

MATH 3795

Lecture 12. Numerical Solution of Nonlinear Equations.

Dmitriy Leykekhman

Fall 2008

Goals

- ▶ Learn about different methods for the solution of $f(x) = 0$, their advantages and disadvantages.
- ▶ Convergence rates.
- ▶ MATLAB's `fzero`

Nonlinear Equations.

Goal: Given a function $f : \mathbb{R} \rightarrow \mathbb{R}$ we want to find x_* such that $f(x_*) = 0$.

Definition

A point x_* with $f(x_*) = 0$ is called a **root** of f or **zero** of f .

Bisection Method.

- **Intermediate Value Theorem:** If $f : \mathbb{R} \rightarrow \mathbb{R}$ is a continuous function and $a, b \in \mathbb{R}$, $a < b$, then for every y between $f(a)$ and $f(b)$, i.e.,

$$y \in [\min \{f(a), f(b)\}, \max \{f(a), f(b)\}],$$

there exists $x \in [a, b]$, such that $f(x) = y$.

Bisection Method.

- ▶ **Intermediate Value Theorem:** If $f : \mathbb{R} \rightarrow \mathbb{R}$ is a continuous function and $a, b \in \mathbb{R}$, $a < b$, then for every y between $f(a)$ and $f(b)$, i.e.,

$$y \in [\min \{f(a), f(b)\}, \max \{f(a), f(b)\}],$$

there exists $x \in [a, b]$, such that $f(x) = y$.

- ▶ Given $[a_k, b_k]$ with $f(a_k)f(b_k) < 0$ (i.e., $f(a_k)$ and $f(b_k)$ have opposite signs, compute the interval midpoint $c_k = \frac{1}{2}(a_k + b_k)$ and evaluates $f(c_k)$.
- ▶ If $f(c_k)$ and $f(a_k)$ have opposite sign, then $[a_k, c_k]$ contains a root of f . Sets $a_{k+1} = a_k$ and $b_{k+1} = c_k$.
- ▶ Otherwise $f(c_k)$ and $f(b_k)$ must the opposite sign. In this case, $[c_k, b_k]$ contains a root of f and we sets $a_{k+1} = c_k$ and $b_{k+1} = b_k$.

Bisection Method.

Input: Initial values $a(0)$; $b(0)$ such that $f(a(0))f(b(0)) < 0$
and a tolerance tolx

Output: approximation of a root of f

```
for k = 0, 1, ... do
    if b(k)-a(k) < tolx,
        return c(k) = (a(k) + b(k))/2 as an approximate root of
    end
    Compute c(k) = (a(k) + b(k))/2 and f(c(k)).
    if f(c(k)) = 0,    x = c(k);    end

    if f(a(k))f(c(k)) < 0 , then
        a(k+1) = a(k);
        b(k+1) = c(k);
    else
        a(k+1) = c(k);
        b(k+1) = b(k);
    end
end
end
```

Analysis of the Bisection Method.

- ▶ In each step the interval $[a_k, b_k]$ is halved. Hence

$$\begin{aligned}|a_{k+1} - b_{k+1}| &= \frac{1}{2}|a_k - b_k| \\ &= \frac{1}{2^2}|a_{k-1} - b_{k-1}| \\ &= \cdots = \frac{1}{2^{k+1}}|a_0 - b_0|\end{aligned}$$

- ▶ There is a root of f such that the midpoints satisfy

$$|x_* - c_k| \leq 2^{-k+1}|b_0 - a_0|.$$

- ▶ In particular, $\lim_{k \rightarrow \infty} c_k = x_*$.
- ▶ By $\lfloor z \rfloor$ we denote the largest integer less or equal to z . After

$$k = \left\lfloor \log_2 \left(\frac{|b_0 - a_0|}{tol_x} \right) \right\rfloor - 1$$

iterations we are guaranteed to have an approximation c_k of a root x_* of f that satisfies $|x_* - c_k| \leq tol_x$.

