

Lecture 04 - C

§3.3 Secant Method

T. Takahashi

Department of Mathematics and Statistics
University of New Mexico



We will answer the following questions:

- What is the secant method?
- How fast does the secant method converge?
- What are the relative advantages and disadvantages of the bisection, Newton, and secant methods?



Secant Method



Drawback to Newton's Method

Newton Formula

$$x_{k+1} = x_k - \underbrace{\frac{f(x_k)}{f'(x_k)}}_{\text{need derivative}}$$

Pro:

- Very fast convergence near a simple root ($r = 2$)

Con:

- Requires evaluating the derivative $f'(x)$
- Derivatives can be difficult or expensive to compute
- In higher dimensions, $f'(x)$ becomes the Jacobian matrix

Question

Can we approximate the derivative instead?

Approximating the Derivative

- Recall the definition of the derivative:

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

- This ratio is called the **difference quotient** (DQ)

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

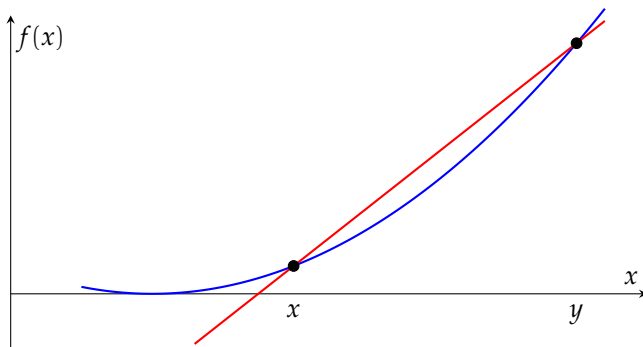
- Here is an alternate equivalent DQ using two points x and y ,

