Math 541 - Numerical Analysis

Lecture Notes – Zeros and Roots

Joseph M. Mahaffy,
⟨jmahaffy@mail.sdsu.edu⟩

Department of Mathematics and Statistics
Dynamical Systems Group
Computational Sciences Research Center
San Diego State University
San Diego, CA 92182-7720

http://jmahaffy.sdsu.edu

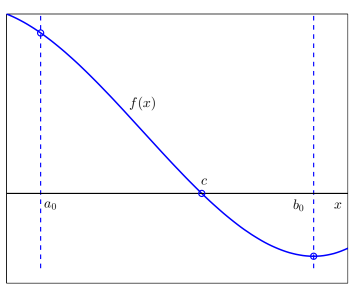
Spring 2018

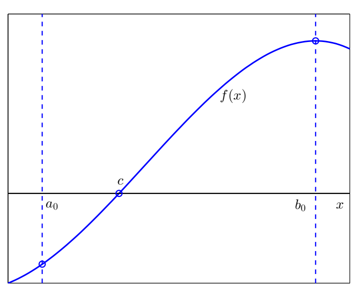
Outline

1. Bisection Method
   - Example and Program
   - Rate of Convergence
2. Newton's Method
   - Tangent Lines
   - Iterative Scheme
   - MatLab – Newton's Method
   - Rate of Convergence
3. Secant Method
   - MatLab - Secant Method
   - Rate of Convergence
4. Summary
   - Example
   - Modifying Newton's Method

Intermediate Value Theorem

- Suppose f is continuous on the interval (a_0, b_0) and $f(a_0) \cdot f(b_0) < 0$
  - This means the function changes sign at least once in the interval
- The **Intermediate Value Theorem** guarantees the existence of $c \in (a_0, b_0)$ such that $f(c) = 0$ (could be more than one)





The Bisection Method

The **Bisection Method** approximates the root ($f(c) = 0$) of a **continuous function** that changes sign at least once for $x \in [a_0, b_0]$

- Thus, $f(a_0) \cdot f(b_0) < 0$
- Iteratively find the **midpoint**

$$m_k = \frac{a_k + b_k}{2}$$

- If $f(m_k) = 0$, we're done
- Check if $f(m_k) \cdot f(b_k) < 0$ or $f(m_k) \cdot f(a_k) < 0$
- If $f(m_k) \cdot f(b_k) < 0$, then $c \in [m_k, b_k]$ and we take $a_{k+1} = m_k$ and $b_{k+1} = b_k$
- Otherwise $f(m_k) \cdot f(a_k) < 0$, and $c \in [a_k, m_k]$, so we take $b_{k+1} = m_k$ and $a_{k+1} = a_k$

The Bisection Method

The **Bisection Method** for solving $f(c) = 0$ from the previous slide:

- Constructs a sequence of intervals containing the root c:

$$(a_0, b_0) \supset (a_1, b_1) \supset \cdots \supset (a_{n-1}, b_{n-1}) \supset (a_n, b_n) \ni c$$

- After k steps

$$|b_k - a_k| = \frac{1}{2}|b_{k-1} - a_{k-1}| = \left(\frac{1}{2}\right)^k |b_0 - a_0|$$

- At step k, the midpoint $m_k = \frac{a_k + b_k}{2}$ is an estimate for the root c with

$$m_k - d_k \leq c \leq m_k + d_k, \qquad d_k = \left(\frac{1}{2}\right)^{k+1} |b_0 - a_0|$$

The Bisection Method

Convergence is slow:

- At each step we gain ***one binary digit in accuracy***
- Since $10^{-1} \approx 2^{-3.3}$, it takes on average 3.3 iterations to gain one decimal digit of accuracy
- **Note:** The rate of convergence is completely independent of the function f
- We are only using the **sign of** f at the endpoints of the interval(s) to make decisions on how to update
- By making more effective use of the values of f we can attain significantly faster convergence

Example: Bisection Method

The bisection method applied to

$$f(x) = \left(\frac{x}{2}\right)^2 - \sin(x) = 0$$

with $(a_0, b_0) = (1.5, 2.0)$, and $(f(a_0), f(b_0)) = (-0.4350, 0.0907)$ gives:

k	a_k	b_k	m_k	$f(m_k)$
0	1.5000	2.0000	1.7500	–0.2184
1	1.7500	2.0000	1.8750	–0.0752
2	1.8750	2.0000	1.9375	0.0050
3	1.8750	1.9375	1.9062	–0.0358
4	1.9062	1.9375	1.9219	–0.0156
5	1.9219	1.9375	1.9297	–0.0054
6	1.9297	1.9375	1.9336	–0.0002
7	1.9336	1.9375	1.9355	0.0024
8	1.9336	1.9355	1.9346	0.0011
9	1.9336	1.9346	1.9341	0.0004

