Numerical Methods Lecture 6 - Optimization

NOTE: The unit on differential equations will not be available online. We will use notes on the board only.

Topics: numerical optimization
- Newton again
- Random search
- Golden Section Search

Optimization - motivation

What?

- Locating where some function reaches a maximum or minimum

Find x where $f(x) = max$ or $f(x) = min$

Why?

- For example:

  A function represents the cost of a project
  minimize the cost of the project

When?

- When an exact solution is not available or a big pain

Example 1:

Given: $y = 5 + 6(x-3)^2$, Find x where y is minimum

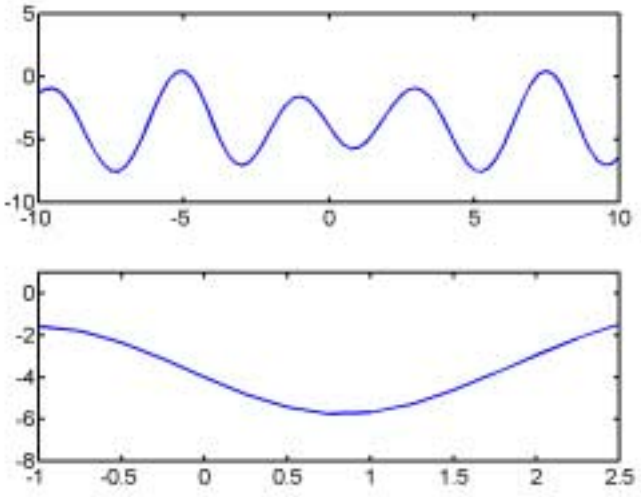
analytical solution: $\frac{dy}{dx} = 12(x-3) = 0$ ==> $x = 3$

Example 2:

Given: $y = \exp(\sin(x)) - 3\sin(1.5x) - 5$, Find x where y is minimum

Depends on the range we're interested in...

Having lots of local maxima and minima means having lots of zero slope cases. An exact solution would be a big pain...



Single variable - Newton

Recall the Newton method for finding a root of an equation

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad \text{where} \quad f'(x_k) = \frac{df(x)}{dx}$$

We can use a similar approach to find a min or max of $f(x)$

The min / max occurs where the slope is zero

So if we find the root of the derivative, we find the max / min location

$$f'(x) = g(x) = 0$$

Find roots of $g(x)$ using Newton

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

Example: Newton Method

find the maximum of this function

$$f(x) = x^4 - 5x^3 - 2x^2 + 24x$$

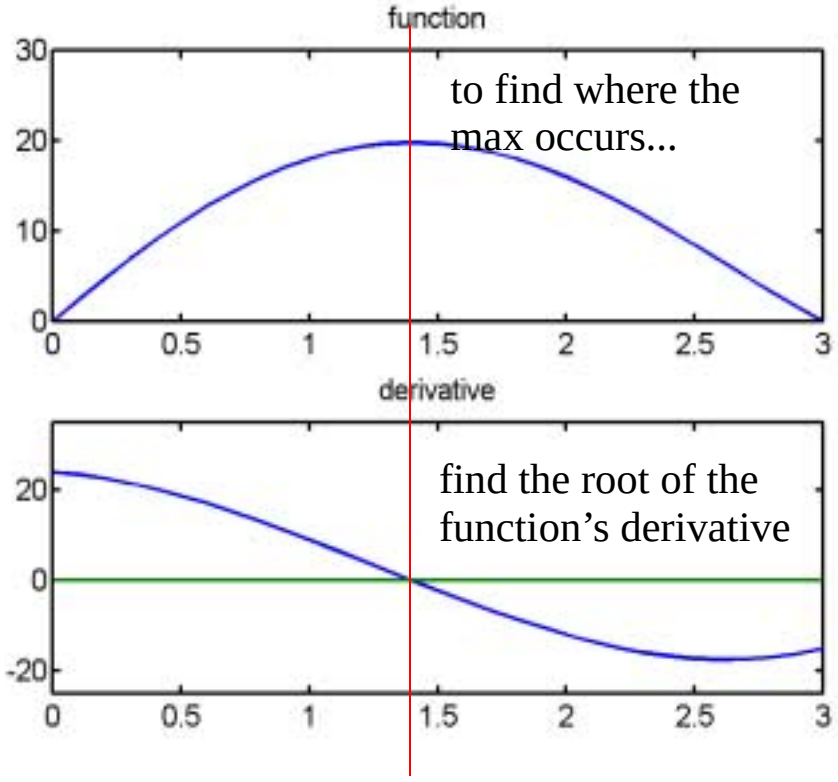
we'll need these

$$f'(x) = 4x^3 - 15x^2 - 4x + 24$$

$$f''(x) = 12x^2 - 30x - 4$$

for x = [0,3]