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



Difference Quotient

Let $y \rightarrow x$

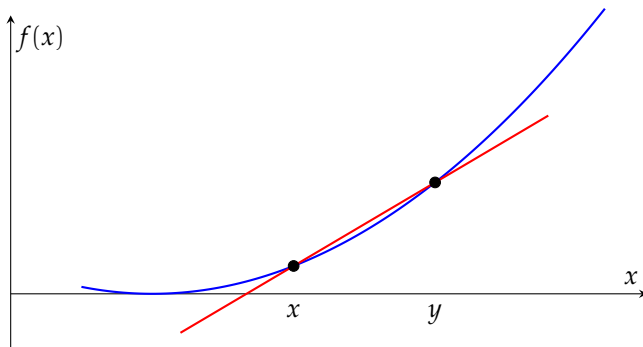


$$\frac{f(x) - f(y)}{x - y} = \text{slope of the secant line}$$



Difference Quotient

Let $y \rightarrow x$

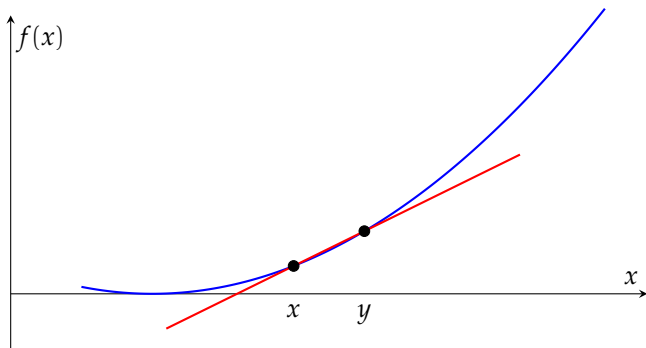


$$\frac{f(x) - f(y)}{x - y} = \text{slope of the secant line}$$



Difference Quotient

Let $y \rightarrow x$

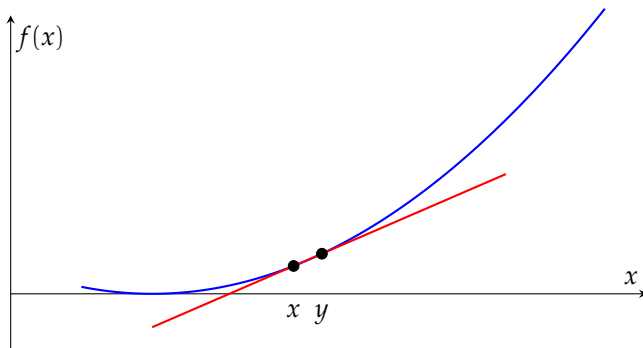


$$\frac{f(x) - f(y)}{x - y} = \text{slope of the secant line}$$



Difference Quotient

Let $y \rightarrow x$

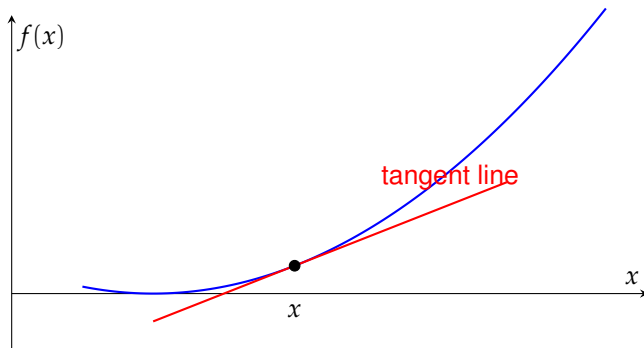


$$\frac{f(x) - f(y)}{x - y} = \text{slope of the secant line}$$



Difference Quotient

Let $y \rightarrow x$



$$\lim_{y \rightarrow x} \frac{f(x) - f(y)}{x - y} = f'(x)$$



Iterations

1. Suppose you have a sequence x_k that converges to x^*

$$x_k, x_{k+1}, x_{k+2}, \dots, x^*$$

2. Suppose you keep $f(x_{k-1})$ in memory as you compute $f(x_k)$

$$\underbrace{(x_{k-1}, f(x_{k-1}))}_{\text{saved}} \quad \text{and} \quad \underbrace{(x_k, f(x_k))}_{\text{newly computed}}.$$

- You will always have access to a derivative approximation

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}.$$

Key Idea

As long as you save x_{k-1} and $f(x_{k-1})$, approximating $f'(x_k)$ is nearly free.

Secant Method

1. Save the values of x_{k-1} and $f(x_{k-1})$
2. Instead of doing the Newton iteration

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)},$$

replace $f'(x_k)$ with the approximation

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

Secant Method Formula

Begin with two initial guesses x_0, x_1 . Update the guess with the recurrence equation:

$$x_{k+1} = x_k - f(x_k) \left[\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \right]$$

Secant Method Algorithm

Method: Secant Method

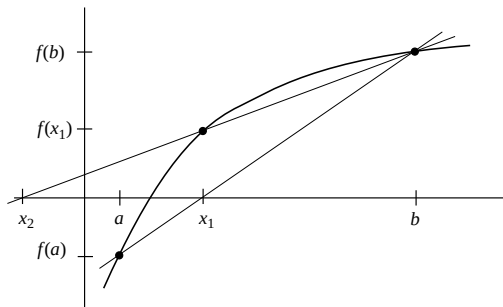
- **Given:** $f(x)$, two initial guesses x_0, x_1
- **Output:** Approximate solution to $f(x) = 0$
- **Idea:** Similar to Newton but the derivative is not required.

Algorithm:

```
1 initialize:  $x_0, x_1$ 
2 for  $k = 1, 2, 3 \dots$ 
3      $x_{k+1} = x_k - f(x_k)(x_k - x_{k-1}) / (f(x_k) - f(x_{k-1}))$ 
4     if converged, stop
5 end
```



Geometric Interpretation



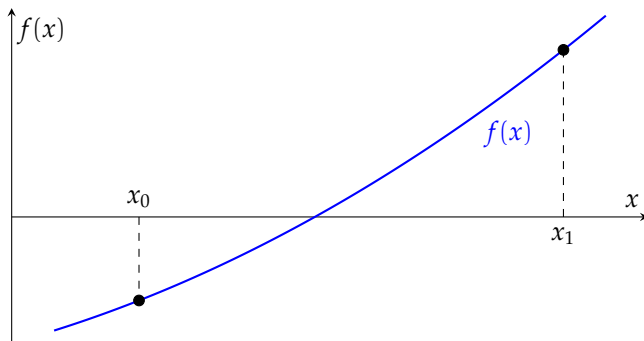
There is a geometric interpretation to secant method

