Introduction to Scientific Computing

Polynomials

Curve Fitting

Interpolation

Polynomials

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

- **MATLAB representation is a vector**

$$8x+5 \quad \rightarrow \quad \texttt{p = [8 5]}$$

$$2x^2 - 4x + 10 \quad \rightarrow \quad \texttt{p = [2 -4 10]}$$

$$6x^2 - 150 \quad \rightarrow \quad \texttt{p = [6 0 -150]}$$

$$5x^5 + 6x^2 - 7x \quad \rightarrow \quad \texttt{p = [5 0 0 6 -7 0]}$$

- **Value of a polynomial**

```
v = polyval(p, x)
```

`p` = vector of coefficients

`x` = point to evaluate (scalar, vector, matrix)

$$f(x) = x^5 - 12.1x^4 + 40.59x^3 - 17.015x^2 - 71.95x + 35.88$$

Polynomial Functions

- **Roots of a polynomial**

```
r = roots(p)
```

- **Polynomial from its roots**

```
p = poly(r)
```

- **Polynomial multiplication (convolution)**

```
c = conv(a,b)
```

- **Polynomial division (deconvolution)**

```
[q, r] = deconv(a,b)
```

- **Derivatives of polynomials**

```
pd = polyder(p)
```

Curve Fitting

- **If we can fit a function to a set of data points, the result is a mathematical model for the data.**

- **Many types of functions could be a fit: linear, polynomial, power, exponential, etc.**

- **Polynomial fitting**
  - Curve passes through all data points: with `n` points, a polynomial of degree `n-1` will pass through the points.
  - Curve may not pass through all points: polynomial of degree less than `n` that overall approximates the data.

Curve Fitting (cont.)

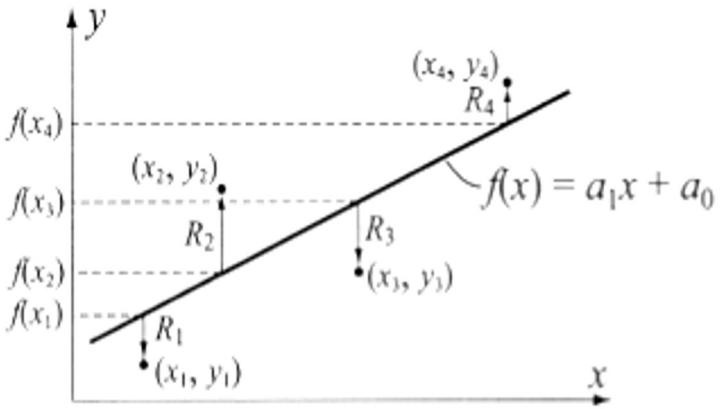
- **The Method of Least Squares (regression or curve fitting)**
  - Minimize sum of squares of residuals at all data points

```
p = polyfit(x, y, n)
```

`x, y`: vectors of data coordinates

`n`: degree of polynomial

`p`: vector of coefficients for polynomial fit



Theory?

Given: (x_k,y_k)

Find: f(x)=a+b*x

$J(a,b)=\text{Sum}([y_k-f(x_k)]^2)$ is minimum

Function Discovery

- **Find a function that describes a set of data**
  - – *Linear* function `y = mx + b` yields a straight line on normal axes.
  - – *Power* function $y = bx^m$ gives a straight line on `log- log` axes.
  - – *Exponential* function $y = b(10)^{mx}$ or $y = be^{mx}$ plots as a straight line on `semilog` axes where `y`-axis is logarithmic.
- **Use `polyfit` with modified arguments to find `m` and `b`.**

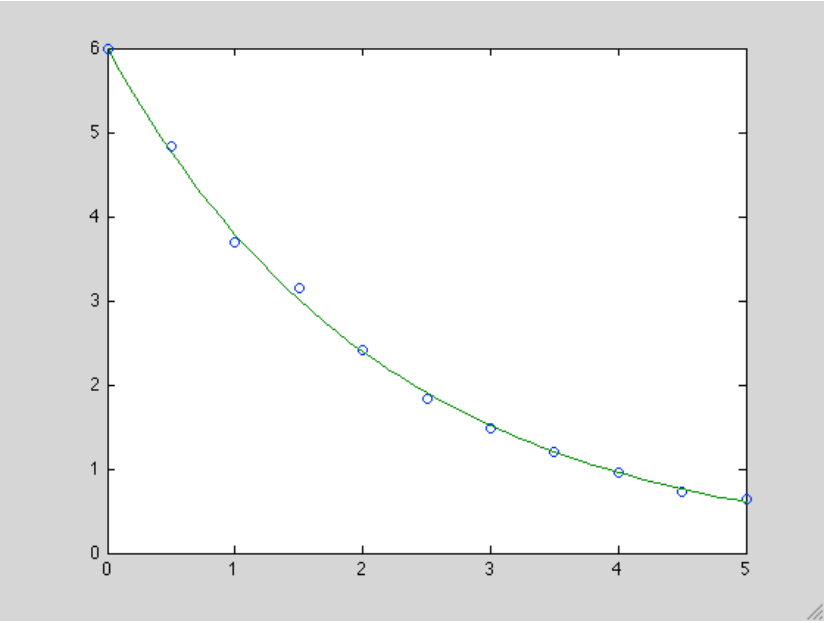
Fitting with Other than Polynomials

- **Rewrite functions in a form that can be fitted with a linear polynomial (`n=1`) such as** `y = mx + b`

Power	$y = bx^m$	`ln(y)= mln(x)+ln(b)`	`p=polyfit(log(x),log(y),1)`
Exponential	$y = be^{mx}$ $y = b10^{mx}$	`ln(y)= mx + ln(b)` `log(y)= mx + log(b)`	`p=polyfit(x,log(y),1)` `p=polyfit(x,log10(y),1)`
Logarithmic	`y=mln(x) +b` `Y=mlog(x)+b`		`p=polyfit(log(x),y,1)` `p=polyfit(log10(x),y,1)`
Reciprocal	`y=1/(mx+b)`	`1/y = mx + b`	`p=polyfit(x,1./y,1)`

Example - Fitting an Equation to Data Points

t	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
w	6.00	4.83	3.70	3.15	2.41	1.83	1.49	1.21	0.96	0.73	0.64



$$w = be^{mt}$$

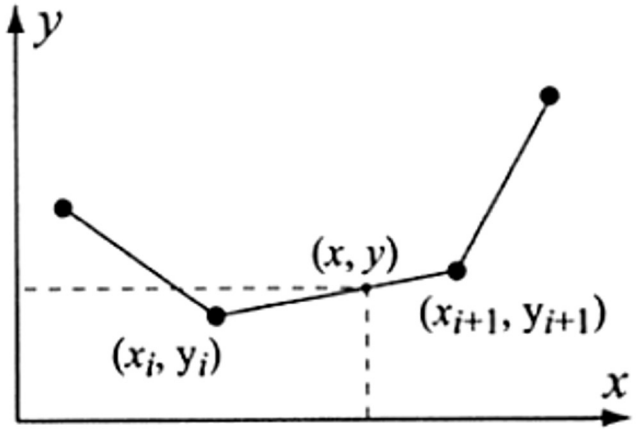
$$b = e^{1.7899} = 5.9889$$

$$m = -0.4580$$

Interpolation

- A polynomial fit to data allows us to estimate values between the data points.
- With more data points, a higher degree polynomial may be necessary and it may not do a good job with these approximations.
- Instead, consider a smaller number of points in the neighborhood of where the interpolated value is needed.
- This is called ***spline interpolation*** and is something MATLAB can do with a built-in function.