OR

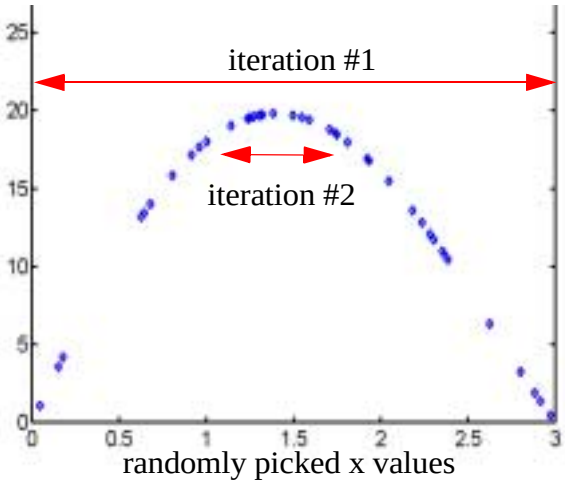


we can use central difference to replace the analytical derivatives.

single variable - Random search

A brute force method:

- 1) Sample the function at many random x values in the range of interest
- 2) If a sufficient number of samples are selected, a number close to the max and min will be found.
- 3) Re-set the range to a smaller sub-range and look again. Now the values are more dense, and you are more likely to find the max



A simple Mathcad program for random search

$ORIGIN \equiv 1$

$num := 10$ $f(x) := x^4 - 5 \cdot x^3 - 2 \cdot x^2 + 24 \cdot x$

$x := runif(num, 0, 3)$ $fx := f(x)$ $biggest := max(fx)$

$Find\_Max\_Index(fx) :=$
- $index \leftarrow 1$
- for $i \in 2 .. length(fx)$
  - $index \leftarrow i$ if $fx_i > fx_{index}$
- $index$

$index := Find\_Max\_Index(fx)$

$x_{index} = 1.374$

$biggest = 19.794$

Start over and narrow region

$x := runif(num, x_{index} - .25, x_{index} + .25)$ $fx := f(x)$ $biggest := max(fx)$

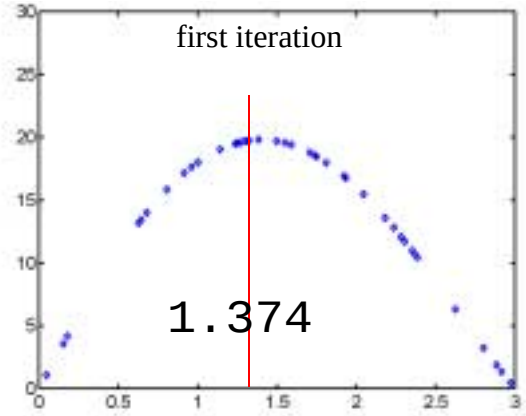
$index := Find\_Max\_Index(fx)$

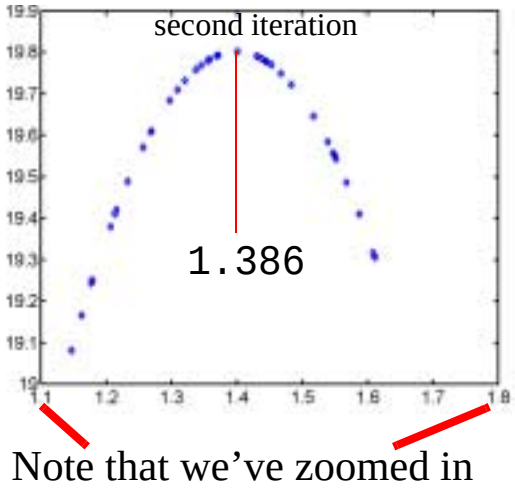
$x_{index} = 1.386$

$biggest = 19.8$

Note that if we make ‘num’ bigger, the first and second iterations will be much closer since the density of random numbers over the same area will increase, improving the liklihood of hitting the largest number.