- Draw a secant line
- Find the root of the secant line
- Repeat



Secant Method

Geometric Interpretation

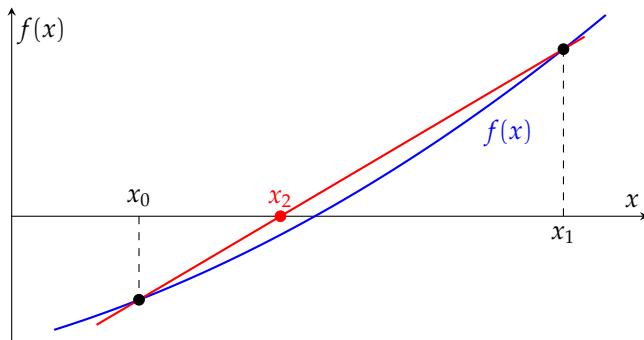


1. Start with a bracket $[x_0, x_1]$



Secant Method

Geometric Interpretation

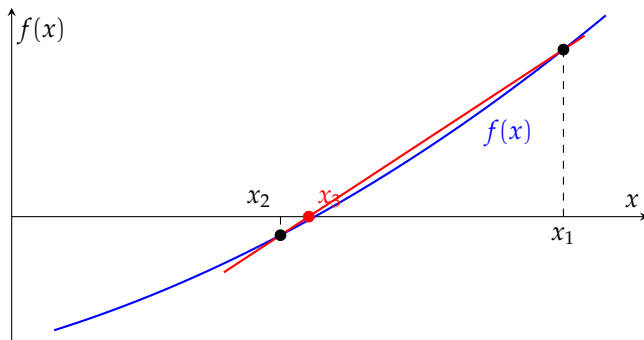


2. Draw a line through $(x_0, f(x_0))$ and $(x_1, f(x_1))$.
3. Find the root of the secant line.



Secant Method

Geometric Interpretation



4. Repeat with new pair $(x_1, f(x_1))$ and $(x_2, f(x_2))$.



Example

Do one iteration of Newton's Method and the Secant method for the function

$$f(x) = x^2 - 3x + 1.$$

Use $x_0 = 1$ for Newton, and use $x_0 = 1, x_1 = 5$ for Secant.

Newton:

- First find the derivative: $f'(x) = 2x - 3$
- One iteration

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 1 - \frac{-1}{-5} = \frac{6}{5}.$$

Secant

- One iteration

$$x_2 = x_1 - f(x_0) \frac{x_1 - x_0}{f(x_1) - f(x_0)} = 1 - (-1) \frac{5 - 1}{11 - 1} = 1 + \frac{1}{3} = \frac{4}{3}.$$



Error Analysis



Convergence Rate

Goal

Determine the convergence order r satisfying

$$|e_{k+1}| \approx C|e_k|^r.$$

- We will use the following theorem

Theorem

The error $e_k = x_* - x_k$ for secant method satisfies

$$e_{k+1} = -\frac{1}{2} \left(\frac{f''(\xi_n)}{f'(\zeta_n)} \right) e_k e_{k-1}$$