Interpolation (cont.)



$$y = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} x + \frac{y_i x_{i+1} - y_{i+1} x_i}{x_{i+1} - x_i}$$

Interpolation (cont.)

```
yi = interp1(x, y, xi, 'method')
```

`x, y`: vectors of data coordinates

`xi`: horizontal coordinate of interpolation point

`method`: method to use for interpolation

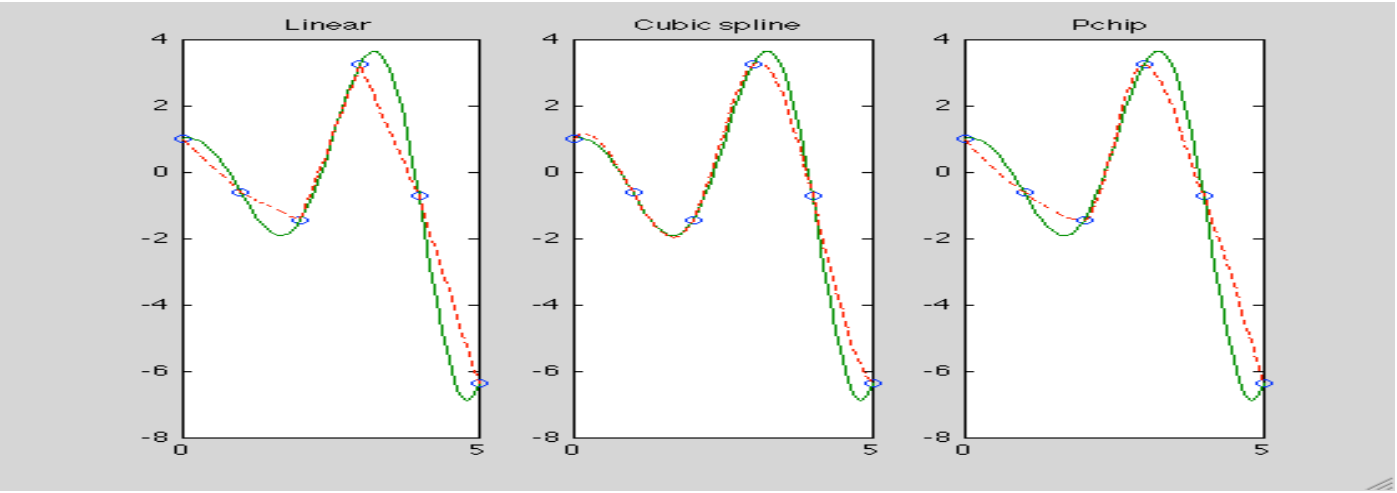
- `'nearest'` value of data point nearest
- `'linear'` linear spline
- `'spline'` cubic spline interpolation
- `'pchip'` piecewise cubic Hermite interpolation

`yi`: interpolated value

Example - Interpolation

$$f(x) = 1.5^x \cos(2x)$$

x	0	1	2	3	4	5
y	1.0	-0.6242	-1.4707	3.2406	-0.7366	-6.3717



Example Determining the Size of a Capacitor

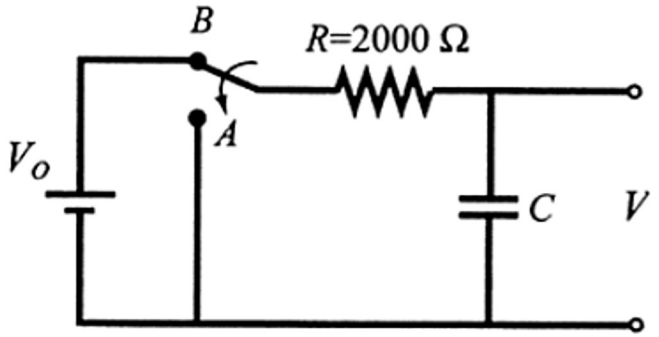
t	1	2	3	4	5	6	7	8	9	10
V	9.4	7.31	5.15	3.55	2.81	2.04	1.26	0.97	0.74	0.58

- Use measurements of *V* as a function of time to find the capacitance *C*.

$$V = V_0 e^{-t/RC}$$

$$\ln V = -\frac{1}{RC} t + \ln V_0$$

$$y = m \cdot x + b$$



- Use `polyfit` to find `m` and `b` and solve `C=-1/Rm`

Probability and Statistics

Data Statistics

- **Useful functions for**
  - understanding numerical datasets
  - predicting future outcomes based on the past
  - simulating processes

- **The basics**

```
avg = mean(x)
med = median(x)
```

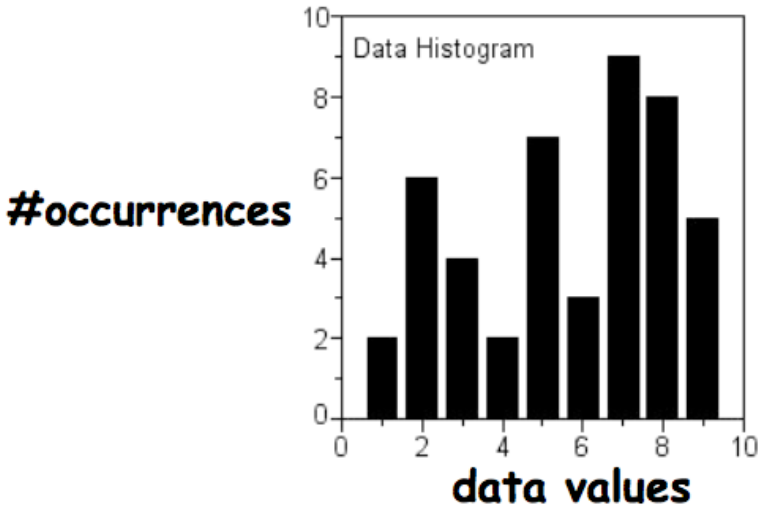
`x`: vector or matrix of data

`avg`: mean of values (scalar) or row vector of mean values of columns

`med`: median of values (scalar) or row vector of medians of columns

Histograms

- Mean and median sometimes do not adequately describe the data
- Histograms plot the frequency of occurrence of the data values not the data themselves



Histograms

- **Useful for showing the distribution of data**
- **Data is divided into bins (subranges)**
- **Histogram shows number of points in each bin**

`hist(y)` Data divided into 10 bins

`hist(y,n)` Data divided into n (scalar) bins

`hist(y,x)` Vector x specifies bin centers

- **Numerical output**

`n = hist(y)` Vector of number of points in each bin

Example Histograms

-Test 1

61	61	65	67	69	72	74	74	76	77
83	83	85	88	89	92	93	93	95	98

-Test 2

66	69	72	74	75	76	77	78	78	79
79	80	81	83	84	85	87	88	90	94

Range	Test1	Test2
60-69	5	2
70-79	5	9
80-89	5	7
90-99	5	2

Example 2 Breaking Strength of Thread

- Thread manufacturer samples and tests thread from the production line for breaking strength

- 20 samples are collected and the breaking force is measured in Newtons

92	94	93	96	93	94	95	96	91	93
95	95	95	92	93	94	91	94	92	93

Probability

- Probability describes the likelihood that something will happen; here, the chance that a particular value will occur in an experimental result
- Normally expressed as a number from 0 to 1 or a percentage from 1 to 100
- Obtaining theoretical probabilities can be hard or impossible
- Relative frequencies can be a good estimate of probability

Scaled Frequency Histogram

- **Height data in inches for 100 men 20 years old to nearest 1/2 inch**

height	64	64.5	65	65.5	66	66.5	67	67.5	68	68.5
Freq	1	0	0	0	2	4	5	4	8	11

height	69	69.5	70	70.5	71	71.5	72	72.5	73	73.5
Freq	12	10	9	8	7	5	4	4	3	1

height	74	74.5	75
Freq	1	0	1

Scaled Frequency Histogram (cont.)