Bisection Method.

- + The Bisection method requires only function values.
- + In fact, the Bisection method only requires the sign of function values (that means as long as the sign is correct, the function values can be inaccurate).
- + The Bisection method converges for any $[a_0, b_0]$ that brackets a root of f (the Bisection method converges globally).
 - The Bisection method only requires the sign of function values. In general, it will not find the root of the simple affine linear function $f(x) = ax + b$ in a finite number of iterations.
 - The local convergence behavior of the Bisection method is rather slow (the error only reduced by a factor 2, no matter how close we are to the solution).

Regula Falsi.

- ▶ Given an initial interval $[a_0, b_0]$ such that $f(a_0)f(b_0) < 0$ (i.e., $[a_0, b_0]$ contains a root of f), Regula Falsi constructs an affine linear function $m(x) = \alpha x + \beta$ such that $m(a_0) = f(a_0)$ and $m(b_0) = f(b_0)$. One says that m interpolates f at a_0 and at b_0 . This particular interpolation is also known as the structuresecant of f .
- ▶ The secant of f at a_0 and at b_0 is given by

$$m(x) = f(a_0) + (f(b_0) - f(a_0)) \frac{x - a_0}{b_0 - a_0}$$

- ▶ The function m is used as a model for f and the root

$$c_0 = a_0 - \frac{b_0 - a_0}{f(b_0) - f(a_0)} f(a_0)$$

of the model is used as an approximation of the root of f .

- ▶ This root satisfies $c_0 \in (a_0, b_0)$.
- ▶ If $f(c_0)$ and $f(a_0)$ have opposite signs, then we set $a_1 = a_0$ and $b_1 = c_0$. Otherwise $f(c_0)$ and $f(b_0)$ must have opposite signs and we sets $a_1 = c_0$ and $b_1 = b_0$.

Regula Falsi.

Input: Initial values $a(0)$; $b(0)$ such that $f(a(0)) \cdot f(b(0)) < 0$,
a maximum number of iterations maxit , a tolerance tolf and a tol

Output: approximation x of the root

```
1 For  $k = 0, 1, \dots, \text{maxit}$  do
2 If  $b(k) - a(k) < \text{tolx}$ , then return  $x = c(k)$  and stop
3 Compute  $c(k) = a(k) - f(a(k))(b(k) - a(k))(f(b(k)) - f(a(k)))$ 
   and  $f(c(k))$ .
4 If  $|f(c(k))| < \text{tolf}$  , then return  $x = c(k)$  and stop
5 If  $f(a(k))f(c(k)) < 0$  , then
6  $a(k+1) = a(k)$ ;  $b(k+1) = c(k)$ ,
7 else
8  $a(k+1) = c(k)$ ;  $b(k+1) = b(k)$ .
9 End
10 End
```

Newton's Method.

- ▶ Suppose we are given an approximation x_0 of a root x_* of f .
- ▶ Taylor approximation of f around x_0 gives

$$f(x_*) = f(x_0 + (x_* - x_0)) \approx f(x_0) + f'(x_0)(x_* - x_0).$$

- ▶ We use the tangent of f at x_0 ,

$$m(x) = f(x_0) + f'(x_0)(x - x_0)$$

as a model for f .

- ▶ The root

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

of the tangent model is used as an approximation of the root of f .

- ▶ Write the previous identity as

$$s_0 = -f(x_0)/f'(x_0) \quad \text{step (correction)}$$

$$x_1 = x_0 + s_0.$$

Newton's Method.

Input: Initial values $x(0)$, tolerance tol ,
maximum number of iterations maxit

Output: approximation of the root

```
1 For k = 0:maxit do
2 Compute  $s(k) = -f(x(k))/f'(x(k))$ .
3 Compute  $x(k+1) = x(k) + s(k)$ .
4 Check for truncation
5 End
```

Convergence of Sequences.

Let $\{x_k\}$ be a sequence of real numbers.