We will not prove this. See book NMC7 pg. 144

- We can simplify the theorem

$$|e_{k+1}| = K|e_k||e_{k-1}|$$



Finding the Convergence Order

Part 1

1. Suppose the Secant Method has convergence order r :

$$|e_{k+1}| \approx C|e_k|^r.$$

2. Then one iteration earlier,

$$|e_k| \approx C|e_{k-1}|^r.$$

3. Solving for the older error gives

$$|e_{k-1}| \approx \left(\frac{|e_k|}{C} \right)^{1/r}.$$



Finding the Convergence Order

Part 1

1. Suppose the Secant Method has convergence order r :

$$|e_{k+1}| \approx C|e_k|^r.$$

2. Then one iteration earlier,

$$|e_k| \approx C|e_{k-1}|^r.$$

3. Solving for the older error gives

$$|e_{k-1}| \approx \left(\frac{|e_k|}{C}\right)^{1/r}.$$

4. Substitute old error into the theorem equation

$$|e_{k+1}| = K|e_k||e_{k-1}|$$

to obtain

$$|e_{k+1}| \approx KC^{-1/r}|e_k|^{1+1/r}.$$



Finding the Convergence Order

Part 2

5. Compare the two expressions for the new error:

$$|e_{k+1}| \approx C|e_k|^r \quad \text{vs.} \quad |e_{k+1}| \approx KC^{-1/r}|e_k|^{1+1/r}.$$

For these to be consistent, the powers of $|e_k|$ must agree:

$$r = 1 + \frac{1}{r}.$$



Finding the Convergence Order

Part 2

5. Compare the two expressions for the new error:

$$|e_{k+1}| \approx C|e_k|^r \quad \text{vs.} \quad |e_{k+1}| \approx KC^{-1/r}|e_k|^{1+1/r}.$$

For these to be consistent, the powers of $|e_k|$ must agree:

$$r = 1 + \frac{1}{r}.$$

6. Multiply by r and rearrange:

$$r^2 - r - 1 = 0.$$



Finding the Convergence Order

Part 2

5. Compare the two expressions for the new error:

$$|e_{k+1}| \approx C|e_k|^r \quad \text{vs.} \quad |e_{k+1}| \approx KC^{-1/r}|e_k|^{1+1/r}.$$

For these to be consistent, the powers of $|e_k|$ must agree:

$$r = 1 + \frac{1}{r}.$$

6. Multiply by r and rearrange:

$$r^2 - r - 1 = 0.$$

7. Solve using the quadratic formula to obtain the convergence rate

$$r = \frac{1 + \sqrt{5}}{2} \approx 1.618$$

where we discard the negative root.



Superlinear Convergence

- This quantity is called the **golden ratio**

$$r = \frac{1 + \sqrt{5}}{2} \approx 1.618$$

- Convergence rate for secant is *between* linear and quadratic

$$\underbrace{1}_{\text{linear}} < 1.618 < \underbrace{2}_{\text{quadratic}}$$

- We call this **superlinear** convergence

Key Idea

Secant is faster than bisection but slower than Newton.



Experimental Approach

Make a Table

- 1 Calculate a sequence of errors $e_0, e_1, \dots, e_k, \dots$
- 2 Calculate a table of error ratios with k and e^{k+1}/e_k^r
- 3 Numbers should be roughly constant

Formula

- Assume $e_{k+1} \approx Ce_k^r$
- Then $e_k \approx Ce_{k-1}^r$
- Divide

$$\frac{e_{k+1}}{e_k} \approx \left(\frac{e_k}{e_{k-1}} \right)^r$$

- Solve for r



Comparison of Rootfinding Methods



Revisit Example

Problem

Consider finding the root of

$$f(x) = x - x^{1/3} - 2 = 0$$

using bisection, Newton's method, and Secant method. How many iterations does each one require to get machine precision?



Revisit Example

Bisection

Bisection:

k	a	b	x_{mid}	$f(x_{mid})$
1	3.00000000	4.00000000	3.50000000	-0.01829449
2	3.50000000	4.00000000	3.75000000	0.19638375
3	3.50000000	3.75000000	3.62500000	0.08884159
4	3.50000000	3.62500000	3.56250000	0.03522131
5	3.50000000	3.56250000	3.53125000	0.00845016
6	3.50000000	3.53125000	3.51562500	-0.00492550
7	3.51562500	3.53125000	3.52343750	0.00176150
8	3.51562500	3.52343750	3.51953125	-0.00158221
9	3.51953125	3.52343750	3.52148438	0.00008959
10	3.51953125	3.52148438	3.52050781	-0.00074632

⋮

At iteration 51, $\text{fl}(f(x_{mid})) = 0$.



