- and fifth-order Runge-Kutta methods.

We will not go over the details of these methods in here.

The Runge-Kutta method is described in all numerical analysis textbooks.

Solving ODEs: Solow Growth

Suppose now we want to solve the Solow growth model

$$\dot{k}(t) = sA[k(t)]^{\alpha} - \delta k(t) \tag{2}$$

with $s = 0,4$, $A = 1$, $\alpha = 0,3$, and $\delta = 0,1$. This equation admits a unique steady state which is given by

$$k^{*} = \left(\frac{sA}{\delta}\right)^{\frac{1}{1-\alpha}} \tag{3}$$

We will solve the solution for (2) over the range $[k^{*}/2, k^{*}]$.

```
%First we write an M-file that contains the
%differential equation. Open an M-file and enter the followin

 function kdot = solow(t,k);
 global s A alpha delta;
 kdot = s*A*k^alpha - delta*k;

%Save this as solow in the current directory.
%Next open another M-file which will be the main program.

 close all;clear all;
 global s A alpha delta;
 s = 0.4;    A = 1;  alpha = 0.3;    delta = 0.1;
 k_ss = ( s*A/delta )^(1/(1 - alpha));
 k_0 = 0.5*k_ss;                   time = [0, 150];
 [t, k] = ode45('solow', time, k_0); plot(t, k);
% Save this as main_solow under the current directory.
% In the ode45 command, you have to specify
% the function that evaluates the right side of the
% differential equation, the time span, and the
% initial value. Choose a di erent time span,
% say time = [0,10], and see what happens.
```

Function-Solver in Matlab

MATLAB has a built-in function-solver called fsolve that solves equations of the form: $F(X) = 0$ for $F : R^n \rightarrow R^m$.

$$\begin{aligned} y &= \alpha x^\theta \\ \delta y &= \lambda x - \psi \end{aligned}$$

This system involves two variables (x, y) and five parameters

$(\alpha, \theta, \delta, \lambda, \psi)$.

We then specify their values in the main program.

```
%In the main program, we have
close all;  clear all;
global alpha theta delta lambda phi;
alpha = 2;
theta = 0.4;
delta = 2;
lambda = 1;
phi = 1;
guess = [1.2, 0.2];
sol = fsolve('m1', guess, optimset('TOLFUN', 1e-10, 'TOLX', 1

% In the function M-file we have:
function f = m1(unkown);
global alpha theta delta lambda phi;
x = unknown(1);
y = unknown(2);
f(1) = y - alpha*x^theta;
f(2) = delta*y - lambda*x + phi;
```

In here, sol is the vector that contains the solutions.

Inside the fsolve command, we need to provide the name of the function (the M-file containing the function must be in the current working directory) and an initial guess.

The rest are some optional features that the user can specify.

Enter help optimset in the command window to see the details of this function.

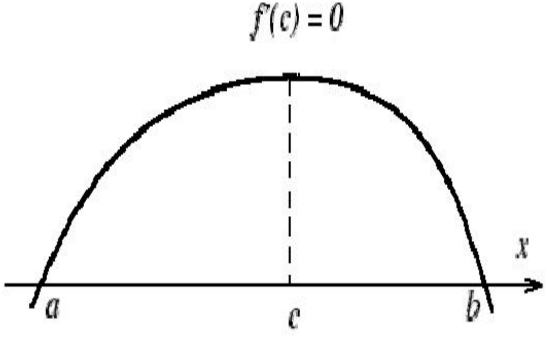
Introduction to Numerical Algorithms

- Numerical analysis is concerned with the processes by which mathematical problems can be solved by the operations of arithmetic, then the practice of numerical analysis requires that a problem statement be turned into a sequence of arithmetic operations which convert the data of the problem into the results.
- The sequence of arithmetic operations and the set of decisions which indicate which operation in the sequence to perform next constitute a rule, a recipe for solving the given problem.
- Such a rule in mathematics and computer science is called an algorithm.