1. The sequence is called **q -linearly convergent** if there exists $c \in (0, 1)$ and $\hat{k} \in \mathbb{N}$ such that

$$|x_{k+1} - x_*| \leq c|x_k - x_*| \quad \text{for all } k \geq \hat{k}.$$

2. The sequence is called **q -superlinearly convergent** if there exists a sequence $\{c_k\}$ with $c_k > 0$ and $\lim_{k \rightarrow \infty} c_k = 0$ such that

$$|x_{k+1} - x_*| \leq c_k |x_k - x_*|$$

or, equivalently, if

$$\lim_{k \rightarrow \infty} \frac{|x_{k+1} - x_*|}{|x_k - x_*|} = 0.$$

3. The sequence is called **q -quadratically convergent** to x_* if $\lim_{k \rightarrow \infty} x_k = x_*$ and if there exists $c > 0$ and $\hat{k} \in \mathbb{N}$ such that

$$|x_{k+1} - x_*| \leq c|x_k - x_*|^2 \quad \text{for all } k \geq \hat{k}.$$

Convergence of Newton's Method.

Theorem

Let $D \subset \mathbb{R}$ be an open interval and let $f : D \rightarrow \mathbb{R}$ be differentiable on D . Furthermore, let f' be Lipschitz continuous with Lipschitz constant L . If $x_ \in D$ is a root and if $f'(x_*) \neq 0$, then there exists an $\epsilon > 0$ such that Newton's method with starting point x_0 with $|x_0 - x_*| < \epsilon$ generates iterates x_k which converge to x_* ,*

$$\lim_{k \rightarrow \infty} x_k = x_*,$$

and which obey

$$|x_{k+1} - x_*| \leq \frac{L}{|f'(x_*)|} |x_k - x_*|^2 \quad \text{for all } k \in \mathbb{N}.$$

Convergence of Newton's Method.

Theorem

Let $D \subset \mathbb{R}$ be an open interval and let $f : D \rightarrow \mathbb{R}$ be differentiable on D . Furthermore, let f' be Lipschitz continuous with Lipschitz constant L . If $x_ \in D$ is a root and if $f'(x_*) \neq 0$, then there exists an $\epsilon > 0$ such that Newton's method with starting point x_0 with $|x_0 - x_*| < \epsilon$ generates iterates x_k which converge to x_* ,*

$$\lim_{k \rightarrow \infty} x_k = x_*,$$

and which obey

$$|x_{k+1} - x_*| \leq \frac{L}{|f'(x_*)|} |x_k - x_*|^2 \quad \text{for all } k \in \mathbb{N}.$$

Newton's method is locally q-quadratically convergent (under the assumptions stated in the theorem).

Variations of Newton's Method.

Recall Newton's Method:

Input: Initial values x_0 ,

tolerance tol , maximum number of iterations maxit

Output: approximation of the root

1. For $k = 0, \dots, \text{maxit}$ do
2. Compute $s_k = -f(x_k)/f'(x_k)$.
3. $x_{k+1} = x_k + s_k$.
4. Check for truncation
5. End

Requires the evaluation of derivatives. If derivative evaluations are expensive or difficult to compute, the following variations are useful

Finite difference Newton method and **Secant method**.

Finite Difference Newton Method.

Recall

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}.$$

In the finite difference Newton method we replace $f'(x_k)$ by the finite difference $(f(x_k + h_k) - f(x_k))/h_k$ for some $h_k \neq 0$.

Input: Initial values x_0 ,

tolerance tol , maximum number of iterations maxit

Output: approximation of the root

1. For $k = 0, \dots, \text{maxit}$ do
2. Compute $s_k = -f(x_k)/(f(x_k + h_k) - f(x_k))$.
3. $x_{k+1} = x_k + s_k$.
4. Check for truncation
5. End

Secant Method.