Example: Bisection Method

This **MatLab code** can easily be modified to any ***function***

```
function root = bisection(a,b,tol)
%BISECTION METHOD - Modify function below, then can
%   find its roots in [a,b] to tolerance tol
f = @(x) (x/2).^2 - sin(x);
while (abs(b-a) ≥ tol)
    m = (a+b)/2;
    if (f(m) == 0)
        break;
    elseif(f(b)*f(m) < 0)
        a = m;
    else
        b = m;
    end
end
root = m;
```

Demonstrate in class

Example: Bisection Method

This **MatLab code** (2 Slides) graphically shows example

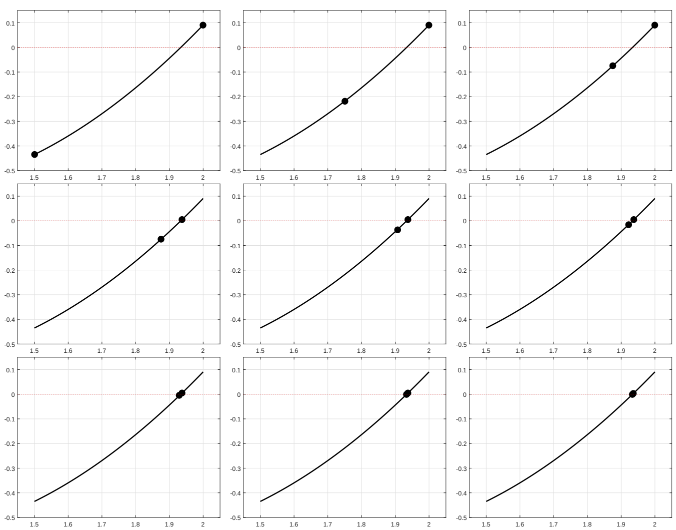
```
% WARNING: This example ASSUMES that f(a)<0<f(b)...
x = 1.5:0.001:2;
f = inline('(x/2).^2-sin(x)','x');
a = 1.5;
b = 2.0;
for k = 0:9
plot(x,f(x),'k-','linewidth',2)
axis([1.45 2.05 -0.5 .15])
grid on
hold on
plot([a b],f([a b]),'ko','linewidth',5)
plot([1.45 2.05],[0 0],'r:')
hold off
```


Example: Bisection Method


```
m = (a+b)/2;
if( f(m) < 0 )
a = m;
else
b = m;
end
pause
print('-depsc',['bisec' int2str(k) '.eps'],'-f1');
end
```

The next Slides show the **output**

Example: Bisection Method



Stopping Criteria

When do we stop?

We can *(1)* keep going until successive iterates are close:

$$|m_k - m_{k-1}| < \epsilon$$

or *(2)* close in relative terms

$$\frac{|m_k - m_{k-1}|}{|m_k|} < \epsilon$$

or *(3)* the function value is small enough

$$|f(m_k)| < \epsilon$$

No choice is perfect. In general, where no additional information about f is known, the second criterion is the preferred one (since it comes the closest to testing the relative error).

Rate of Convergence

Suppose an **algorithm** generates a **sequence** of **approximations**, c_n, which approaches a limit, c_*, or

$$\lim_{n\to\infty} c_n = c_*$$

How quickly does $c_n \to c_*$?

Definition (Rate of Convergence)

If a sequence $c_1, c_2, \dots, c_n$ converges to a value c_* and if there exist real numbers $\lambda > 0$ and $\alpha \geq 1$ such that

$$\lim_{n\to\infty} \frac{|c_{n+1} - c_*|}{|c_n - c_*|^\alpha} = \lambda$$

then we say that α is the ***rate of convergence*** of the sequence.

Cauchy Sequence

Definition (Cauchy Sequence)

Consider a sequence $c_1, c_2, \dots, c_n$ of real numbers. This sequence is called a ***Cauchy sequence***, if for every $\varepsilon > 0$, there is a positive integer N such that for all natural numbers $m, n > N$,

$$|x_m - x_n| < \varepsilon.$$

From the properties of real numbers (***completeness***), a ***Cauchy sequence***, $\{c_n\}$ converges to a unique real number c_*.