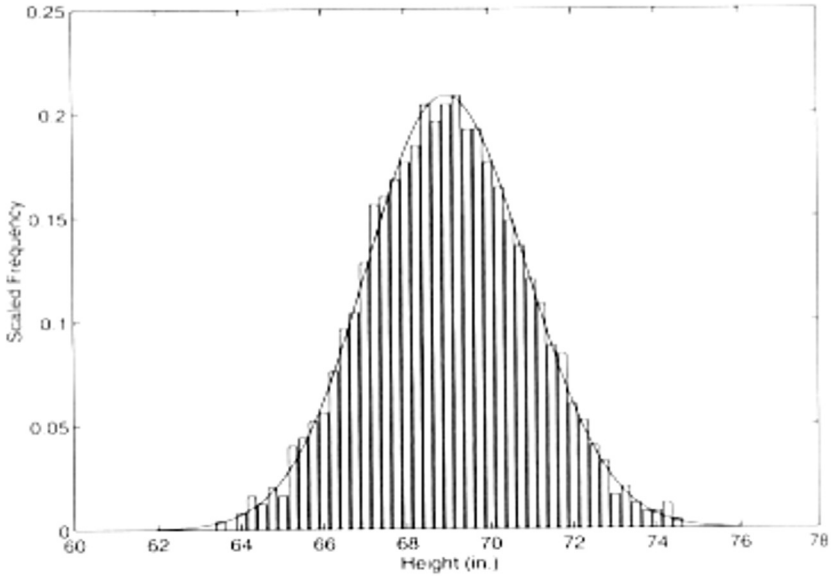
- A scaled frequency histogram is such that the total area under the rectangles is 1
- Fractional area for a range of heights gives the probability that a randomly selected man would have a height in that range
- From the plot, the total area of rectangles for heights 67 through 69 sums to

```
(0.1 + 0.08 + 0.16 + 0.22 + 0.24)(0.5) = 0.4
```

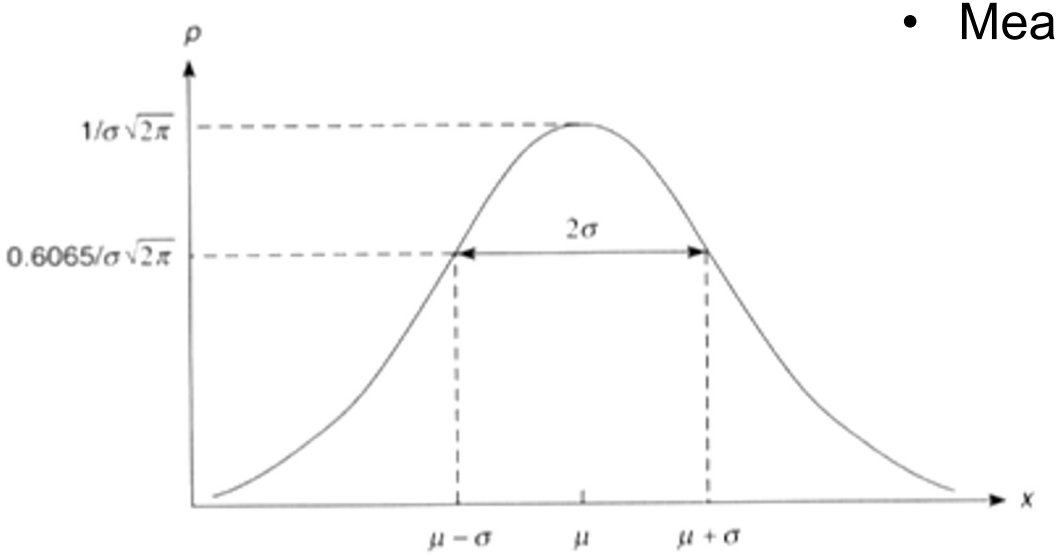
- Thus, 40% of the time the heights lie in this range

Continuous Approximations

- If we measured the height to greater and greater precision and increase the number of people measured, the scaled histogram approximates a smooth curve



Normal Distribution



- The "bell-shaped curve"
- Mean is μ and standard deviation is σ

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

Example 3 Height Distribution

- **Find the mean and standard deviation for the height data**

Function mean

$$\bar{x} = \frac{(x_1 + x_2 + ... + x_n)}{n}$$

Function std

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}}$$

Reconstruct raw height data from frequencies

Error Function

- **Find probability at other points on the curve**

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

`a = erf(x)`

`x`: **parameter**

`a`: **area to left of t = x under the curve**

$$\frac{2e^{-t^2}}{\sqrt{\pi}}$$

$$P(x \le b) = \frac{1}{2}\left[1 + erf\left(\frac{b-\mu}{\sigma\sqrt{2}}\right)\right]$$

$$P(a \le x \le b) = \frac{1}{2}\left[erf\left(\frac{b-\mu}{\sigma\sqrt{2}}\right) - erf\left(\frac{a-\mu}{\sigma\sqrt{2}}\right)\right]$$

Random Numbers

```
v = rand
rand(n)
rand(r, c)
s = rand('state')
rand('state', s)
rand('state', sum(100*clock))
```

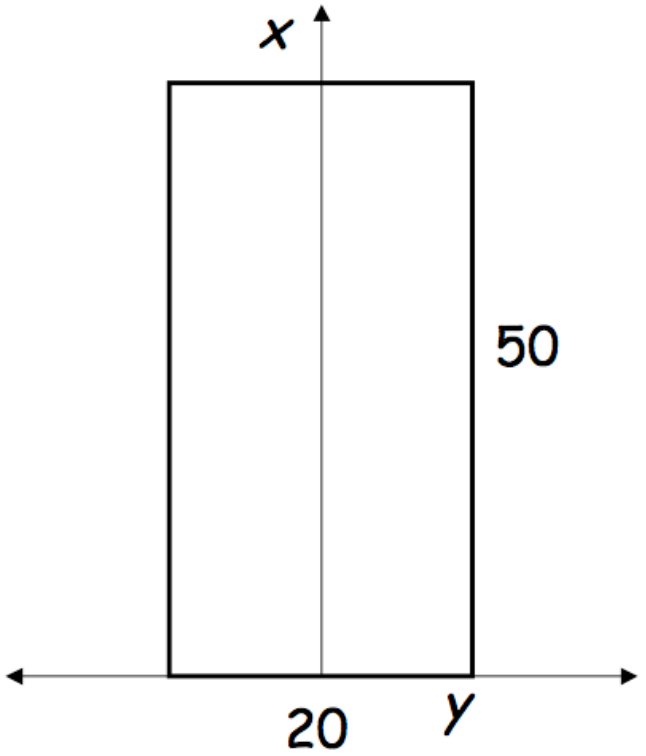
`v`: **random value**
`n`: **size of** `n x n` **matrix**
`r,c`: **size of** `r x c` **matrix**
`s`: **current state of random number generator**

Example 5 Sailor's Night Out

- A sailor returning to ship after a night on the town is trying to negotiate the pier and reach his ship
- The pier is 50 paces long and 20 wide; he starts at the middle on the end away from the ship
- At every step he has a 60% chance of stumbling blindly toward the ship and a 20% chance of lurching left or 20% right
- Simulate his progress and estimate his chances of reaching the ship