Illustration of the code



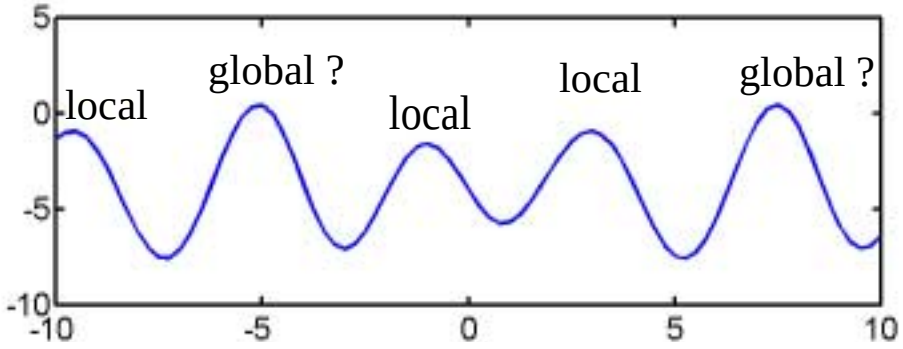


Advantages of random search:

- Simple to implement
- Distinguishing global from local maxima is easy

Example: same equation over a larger range

Finding roots of derivative (Newton) leaves us with lots of minima
and maxima (if Newton can even find them all)
Random Search can simply pick through and I.D. the biggest maximum
Since it compares all the f(x) values anyway.



Disadvantages of Random Search:

- Brute force method - uses no special information about the function or its derivatives.
- Random number pattern may miss narrow peaks
- Becomes very slow for multi-dimensional problems

if $y = f(x)$ needs 1000 numbers to get a good look...

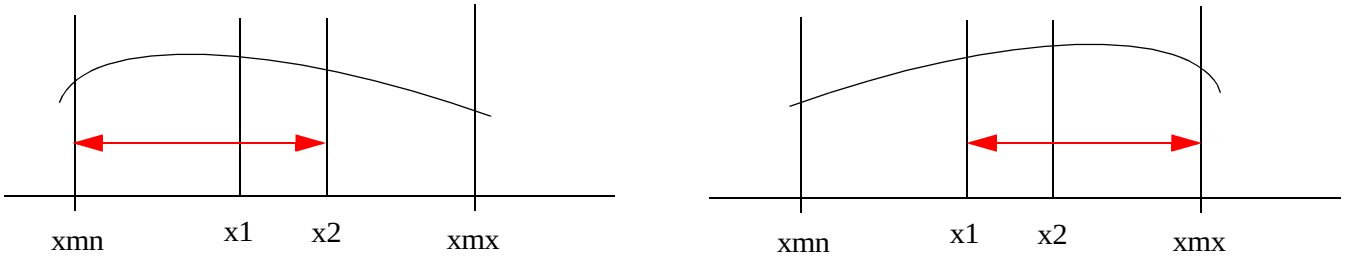
$z = f(p, q, x, y)$ needs 1000^4 numbers to get a good look

Single Variable - Golden Section Search Optimization Method

Similar to the bisection method

- Define an interval with a single answer (unique maximum) inside the range
  sign of the curvature does not change in the given range
- Divide interval into 3 sections by adding two internal points between ends
- Evaluate the function at the two internal points x1 and x2

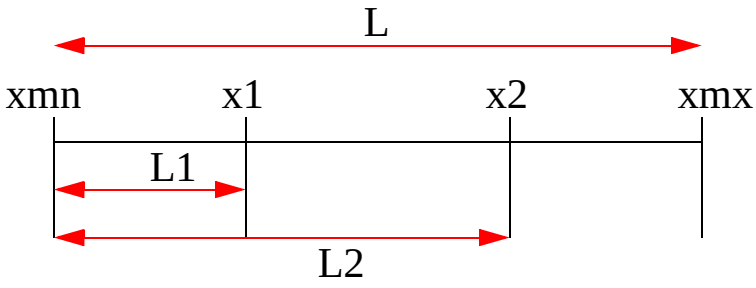
if f(x1) > f(x2)
the maximum is between xmn and x2
redefine range xmn = xmn, xmx = x2
if f(x1) < f(x2)
the maximum is between x1 and xmx
redefine range xmn = x1, xmx = xmx



look inside red range for second iteration

- Divide new smaller interval into 3 sections and start over

Q1: Where should we place the two internal points?
Let's see if we can pick some 'efficient' points