If we use $h_k = x_{k-1} - x_k$, then

$$\frac{f(x_k + h_k) - f(x_k)}{h_k} = \frac{f(x_k + h_k) - f(x_k)}{x_{k-1} - x_k}.$$

Input: Initial values x_0 ,

tolerance tol , maximum number of iterations maxit

Output: approximation of the root

- 1 For $k = 0, \dots, \text{maxit}$ do
 - 2 Compute $s_k = -f(x_k)/(f(x_k + h_k) - f(x_k))$.
 - 3 $x_{k+1} = x_k + s_k$.
 - 4 Check for truncation
 - 5 End
-

Stopping Criteria.

We have discussed iterative methods which generate sequences x_k with (under suitable conditions) $\lim_{k \rightarrow \infty} x_k = x_*$. When do we stop the iteration?

- ▶ For a given tolerance $tol_a > 0$ we want to find x_k such that
 $|x_k - x_*| < tol_a$, stop if absolute error is small; or
 $|x_k - x_*| < tol_r |x_*|$, stop if relative error is small.

Stopping Criteria.

We have discussed iterative methods which generate sequences x_k with (under suitable conditions) $\lim_{k \rightarrow \infty} x_k = x_*$. When do we stop the iteration?

- ▶ For a given tolerance $tol_a > 0$ we want to find x_k such that
 $|x_k - x_*| < tol_a$, stop if absolute error is small; or
 $|x_k - x_*| < tol_r |x_*|$, stop if relative error is small.
- ▶ For some $t_k \in [0, 1]$

$$\begin{aligned} |f(x_k)| &= |f(x_k) - f(x_*)| = |f'(x_k + t_k(x_* - x_k))| |x_k - x_*| \\ &\geq \frac{1}{2} |f'(x_*)| |x_k - x_*|, \quad \text{if } x_k \text{ is suff. close to } x_* \end{aligned}$$

Hence, if $|f(x_k)| < tol_f$, and if x_k is sufficiently close to x_* , then

$$|x_k - x_*| < 2tol_f |f'(x_*)|^{-1}.$$

Stopping Criteria.

We have discussed iterative methods which generate sequences x_k with (under suitable conditions) $\lim_{k \rightarrow \infty} x_k = x_*$. When do we stop the iteration?

- ▶ For a given tolerance $tol_a > 0$ we want to find x_k such that $|x_k - x_*| < tol_a$, stop if absolute error is small; or $|x_k - x_*| < tol_r |x_*|$, stop if relative error is small.
- ▶ For some $t_k \in [0, 1]$

$$\begin{aligned} |f(x_k)| &= |f(x_k) - f(x_*)| = |f'(x_k + t_k(x_* - x_k))| |x_k - x_*| \\ &\geq \frac{1}{2} |f'(x_*)| |x_k - x_*|, \quad \text{if } x_k \text{ is suff. close to } x_* \end{aligned}$$

Hence, if $|f(x_k)| < tol_f$, and if x_k is sufficiently close to x_* , then

$$|x_k - x_*| < 2tol_f |f'(x_*)|^{-1}.$$

- ▶ Limit maximum number of iterations.

Stopping Criteria.

- ▶ Stopping when $|x_{k+1} - x_k|$ is small is not a good criteria.
Example

$$f(x) = x \quad \text{and} \quad x_k = 3 - \sum_{i=0}^k \frac{1}{2^i}$$

Then

$$|x_{k+1} - x_k| = \frac{1}{2^k} \rightarrow 0$$

but

$$\lim_{k \rightarrow \infty} x_k = 3 - \sum_{i=0}^{\infty} \frac{1}{2^i} = 1.$$

- ▶ For the same reason stopping when $|f(x_{k+1}) - f(x_k)|$ is small is not a good criteria

MATLAB's `fzero`.

MATLAB has a build-in function called **`fzero`** that tries to find a zero of a function of one variable.