Rate of Convergence - Cauchy

A **Numerical algorithm** produces a sequence of **approximations**, $\{c_n\}$, which is hopefully converging to a limit, c_*, which is **NOT** known.

How can the sequence $\{c_n\}$ be used to find the ***rate of convergence***?

Definition (Rate of Cauchy Convergence)

If a sequence $c_1, c_2, \dots, c_n$ converges and if there exist real numbers $\lambda > 0$ and $\alpha \geq 1$ such that

$$\lim_{n\to\infty} \frac{|c_{n+1} - c_n|}{|c_n - c_{n-1}|^\alpha} = \lambda$$

then we say that α is the ***rate of convergence*** of the sequence.

Numerical Rate of Convergence

Suppose a **Numerical algorithm** produces a sequence of **approximations**, $\{c_n\}$.

The ***rate of Cauchy convergence***, α, for the sequence $c_1, c_2, \dots, c_n$ is derived from the existence of real numbers $\lambda > 0$ and $\alpha \geq 1$ with

$$\lim_{n\to\infty} \frac{|c_{n+1} - c_n|}{|c_n - c_{n-1}|^\alpha} = \lambda.$$

By taking logarithms of the expression above, we have

$$\ln|c_{n+1} - c_n| = \alpha \ln|c_n - c_{n-1}| + \ln(\lambda).$$

Let $Y_n = \ln|c_{n+1} - c_n|$ and $X_n = \ln|c_n - c_{n-1}|$, which are readily computed from the sequence, then the ***rate of Cauchy convergence***, α, is approximated by the ***slope*** of the best fitting line through (X_n, Y_n).

Rate of Convergence - Bisection Method

Let c_* be a **root** of f, so $f(c_*) = 0$

Let $m_n = \frac{a_n+b_n}{2}$ be the **midpoint** between the endpoints of the interval $[a_n, b_n]$, which comes from n iterations of the **Bisection Method** and where $c_* \in [a_n, b_n]$

Earlier we showed that at most

$$|m_n - c_*| \le \frac{1}{2^{n+1}}|b_0 - a_0|$$

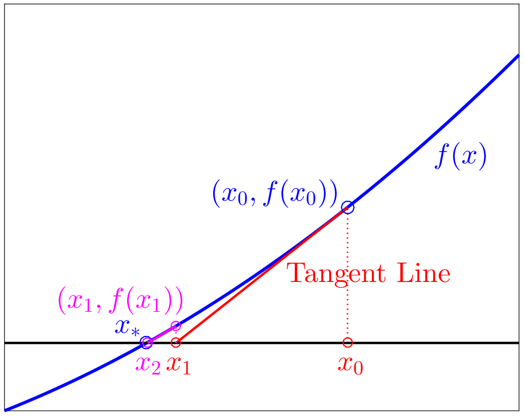
Form the ***sequence of midpoints*** with $c_n = m_n$, then from the worst case scenario

$$|c_n - c_*| \le \frac{1}{2^{n+1}}|b_0 - a_0| \qquad \text{or} \qquad \frac{|c_{n+1} - c_*|}{|c_n - c_*|} \approx \frac{1}{2}$$

It follows that for **Bisection Method** $\alpha = 1$, so the ***rate of convergence*** is **linear**

Tangent Lines

Start at x_0, then follow ***tangent lines*** of $f(x)$ to their zeroes.
Iterate these zeroes **converging** to $\{x_n\}_{n=0}^{\infty} \to x_*$ with $f(x_*) = 0$.



Tangent Lines

- The graphic from previous slide seems to show rapid convergence to the zero of $f(x)$
- The graph shows the use of properties of $f(x)$
- Does this sequence always converge to x_*?
- Assuming convergence, how rapidly does this sequence converge?
- The Method employs techniques from **Calculus**
- Technique is called **Newton's Method**
- What are properties of Newton's Method?

Newton's Method for Root Finding

Recall: we are looking for x^* so that $f(x^*) = 0$.