Rolle's Theorem

Let f be continuous on $[a, b]$, and differentiable on (a, b), and suppose that $f(a) = f(b)$. Then there is some c with $a < c < b$ such that $f'(c) = 0$.

Note: The theorem guarantees that the point c exists somewhere. It gives no indication of how to find c. If f crosses the axis twice, somewhere between the two crossings, the function is flat. The accurate statement of this "obvious" observation is Rolle's Theorem. Here is the diagram to make the point geometrically:



The Mean Value Theorem

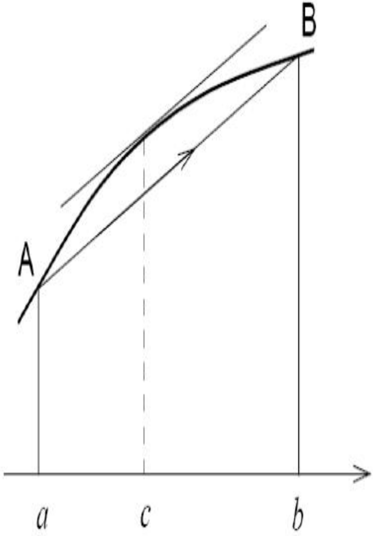
Let f be continuous on $[a, b]$, and differentiable on (a, b). Then there is some c with $a < c < b$ such that

$$\frac{f(b) - f(a)}{b - a} = f'(c)$$

or equivalently,

$$f(b) = f(a) + (b - a)f'(c)$$

We say that a root is **bracketed** in the interval (a, b) if $f(a)$ and $f(b)$ have the opposite signs. If the function is continuous, then at least one root must lie in that interval (the intermediate value theorem). Once we know that an interval contains a root, several classical procedures are available to refine it.



Somewhere inside a chord, the tangent to f will be parallel to the chord. The accurate statement of this common-sense observation is the Mean Value Theorem.

Iteration for Solving $x = g(x)$

- A fundamental principle in computer science is iteration
- As the name suggests, a process is repeated until an answer is achieved
- Iterative techniques are used to find roots of equations, solutions of linear and nonlinear systems of equations, and solutions of differential equations
- A rule or function $g(x)$ for computing successive terms is needed, together with a starting value p_0
- Then a sequence of values $\{p_k\}$ is obtained using the iterative rule $p_{k+1} = g(p_k)$

The sequence has the pattern

$$\begin{aligned} p_0 & \qquad \text{(starting value)} \\ p_1 &= g(p_0) \\ p_2 &= g(p_1) \\ &\vdots \\ p_k &= g(p_{k-1}) \\ p_{k+1} &= g(p_k) \end{aligned}$$

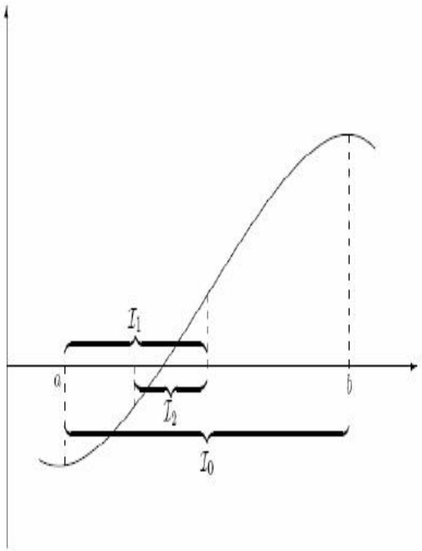
Definition: A **fixed point** of a function $g(x)$ is a real number P such that $P = g(P)$.
Geometrically, the fixed points of a function $y = g(x)$ are the points of intersection of $y = g(x)$ and $y = x$.
Definition: The iteration $p_{n+1} = g(p_n)$ for $n = 0, 1, ...$ is called **fixed-point iteration**.