Revisit Example

Newton and Secant

Newton

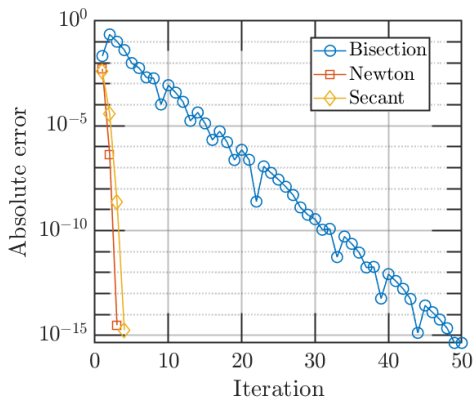
k	x_k	$f'(x_k)$	$f(x)$
0	3	0.83975005	-0.44224957
1	3.52664429	0.85612976	0.00450679
2	3.52138015	0.85598641	3.771×10^{-7}
3	3.52137971	0.85598640	2.664×10^{-15}
4	3.52137971	0.85598640	0.0

Secant

k	x_{k-1}	x_k	$f(x_k)$
0	4	3	-0.44224957
1	3	3.51734262	-0.00345547
2	3.51734262	3.52141665	0.00003163
3	3.52141665	3.52137970	-2.034×10^{-9}
4	3.52137959	3.52137971	-1.332×10^{-15}
5	3.52137971	3.52137971	0.0



Comparison



- Bisection requires many more iterations
- Newton and Secant reach machine precision very quickly

Question

If Bisection is so much slower, why would we ever use it?

Comparison of Rootfinding Methods

	Bisection	Newton	Secant
Convergence	Linear	Quadratic	Superlinear
Rate r	1	2	≈ 1.618
Derivative	No	Yes	No
Initial values	2	1	2
Guaranteed	Yes*	No	No

*Assuming f is continuous and the initial interval brackets a root.

Key Idea

There is no best rootfinding method. Each has different advantages and disadvantages.

Extra Topic: Modern Root-finding



Two Ideas

Idea 1

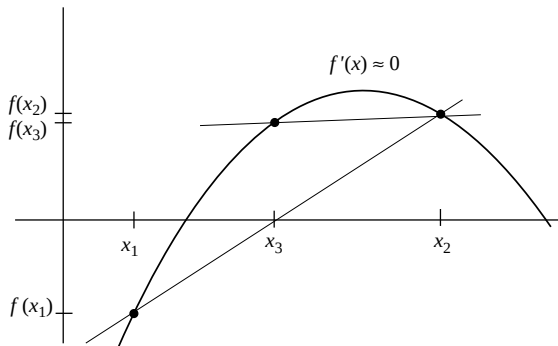
Make an algorithm that uses a combination of bisection, secant, and Newton's method so that it is both fast and reliable. - **Brent's method**

Idea 2

Make an algorithm that is able to find complex roots and handle roots with multiple zeros. - **companion matrix method**



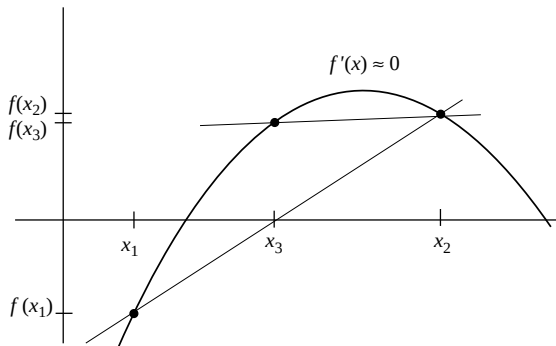
When Secant Can Fail



- Recall the Secant update:

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

When Secant Can Fail



- Recall the Secant update:

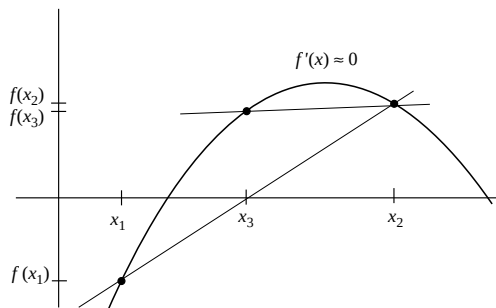
$$x_{k+1} = x_k - f(x_k) \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}.$$