If $f \in C^2[a, b]$, and we know $x^* \in [a, b]$ (possibly by the intermediate value theorem), then we can formally Taylor expand around a point x close to the root:

$$0 = f(x^*) = f(x) + (x^* - x)f'(x) + \frac{(x - x^*)^2}{2} f''(\xi(x)), \quad \xi(x) \in [x, x^*].$$

If we are close to the root, then $|x - x^*|$ is small, which means that $|x - x^*|^2 \ll |x - x^*|$, hence we make the approximation:

$$0 \approx f(x) + (x^* - x)f'(x), \quad \Leftrightarrow \quad x^* \approx x - \frac{f(x)}{f'(x)}.$$

Newton's Method for Root Finding

Newton's Method for root finding is based on the approximation

$$x^* \approx x - \frac{f(x)}{f'(x)}$$

which is valid when x is close to x^*.

Newton's Method

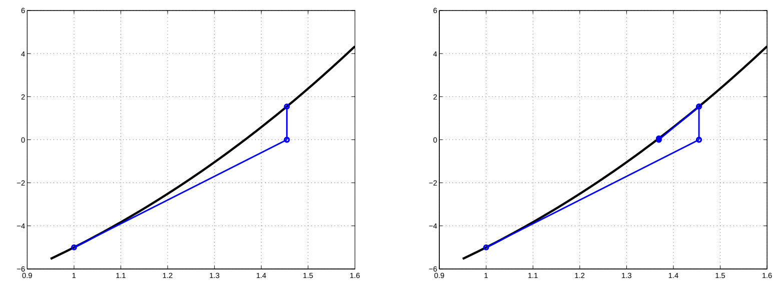
Newton's Method is an **iterative scheme**, where given an x_{n-1}, we compute

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

If x_0 is *"sufficiently close"* to a **root**, x^*, of $f(x)$, then iterations x_n give ***improved approximations*** of x^*, as $n \to \infty$

Geometrically, x_n is the intersection of the tangent of the function at x_{n-1} and the x-axis.

Two Steps of Newton for $f(x) = x^3 + 4x^2 - 10 = 0$



Start with $p_0 = 1$

$$p_1 = p_0 - \frac{p_0^3 + 4p_0^2 - 10}{3p_0^2 + 8p_0} = 1.45454545454545$$

$$p_2 = p_1 - \frac{p_1^3 + 4p_1^2 - 10}{3p_1^2 + 8p_1} = 1.36890040106952$$

$p^* = 1.365230013$ From MAPLE

MatLab Newton Code

```
function p = newton(p0,tol,Nmax)
%NEWTON'S METHOD: Enter f(x), f'(x), x0, tol, Nmax
f = @(x) x^3 + 4*x^2 - 10;
fp = @(x) 3*x^2 + 8*x;
p = p0 - f(p0)/fp(p0);
i = 1;
while (abs(p - p0) ≥ tol)
    p0 = p;
    p = p0 - f(p0)/fp(p0);
    i = i + 1;
    if (i ≥ Nmax)
        fprintf('Fail after %d iterations\n',Nmax);
        break
    end
end
end
```


Finding a Starting Point for Newton's Method

Recall our initial argument that when $|x - x^*|$ is small, then $|x - x^*|^2 \ll |x - x^*|$, and we can neglect the second order term in the Taylor expansion.

In order for Newton's method to converge we need a ***good starting point!***

Theorem

Let $f(x) \in C^2[a,b]$. If $x^* \in [a,b]$ such that $f(x^*) = 0$ and $f'(x^*) \neq 0$, then there exists a $\delta > 0$ such that Newton's method generates a sequence $\{x_n\}_{n=1}^{\infty}$ converging to x^* for any initial approximation $x_0 \in [x^* - \delta, x^* + \delta]$.

The theorem is interesting, but quite useless for practical purposes. In practice: Pick a starting value x_0, iterate a few steps. Either the iterates converge quickly to the root, or it will be clear that convergence is unlikely.

Newton's Method – Rate of Convergence

Suppose x^* is a root of $f(x)$ and consider the Taylor expansion of $f(x^*)$ about x_n

$$0 = f(x^*) = f(x_n) + f'(x_n)(x^* - x_n) + \frac{f''(\xi_n)}{2}(x^* - x_n)^2,$$

where $\xi_n \in (x^*, x_n)$

Dividing by $f'(x_n)$ gives

$$\frac{f(x_n)}{f'(x_n)} + (x^* - x_n) = -\frac{f''(\xi_n)}{2f'(x_n)}(x^* - x_n)^2$$

but $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$, so

$$(x^* - x_{n+1}) = -\frac{f''(\xi_n)}{2f'(x_n)}(x^* - x_n)^2$$

Newton's Method – Rate of Convergence

Taking absolute values

$$\frac{|x^* - x_{n+1}|}{|x^* - x_n|^2} = \frac{|f''(\xi_n)|}{2|f'(x_n)|},$$

so by our definition for ***rate of convergence***, **Newton's Method** has ***quadratic convergence*** provided