Theorem: Assume that g is a continuous function and that $\{p_n\}_{t=0}^{\infty}$ is a sequence generated by fixed-point iteration. If $\lim_{n\to\infty} p_n = P$, then P is a fixed point of $g(x)$.

```
%%%% FIXED-POINT ITERATION %%%%
function fixedpoint(p0, N)
p0=1;          N=100;

i = 1;      p(1) = p0;          tol = 1e-05;
while i <= N
   p(i+1) = g(p(i));
   if abs(p(i+1)-p(i)) < tol              %stopping criterion
      disp('The procedure was successful after k iterations')
      k = i
      disp('The root to the equation is')
      p(i+1)
      return
   end
   i = i+1;
end

if abs(p(i)-p(i-1)) > tol | i > N
   disp('The procedure was unsuccessful')
   disp('Condition |p(i+1)-p(i)| < tol was not sastified')
end

function y = g(x)
y = x - (x^3 + 4*x^2 - 10)/(3*x^2 + 8*x);
```

The bisection method

- The idea is simple. Over some interval the function is known to pass through zero because it changes sign. Evaluate the function at the intervals midpoint and examine its sign. Use the midpoint to replace whichever limit has the same sign. After each iteration the bounds containing the root decrease by a factor of two.
- Problem: find x on $[a, b]$ such that $f(x) = 0$, where $f : \boldsymbol{R} \rightarrow \boldsymbol{R}$ is a continuous function and $f(a)$ and $f(b)$ have opposite signs. (Show picture of a function on $[a, b]$ with $f(a) < 0$ and $f(b) > 0$.)
- Idea: once we bracket a root, we simply divide the interval in half, figure out which of the two halves brackets the root, and repeat the process. (Show this on figure.)


I_1
a
b
I_2
I_0

The bisection method can be applied for solving nonlinear equations like $f(x) = 0$, only in the case where we know some interval $[a, b]$ on which $f(x)$ is continuous and the solution uniquely exists and, most importantly, $f(a)$ and $f(b)$ have the opposite signs.

If the interval happens to contain two or more roots, bisection will find one of them.

Description of the Algorithm

1. Define an interval $[a;b]$ $(a < b)$ on which we want to find a zero of f, such that $f(a)f(b) < 0$ and a stopping criterion $\varepsilon > 0$.

2. Set $x_0 = a$, $x_1 = b$, $y_0 = f(x_0)$ and $y_1 = f(x_1)$;

3. Compute the bisection of the inclusion interval

$$x_2 = \frac{x_0 + x_1}{2}$$

   and compute $y_2 = f(x_2)$.

4. Determine the new interval

   - If $y_0 y_2 < 0$, then $x^\star$ lies between x_0 and x_2 thus set

$$\begin{array}{ccc} x_0 = x_0 & , & x_1 = x_2 \\ y_0 = y_0 & , & y_1 = y_2 \end{array}$$

   - else $x^\star$ lies between x_1 and x_2 thus set

$$\begin{array}{ccc} x_0 = x_1 & , & x_1 = x_2 \\ y_0 = y_1 & , & y_1 = y_2 \end{array}$$

5. if $|x_1 - x_0| \leqslant \varepsilon(1 + |x_0| + |x_1|)$ then stop and set

$$x^\star = x_3$$

The False Position Method

Similarly to the bisection method, the false position method starts with the initial solution interval $[a, b]$ that is believed to contain the solution of $f(x) = 0$. Approximating the curve of $f(x)$ on $[a, b]$ by a straight line connecting the two points $(a, f(a))$ and $(b, f(b))$, it guesses that the solution may be the point at which the straight line crosses the x axis