Example 5 (cont.) Sailor's Night Out

- **Data**
  - Represent the pier with coordinates running down the middle as shown



  - Keep track of
    - number of walks, `n`
    - number of steps, `steps`
    - position, `x, y`
    - successes, `nsafe`

Example 5 (cont.) Sailor's Night Out

- **Algorithm**

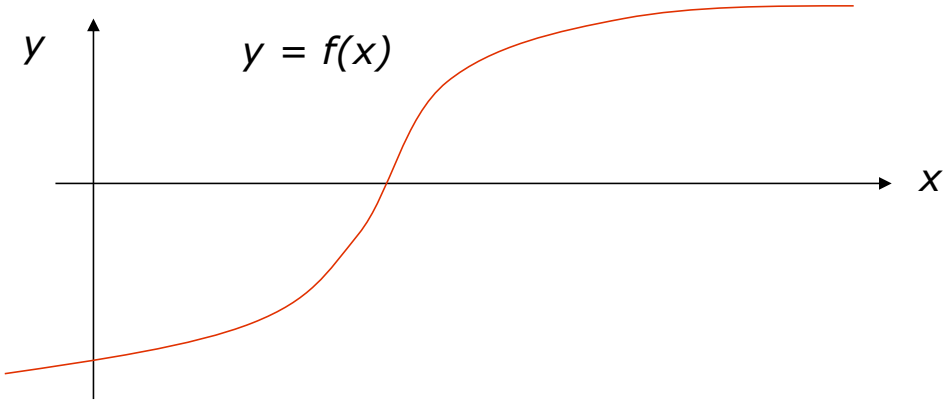
```
% Initialize variables
% Repeat n simulated walks down the pier
  % Start at the land end

  % While still on the pier and still alive
    % Get a random number r
    % If r<0.6 then move forward
    % Else if r<0.8 move left
    % Otherwise move right

  % If still alive count walk as a success

% Print estimated probability of reaching the ship
```

Root Finding & Optimization

Root Finding

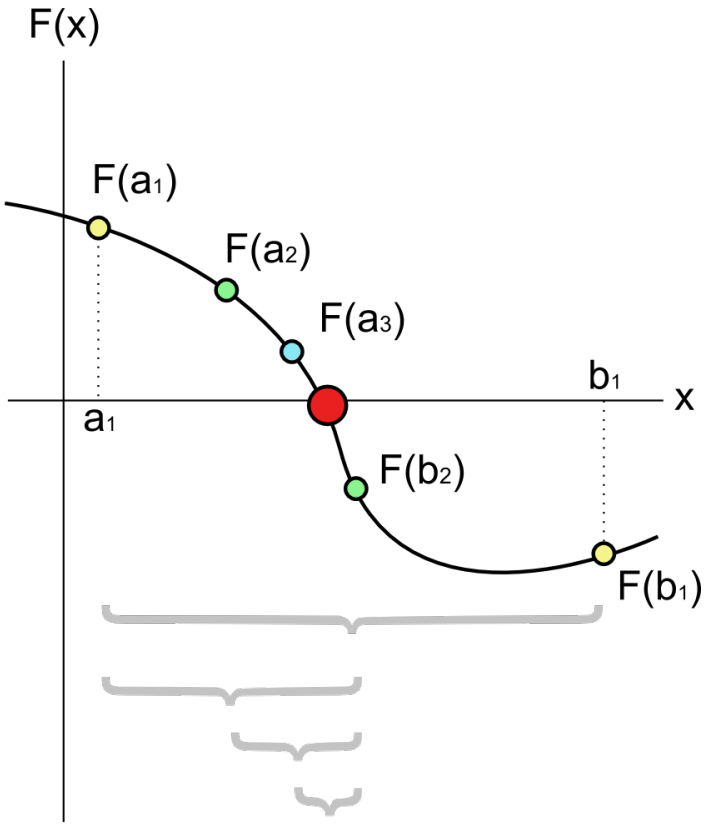
- **Finding the zero crossings of a function is a classical numerical problem with wide applications**

- **Given a scalar function `f(x)` our goal is to find values of `x` for which `f(x) = 0`**



Bisection Method

- **The bisection method starts with an interval that contains a zero crossing and repeatedly bisects that interval to narrow in on the zero crossing.**



`bisectionMethod.m`,
`L24.m`

Bisection method

·Bisection method pseudo code:

·Input

- A function f(x)
- Initial values a and b that bracket the zero crossing

while (abs(b – a) > tolerance)

- c = (a + b)/2; find the midpoint
- If (sign(f(c)) == sign(f(a))) ; update the interval

    a = c;

  Else

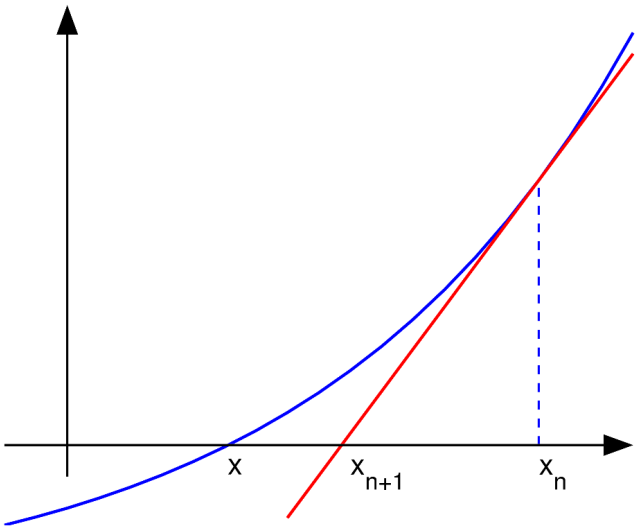
    b = c;

·Pros: Easy to implement / Does not require knowledge of the derivative of the function

·Cons: Slow to converge / Requires input of reasonable window that contains a root

Newton's Method

- Newton's method for root finding uses the derivative of the function. At each iteration we approximate the function using its derivative, predict where the zero will be and jump to that location.



$$f(x+\delta x) \approx f(x) + f'(x)\delta x$$

$$f(x+\delta x) = 0 \Rightarrow \delta x = -\frac{f(x)}{f'(x)}$$

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

`NewtonsMethod.m`

Newton's method

·Newton's method pseudo code:

·Input

- A function f(x) and it's derivative f'(x)
- Initial value, x0 that is close to a zero crossing

x = x0
while (abs(f(x)) > tolerance)

- x = x – (f(x) / f'(x)); update estimate for x

·Pros: Converges quickly / Only requires selection of an initial guess location

·Cons: Requires the function, f(x), and the derivative, f'(x) / Can diverge from the solution in some cases

Termination

- Note in all of these methods we need to tell our program when to stop. This is typically done by terminating when the absolute value of the function is below some tolerance, i.e. it is ‘almost zero’. This tolerance will depend upon the function you are working with.

  (for example 1.e-6)

Root Finding with `fzero`

```
x = fzero(fun, x0)
 fun: function to be solved
 x0:  initial guess at solution
 x:   the solution
```

- **Function can be specified as**
  - a mathematical expression as a string with no predefined variables
  - function handle
  - name of an anonymous function

- **Must be in the form** `f(x)=0`

```
f(x) = c   =>   f(x)-c = 0
```

Optimization Problems

- Generally phrased in terms of finding the optimal value of a particular parameter.

- You are typically provided with an objective function f(x) which returns a real number which indicates how good or bad a given choice for the parameter x is.