- Problem when

$$\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}} \approx 0,$$



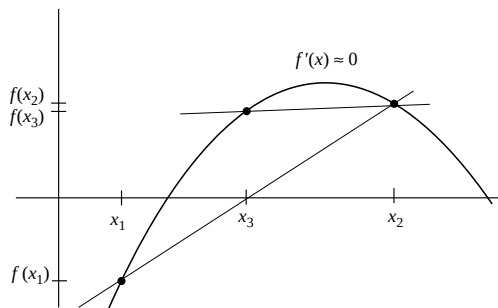
Can We Make Secant Safer?



- Bisection always preserves a bracket

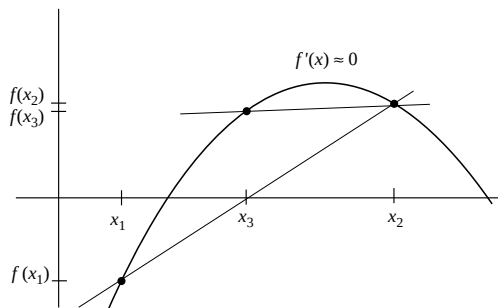


Can We Make Secant Safer?



- Bisection always preserves a bracket
- Secant can leave the bracket and potentially diverge

Can We Make Secant Safer?



- Bisection always preserves a bracket
- Secant can leave the bracket and potentially diverge

Question

Can we get the speed of Secant with the reliability of Bisection?

Idea 1: fzero Function

Brent's Method

Brent's Method uses a hybrid rootfinding algorithm that combines **bisection**, **secant**, and **inverse quadratic interpolation**.

In Matlab, use the built-in function `fzero()`

```
1 r = fzero(fun,x0)
2 r = fzero(fun,x0,options)
```

`x0` can be a scalar or a two-element vector:

- If `x0` is a scalar, `fzero` searches for a bracket containing a sign change.
- If `x0` is a two-element vector, it must bracket a sign change.

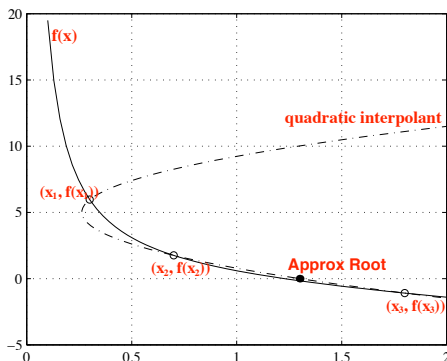


Inverse Quadratic Interpolation

- Start with 3 points

$$(x_0, f(x_0)), \quad (x_1, f(x_1)), \quad (x_2, f(x_2)).$$

- Approximate f with a **sideways parabola**
- New point x_3 is the root of the parabola



fzero: A Hybrid Method

At each iteration, fzero can use:

- **Inverse quadratic interpolation** when three suitable points are available.



fzero: A Hybrid Method

At each iteration, fzero can use:

- **Inverse quadratic interpolation** when three suitable points are available.
- **Secant interpolation** when two suitable points are available.



fzero: A Hybrid Method

At each iteration, fzero can use:

- **Inverse quadratic interpolation** when three suitable points are available.
- **Secant interpolation** when two suitable points are available.
- **Bisection** when an interpolation step is not considered safe.



fzero: A Hybrid Method

At each iteration, fzero can use:

- **Inverse quadratic interpolation** when three suitable points are available.
- **Secant interpolation** when two suitable points are available.
- **Bisection** when an interpolation step is not considered safe.