1. $f'(x) \neq 0$, for all $x \in I$, where $I = [x^* - r, x^* + r]$ for some $r \geq |x^* - x_0|$
2. $f''(x)$ is continuous for all $x \in I$
3. x_0 is "sufficiently close" to x^*

Newton's Method – Rate of Convergence

"Sufficiently close" means

- We can ignore higher order terms of the Taylor expansion
- $\frac{|f''(x_n)|}{2|f'(x_n)|} < C\frac{|f''(x^*)|}{|f'(x^*)|}$ for some $C < \infty$
- $C\frac{|f''(x^*)|}{|f'(x^*)|}|x^* - x_n| < 1$ for all n

If

$$M = \sup_{x \in I} \frac{|f''(x)|}{2|f'(x)|},$$

we have convergence for an initial point x_0 provided $M|x^* - x_0| < 1$

MatLab Newton Example


```
function z = newtoneger(p0,tol,Nmax)
%NEWTON'S METHOD: Enter f(x), f'(x), x0, tol, Nmax
f = @(x) x^3 + 4*x^2 - 10;
fp = @(x) 3*x^2 + 8*x;
p = p0 - f(p0)/fp(p0);
z = [p]; i = 1;
while (abs(p - p0) ≥ tol)
    p0 = p;
    p = p0 - f(p0)/fp(p0);
    z = [z,p];
    i = i + 1;
    if (i ≥ Nmax)
        fprintf('Fail after %d iterations\n',Nmax);
        break
    end
end
end
```

MatLab Newton Example 2

The previous code generates Newton iterates to solve

$$f(x) = x^3 + 4x^2 - 10 = 0$$

with the first **five** iterates being:

$$z = [1.454545454545455, \quad 1.368900401069519, \quad 1.365236600202116, \quad 1.365230013435367, \quad 1.365230013414097]$$

Quadratic convergence suggests examining:

$$Q_n = \frac{|z_{n+1} - z_n|}{|z_n - z_{n-1}|^2}.$$

Substituting the sequence $\{z_n\}$ into the fraction above gives:

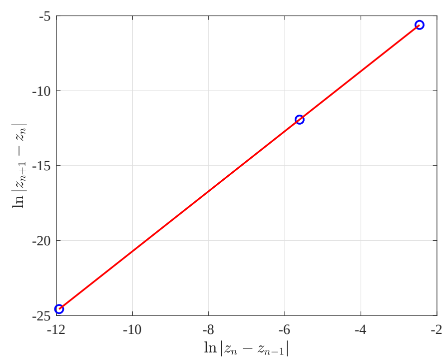
$$Q_2 = 0.49949, \quad Q_3 = 0.49069, \quad Q_4 = 0.49025.$$

MatLab Newton Example 3

```
xlab = '$\ln|z_{n}-z_{n-1}|$';        % X-label
ylab = '$\ln|z_{n+1}-z_{n}|$';            % Y-label
mytitle = '';    % Title
z = newtoneger(1,1e-7,20);
%z = secanteger(1,2,1e-9,20);
N = length(z);
xx = log(abs(z(2:N-1)-z(1:N-2)));
yy = log(abs(z(3:N)-z(2:N-1)));
p = polyfit(xx,yy,1)
x1 = min(xx); x2 = max(xx);
y1 = p(1)*x1 + p(2);
y2 = p(1)*x2 + p(2);
plot(xx,yy,'bo','MarkerSize',7);
hold on
plot([x1,x2],[y1,y2],'r-','LineWidth',1.5);
grid
```


MatLab Newton Example 4

The previous program plots $Y_n = \ln|z_{n+1} - z_n|$ vs. $X_n = \ln|z_n - z_{n-1}|$ and finds the slope, which is the ***rate of convergence***.