The bisection method used the midpoint of the interval $[a, b]$ as the next iterate. A better approximation is obtained if we find the point $(c, 0)$ where the secant line joining the points $(a, f(a))$ and $(b, f(b))$ crosses the x axis. To find the value of c, we write down the two versions of the slope m of the line L:

$$m = \frac{f(b) - f(a)}{b - a}$$

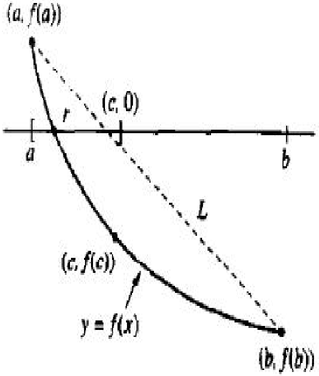
where the points $(a, f(a))$ and $(b, f(b))$ are used, and

$$m = \frac{0 - f(b)}{c - b}$$

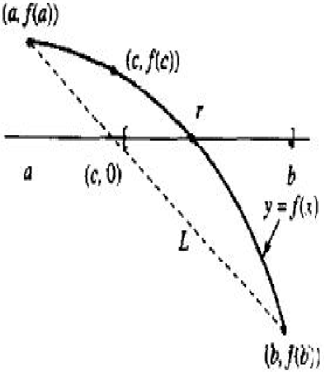
where the $(c, 0)$ and $(b, f(b))$ are used. Equating the slopes, we have

$$c = \frac{af(b) - bf(a)}{f(b) - f(a)}$$

The Decision process for the False Position Method



(a) If $f(a)$ and $f(c)$ have opposite signs then squeeze from the right.



(b) If $f(c)$ and $f(b)$ have opposite signs then squeeze from the left.

The Newton-Raphson method

- Problem: find x such that $f(x) = 0$, where $f : \boldsymbol{R}^n \to \boldsymbol{R}^n$ is a continuous function. (Use picture from bisection method – erase a, b since they are no longer inputs – and draw tangent lines to show updating scheme.)

- Idea: use a Taylor expansion

$$f(x) = f(\bar{x}) + f'(\bar{x})(x - \bar{x}) + \text{higher order terms}$$

1. where the expansion is taken around an approximate solution to $f(x) = 0$. Evaluate f at the solution. Assuming that $\bar{x}$ is sufficiently close to x^*, higher order terms are small and

   $$x^* = \bar{x} - f(\bar{x})/f'(\bar{x}).$$

   This leads to the following updating scheme:

   $$x^{k+1} = x^k - f(x^k)/f'(x^k),$$

The Newton-Raphson method

1. $k = 0, 1, \ldots$. For $n > 1$, the updating scheme is:

$$x^{k+1} = x^k - J(x^k)^{-1} f(x^k)$$

   where the i, j element of $J(x)$ is the derivative of the ith element of f with respect to the jth element of x.

- Issues: when you are not close to a solution, you can have problems. (Draw three examples: $f'(x^k) = 0$, $x^k = x^{k+2}$ and $x^{k+1} = x^{k+3}$, and x^k diverging.) However, when you are close to a solution, the method achieves quadratic convergence (i.e., $|x^{k+1} - x^*| \leq c|x^k - x^*|^2$).

The secant method

- The secant method uses the same updating scheme as Newton-Raphson but numerical derivatives are used for J. (See next section.)

Numerical differentiation

- Problem: find $df(x)/dx$.
- Idea: use Taylor expansions around a point x

$$
\begin{aligned}
f(x+h) &= f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + \text{higher order terms} \\
f(x-h) &= f(x) - f'(x)h + \frac{1}{2}f''(x)h^2 + \text{higher order terms}
\end{aligned}
$$

1. to get approximations

$$
\begin{aligned}
f'(x) &\approx \frac{f(x+h)-f(x)}{h} \\
f'(x) &\approx \frac{f(x)-f(x-h)}{h} \\
f'(x) &\approx \frac{f(x+h)-f(x-h)}{2h}
\end{aligned}
$$

2. and for the second derivative
   $f''(x) \approx \frac{1}{h^2}\{f(x+h) - 2f(x) + f(x-h)\}$.