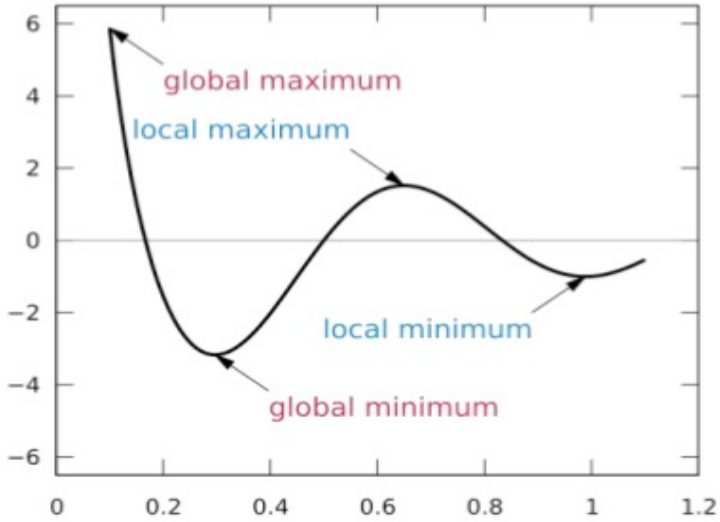
- Our job is to find the choice of x that maximizes, or minimizes, the given objective function.

- **Root Finding and Optimization**
  - Root Finding and optimization are intimately related. The maxima or minima of a function correspond to places where its derivative f'(x) is zero. So we are looking for zero crossings of f'(x).
  - We could apply Newton's method to f'(x) and f''(x) or apply a bisection method to f'(x)

Finding Minima and Maxima

- **Local minima or maxima occur when the derivative of the function is zero**

L24.m




```
x = fminbnd(function, x1, x2)

function: function to be solved
x1,x2: interval for minimum
x: function minimum
```