Syntax:

```
x = fzero(fun,x0)
x = fzero(fun,x0,options)
x = fzero(fun,x0,options,P1,P2,...)
[x,fval] = fzero(...)
[x,fval,exitflag] = fzero(...)
[x,fval,exitflag,output] = fzero(...)
```

MATLAB's `fzero`.

```
x = fzero(fun,x0)
```

tries to find a zero of `fun` near `x0`, if `x0` is a scalar. The value `x` returned by `fzero` is near a point where `fun` changes sign, or `NaN` if the search fails. In this case, the search terminates when the search interval is expanded until an `Inf`, `NaN`, or complex value is found.

If `x0` is a vector of length two, `fzero` assumes `x0` is an interval where the sign of `fun(x0(1))` differs from the sign of `fun(x0(2))`. An error occurs if this is not true.

MATLAB's fzero.

Calling

```
x = fzero(fun,x0)
```

you must specify the function `fun`. There are several ways to do that. MATLAB has many build-in functions, such as `sin`, `cos`, `tan`, etc. The syntax for using such functions is

```
x = fzero(@fun,x0)
```

or

```
x = fzero('fun',x0)
```

MATLAB's fzero.

Example

Calling

```
x = fzero(@sin,3)
```

tells MATLAB to look for the root near 3 and produces

```
x = 3.141592653589793
```

Calling

```
x = fzero('tan',[-.5, 1])
```

tells MATLAB to look for the root on the interval $[-.5, 1]$ and produces

```
x = -1.128055202807828e-016
```

MATLAB's fzero.

Example

On the other hand, calling

```
x = fzero(@cos, [0 1])
```

gives you an error since `cos` does not change the sign on $[0, 1]$.
You can observe more severe limitations of `fzero` if you call

```
x = fzero(@tan, [1 3])
```

MATLAB produces

```
x = 1.570796326794897
```

which is not a zero of `tan`. Moreover this number is close to $\pi/2$ where `tan` is unbounded. Which is easy to check by typing

```
tan(x)
```

```
ans = -1.653161781927095e+015
```

MATLAB's fzero.

There are several ways to create your own function. One way is to use `inline` command.

Example

```
f = inline('x^3-2*x-5')
```

produces

```
f =
```

```
    Inline function:  
    f(x) = x^3-2*x-5
```

Now to find a zero closest to 1 just type

```
x = fzero(f, 1)
```

which gives

```
x = 2.094551481542327
```

MATLAB's fzero.

Another way is to create a function in a m-file.

```
function f = my_strange_function(x)

f = zeros(size(x));
for i = 1:length(x)
    if x(i) >= 1
        f(i) = log(x(i));
    elseif (x(i) < 1) && (x(i) > -1)
        f(i) = sin(x(i))*(x(i)^2-1);
    else
        f(i) = cos(x(i))^3*(1+x(i)^2);
    end
end
```

Now to find the zero closest to -2 just type

```
x = fzero('my_strange_function', -2)
```

which gives

```
x = -1.570796326794897
```

MATLAB's fzero.

fzero has some other options. To see them all type `help fzero`.
For example

```
x = fzero(@sin,3,optimset('Display','iter'))
```

returns a root close to π and uses the default tolerance and displays iteration information.

MATLAB's fzero.

Search for an interval around 3 containing a sign change:

Func-count	a	f(a)	b	f(b)
1	3	0.14112	3	0.14112
3	2.91515	0.224515	3.08485	0.0567094
5	2.88	0.258619	3.12	0.021591
7	2.83029	0.306295	3.16971	-0.0281093

Search for a zero in the interval [2.83029, 3.16971]:

Func-count	x	f(x)	Procedure
7	3.16971	-0.0281093	initial
8	3.14118	0.000417192	interpolation
9	3.14159	-5.41432e-008	interpolation
10	3.14159	1.45473e-015	interpolation
11	3.14159	1.22465e-016	interpolation
12	3.14159	1.22465e-016	interpolation

Zero found in the interval [2.83029, 3.16971]

X = 3.141592653589793