The program gives the best fitting line:

$$Y_n = 2.0017\, X_n - 0.69493.$$

Summary: Newton's Method

Newton's Method solves $f(x) = 0$ very efficiently

- Converges ***quadratically*** to the solution
- Roughly doubles the digits with each iteration when close
- Simple algorithm: Zero crossing of tangent line

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Problems:

- Difficult to determine the range of initial conditions for which Newton's method converges
- Algorithm often fails to converge
- Problems if $f'(x) = 0$
- Computing the derivative can be "expensive"
- If the zero of $f(x)$ isn't simple, then convergence is ***linear***

Secant Method

Main weakness of **Newton's Method** is computing the derivative

- Computing the derivative can be difficult
- Derivative often needs many more arithmetic operations

One solution is to **Approximate the Derivative**
By definition,

$$f'(x_{n-1}) = \lim_{x \to x_{n-1}} \frac{f(x) - f(x_{n-1})}{x - x_{n-1}}$$

Take $x = x_{n-2}$, then an approximation is

$$f'(x_{n-1}) \approx \frac{f(x_{n-2}) - f(x_{n-1})}{x_{n-2} - x_{n-1}}$$

Secant Method

The approximation

$$f'(x_{n-1}) \approx \frac{f(x_{n-2}) - f(x_{n-1})}{x_{n-2} - x_{n-1}}$$

is inserted into Newton's method to give

Secant Method

The **Secant Method** is an **iterative scheme**, where given an x_{n-2} and x_{n-1}, we compute

$$x_n = x_{n-1} - \frac{f(x_{n-1})(x_{n-2} - x_{n-1})}{f(x_{n-2}) - f(x_{n-1})}$$

MatLab Secant Code

```
function xn = secant(x0,x1,tol,Nmax)
%SECANT METHOD: Enter f(x), x0, x1, tol, Nmax
f = @(x) x^3 + 4*x^2 - 10;
xn = x1 - f(x1)*(x1-x0)/(f(x1)-f(x0));
i = 1;
while (abs(xn - x1) ≥ tol)
    x0 = x1;
    x1 = xn;
    xn = x1 - f(x1)*(x1-x0)/(f(x1)-f(x0));
    i = i + 1;
    if (i ≥ Nmax)
        fprintf('Fail after %d iterations\n',Nmax);
        break
    end
end
end
```


Secant Method – Convergence

The **Secant method** satisfies:

$$f'(x_{n-1}) \approx \frac{f(x_{n-2}) - f(x_{n-1})}{x_{n-2} - x_{n-1}}$$

- **Algorithm** requires two initial starting points
- New iterate is the zero crossing of the ***secant line***
- Do **NOT** need a derivative, only **function** evaluations
- **Order of Convergence** is ***superlinear***
  - Order of Convergence has been shown to be the ***golden ratio***, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.6$
  - Faster than **Bisection method**, but slower than **Newton's method**

MatLab Secant Example 1

```
function z = secanteger(x0,x1,tol,Nmax)
%SECANT METHOD: Enter f(x), x0, x1, tol, Nmax
f = @(x) x^3 + 4*x^2 - 10;
xn = x1 - f(x1)*(x1-x0)/(f(x1)-f(x0));
z=[xn]; i = 1;
while (abs(xn - x1) ≥ tol)
    x0 = x1;
    x1 = xn;
    xn = x1 - f(x1)*(x1-x0)/(f(x1)-f(x0));
    z=[z,xn];
    i = i + 1;
    if (i ≥ Nmax)
        fprintf('Fail after %d iterations\n',Nmax);
        break
    end
end
end
```


MatLab Secant Example 2

The previous code generates Secant iterates to solve

$$f(x) = x^3 + 4x^2 - 10 = 0$$

with the first **seven** iterates being:

$$z = [1.263157894736842, \quad 1.338827838827839, \quad 1.366616394719345, \quad 1.365211902631857, \quad 1.365230001110859, \quad 1.365230013414206 \quad 1.365230013414097]$$

Superlinear convergence with $\alpha = \frac{1+\sqrt{5}}{2}$ suggests examining:

$$S_n = \frac{|z_{n+1} - z_n|}{|z_n - z_{n-1}|^{\alpha}}.$$

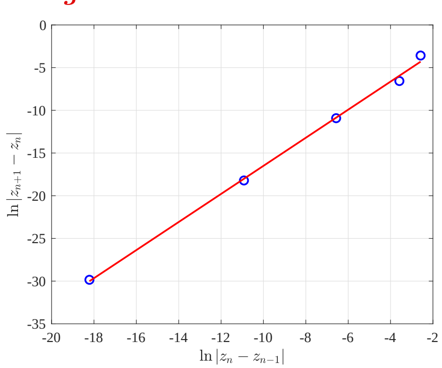
Substituting the sequence $\{z_n\}$ into the fraction above gives:

$$S_2 = 1.8105, \quad S_3 = 0.4628, \quad S_4 = 0.7465, \quad S_5 = 0.5799, \quad S_6 = 0.6871,$$

which is roughly constant.