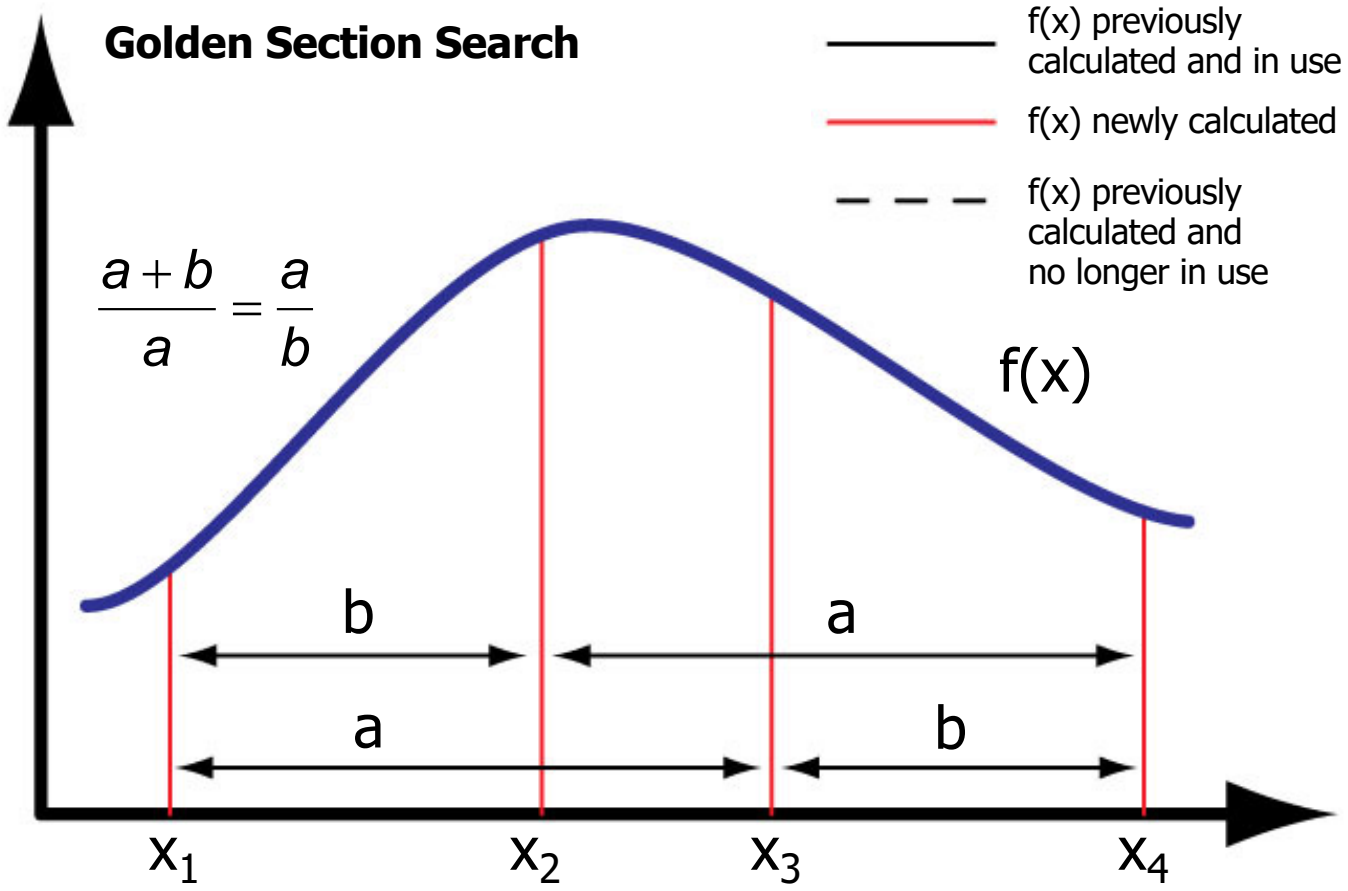
Golden section search

The Golden section search is based on the golden ratio

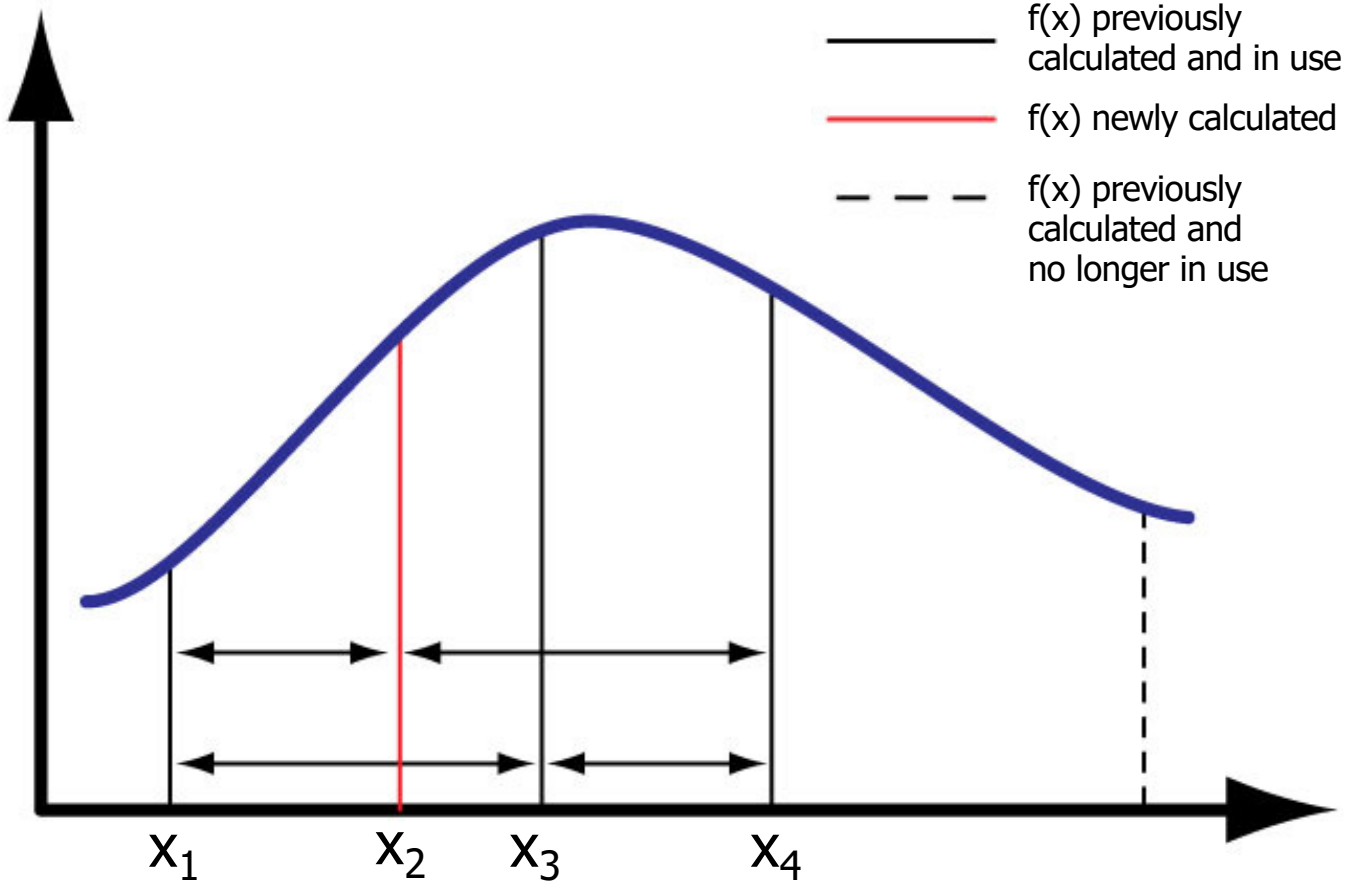
$$\frac{a+b}{a} = \frac{a}{b}$$

$$\varphi = \frac{1-\sqrt{5}}{2} \approx 1.618$$

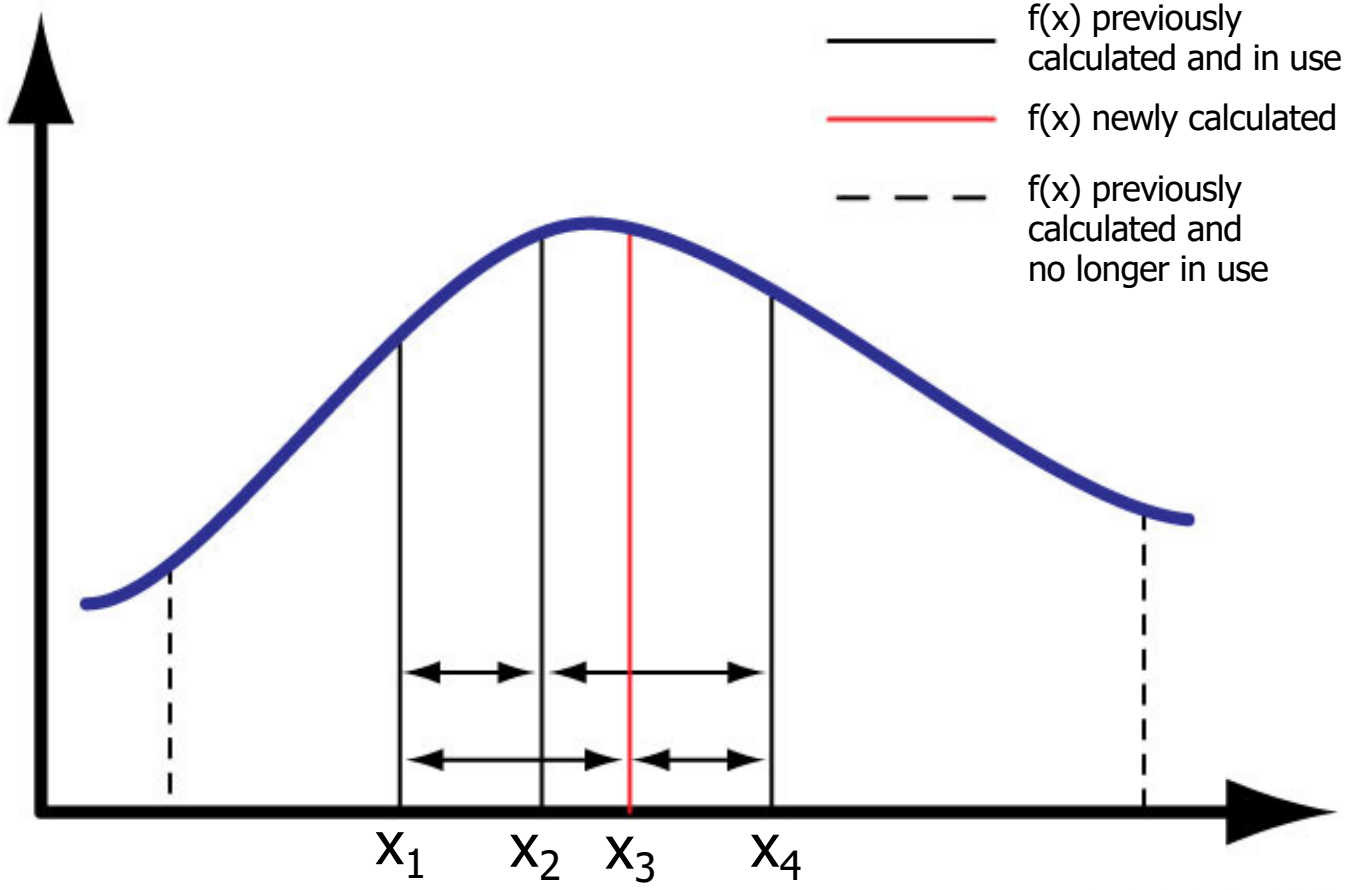
Golden Section search



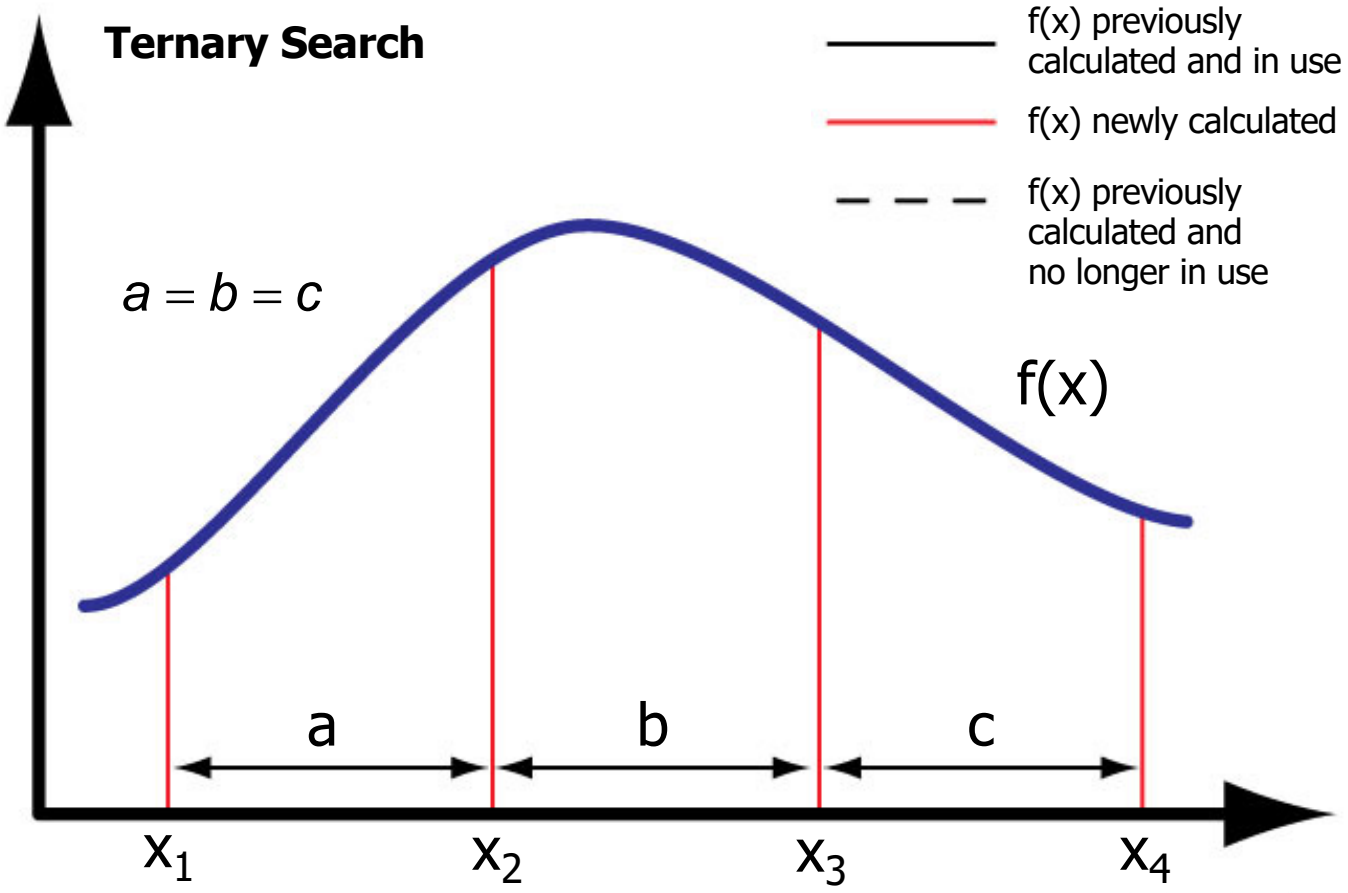
Golden Section search