Key Idea

Use fast interpolation when possible, but use Bisection to safeguard the search.



fzero Function

Command `fzero` finds the root closest to the initial point

```
>> x = fzero(@cos,0.5)
x =
    1.5708
```

Use `optimset` to create settings

```
>> options = optimset('Display','iter');
>> x = fzero(@cos, 0.5, options)
```

Choose the error tolerance

```
>> options = optimset('TolX',5e-9);
>> x = fzero(@cos, 0.5, options)
```



$$f(x) = x - x^{1/3} - 2 = 0$$

```
1 f = @(x) x - x.^(1/3) - 2;
2 options = optimset('Display','iter');
3 r = fzero(f,3,options)
```

Search for an interval around 3 containing a sign change:

Func-count	a	f(a)	b	f(b)	Procedure
1	3	-0.44225	3	-0.44225	initial interval
3	2.91515	-0.513374	3.08485	-0.370868	search
5	2.88	-0.542757	3.12	-0.341229	search
7	2.83029	-0.58423	3.16971	-0.299242	search
9	2.76	-0.642716	3.24	-0.239727	search
11	2.66059	-0.725079	3.33941	-0.155298	search
13	2.52	-0.840818	3.48	-0.035397	search
15	2.32118	-1.00287	3.67882	0.135099	search

Search for a zero in the interval [2.32118, 3.67882]:

Func-count	x	f(x)	Procedure
15	3.67882	0.135099	initial
16	3.51764	-0.00319777	interpolation
17	3.52137	-7.81109e-06	interpolation
18	3.52138	1.34328e-11	interpolation
19	3.52138	0	interpolation

Zero found in the interval [2.32118, 3.67882]

r = 3.5214



Another fzero example

Question

How well does `fzero` handle multiple zeros?

- Consider the polynomial

$$f(x) = x^{10} - 1$$

- Solution: $x = \pm 1$

```
1 >> f = @(x)x.^10 - 1;  
2 >> options = optimset('display','iter');  
3 >> [x,fx]=fzero(f,0.5,options)
```

- It take 35 iterations!



Idea 2: roots

Polynomial Rootfinding

The MATLAB function `roots` is specifically designed to find roots of polynomials.

- No initial guess
- Returns all roots of the polynomial
- Can find real, repeated, and complex roots

Write polynomial in the form

$$c_1x^n + c_2x^{n-1} + \dots + c_nx + c_{n+1} = 0$$

For example, for a cubic polynomial:

```
1 >> c = [c1 c2 c3 c4];  
2 >> r = roots(c)
```

roots - How it works

- Consider the matrix

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -c_0 & -c_1 & -c_2 & -c_3 \end{bmatrix}$$

- Find the eigenvalues by setting the determinant of $A - \lambda I$ to zero

$$A - \lambda I = \begin{bmatrix} -\lambda & 1 & 0 & 0 \\ 0 & -\lambda & 1 & 0 \\ 0 & 0 & -\lambda & 1 \\ -c_0 & -c_1 & -c_2 & (-c_3 - \lambda) \end{bmatrix}$$

- But that is a polynomial

$$\det(A - \lambda I) = p(\lambda) = \lambda^4 + c_3\lambda^3 + c_2\lambda^2 + c_1\lambda + c_0$$

- Use eigenvalue solvers to find roots of polynomials!



roots - Examples

Come examples

$$f_1(x) = x^2 - 3x + 2 \quad \text{two real roots}$$

$$f_2(x) = x^2 - 10x + 25 \quad \text{repeated roots}$$

$$f_3(x) = x^2 - 17x + 72.5 \quad \text{complex roots}$$

```
1 >> roots([1 -3 2])
2 ans =
3      2
4      1
5
6 >> roots([1 -10 25])
7 ans =
8      5
9      5
10
11 >> roots([1 -17 72.5])
12 ans =
13  8.5000 + 0.5000i
14  8.5000 - 0.5000i
```



Degree 10 Polynomial

$$f(x) = x^{10} - 1 = 0$$

```
1 >> c = [1, zeros(1,9), -1];  
2  
3 >> roots(c) % finds all 10 roots  
4 ans =  
5 -1.0000 + 0.0000i % x=-1 found  
6 -0.8090 + 0.5878i  
7 -0.8090 - 0.5878i  
8 -0.3090 + 0.9511i  
9 -0.3090 - 0.9511i  
10 0.3090 + 0.9511i  
11 0.3090 - 0.9511i  
12 1.0000 + 0.0000i % x=1 found  
13 0.8090 + 0.5878i  
14 0.8090 - 0.5878i
```