MatLab Secant Example 3

The program with the Secant method on Slide 30 plots $Y_n = \ln|z_{n+1} - z_n|$ vs. $X_n = \ln|z_n - z_{n-1}|$ and finds the slope, which is the ***rate of convergence***.



The program gives the best fitting line:

$$Y_n = 1.6444\, X_n - 0.057042.$$

Root Finding Methods

The **Bisection method**
- **Very stable Algorithm -** Good technique to find starting point for Newton's method
- Costs only one function evaluation, so rapid iterations
- ***Linear*** convergence, so slow (3.3 iterations/digit)

The **Secant method**
- Hard to find starting points (Unknown **basin of attraction**)
- Costs only two function evaluations, so rapid iterations
- ***Superlinear*** convergence, $\alpha \approx 1.62$, which is pretty fast

The **Newton's method**
- Hard to find starting points (Unknown **basin of attraction**)
- Finding and evaluating derivative requires more machine work at each iteration
- ***Quadratic*** convergence is very fast – doubling the digits at each iteration

Example

Return to Example:

$$f(x) = x^3 + 4x^2 - 10$$

We know the root is between $x = 1$ and $x = 1.5$. (Use for **Bisection** and **Secant** methods)

n	Bisection	Secant	Newton
1	1.25	1.33898305084745	1.45454545454545
2	1.375	1.36356284991687	1.36890040106951
3	1.3125	1.36525168742565	1.36523660020211
4	1.34375	1.36522999568865	1.36523001343536
5	1.359375	1.36523001341391	1.36523001341409
6	1.3671875	1.36523001341409	
7	1.36328125		
8	1.365234375		
9	1.3642578125		
10	1.36474609375		
11	1.364990234375		
12	1.3651123046875		

Modifying Newton's Method

The local nature of **Newton's method** means that we are stuck with problems of finding the ***basin of attraction*** for the root, p^*, where $f(p^*) = 0$

The **Bisection method** is a stable routine often used to narrow the search of p_0, so that **Newton's method** converges

Another major problem is that **Newton's method** breaks when $f'(p^*) = 0$ (division by zero).

The good news is that this problem can be fixed!

— We need a short discussion on the ***multiplicity of zeroes***.

Multiplicity of Zeroes 1 of 2

Definition (Multiplicity of a Root)

A solution p^* of $f(x) = 0$ is a **zero of multiplicity** m of f if for $x \neq p^*$ we can write

$$f(x) = (x - p^*)^m q(x), \quad \lim_{x \to p^*} q(x) \neq 0$$

Basically, $q(x)$ is the part of $f(x)$ which does not contribute to the zero of $f(x)$

If $m = 1$ then we say that $f(x)$ has a ***simple zero***.

Theorem

$f \in C^1[a, b]$ has a simple zero at p^* in (a, b) if and only if $f(p^*) = 0$, but $f'(p^*) \neq 0$.

Multiplicity of Zeroes 2 of 2

Theorem (Multiplicity and Derivatives)

The function $f \in C^m[a, b]$ has a zero of multiplicity m at p^* in (a, b) if and only if

$$0 = f(p^*) = f'(p^*) = \cdots f^{(m-1)}(p^*), \quad \text{but } f^{(m)}(p^*) \neq 0.$$

We know that Newton's method runs into trouble when we have a zero of multiplicity higher than 1

Newton's method only converges ***linearly*** in these cases

Halley's method is a modification that converges ***cubically***, in general, and ***quadratically*** for higher order roots

Halley's Method for Zeroes of Higher Multiplicity

Halley's Method (for Zeroes of Multiplicity ≥ 2)

$$x_{n+1} = x_n - \frac{f(x_n)f'(x_n)}{[f'(x_n)]^2 - f(x_n)f''(x_n)}$$

Drawbacks:

We have to compute $f''(x)$ — more expensive and possibly another source of numerical and/or measurement errors.

We have to compute a more complicated expression in each iteration — more expensive.

Roundoff errors in the denominator — both $f'(x)$ and $f(x)$ approach zero, so we are computing the difference between two small numbers; a serious cancellation risk.