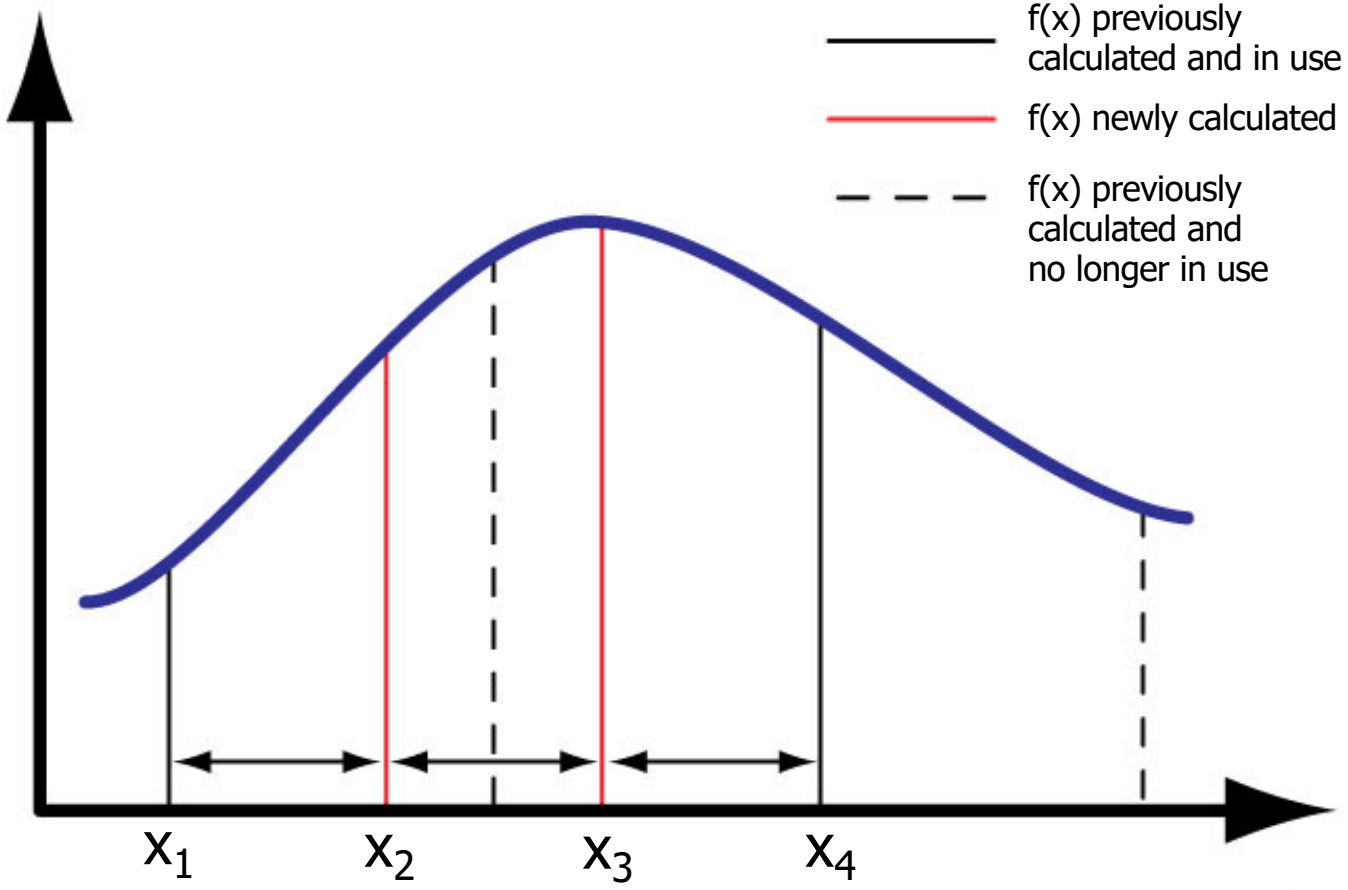
Golden Section search



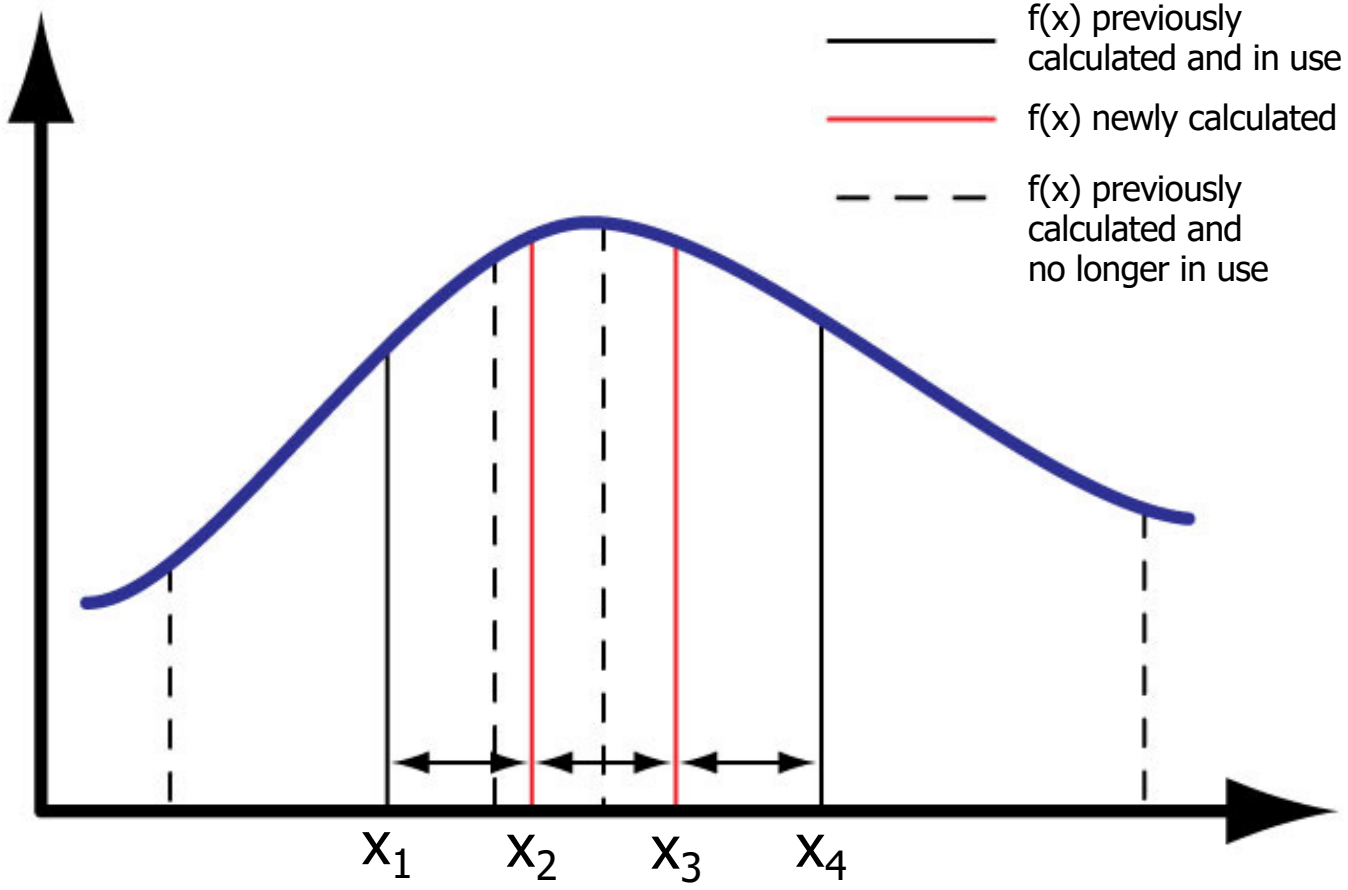
Ternary search



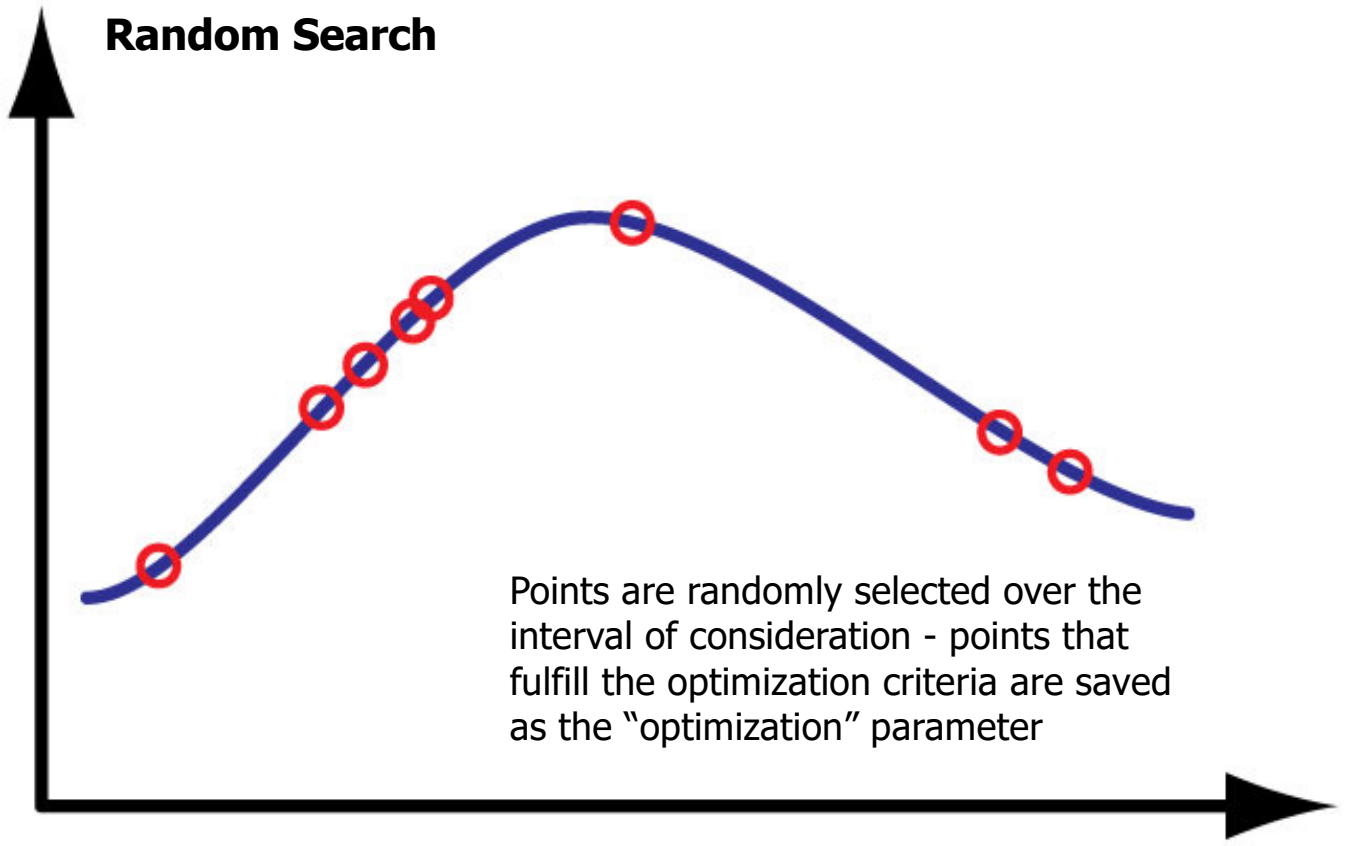
Ternary search



Ternary search



Random search



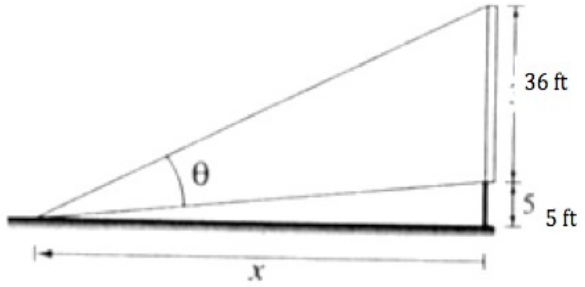
Example: Trajectory of an Artillery Shell

- Find the launch angle that will hit a target 200 meters away
- Find the firing angle that makes the shell go the furthest

ArtillerySimulation.m, *L24.m*

Example: Maximum Viewing Angle

- For best view in a theater, sit at distance x such that angle θ is maximum
- Find x with the configuration shown
- Applying Law of Cosines



- Angle is between 0 and $\pi/2$
- Cosine decreases from 1 at $\theta=0$, thus maximum angle corresponds to minimum $\cos\theta$