Set the following conditions:
(1) L = L1 + L2
(2) R = L/L2 = L2/L1

substitute (1) into (2) ==> 1 + R = 1/R

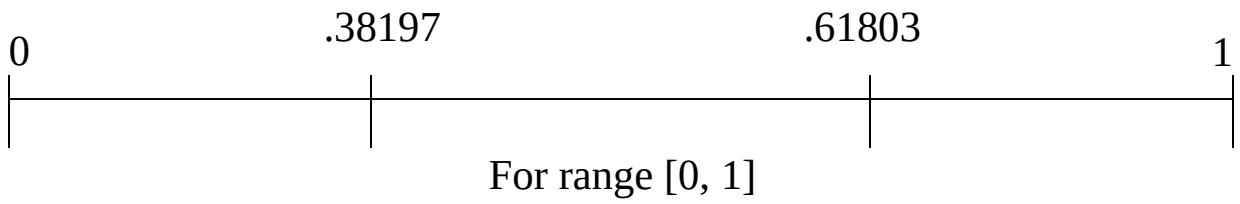
solve for R (R is called the Golden Ratio, named by ancient Greeks)

R = (sqrt(5)-1)/2 = .61803

So if the range is [0 , 1]

X1 = 1 - R = .38197

X2 = 0 + R = .61803

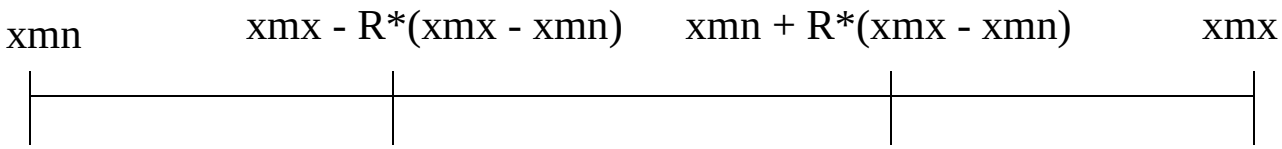


For range [0, 1]

The ratio of the larger length to the smaller remains L2 / L1 = R = .61803

So in general, for the range is [xmn , xmx]

X1 = xmx - R * (xmx - xmn)
X2 = xmn + R * (xmx - xmn)



For generic range [xmn, xmx]

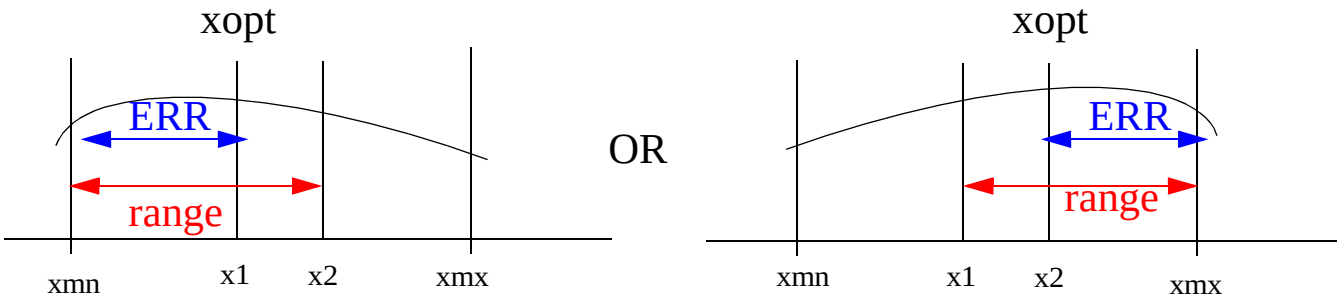
Why is this particular choice of internal points efficient?

> Because we enforce the ratio L/L2 = L2/L1,
> every successive iteration re-uses one of the previous internal values

Q2: How do we evaluate when we are close enough to maximum to stop?

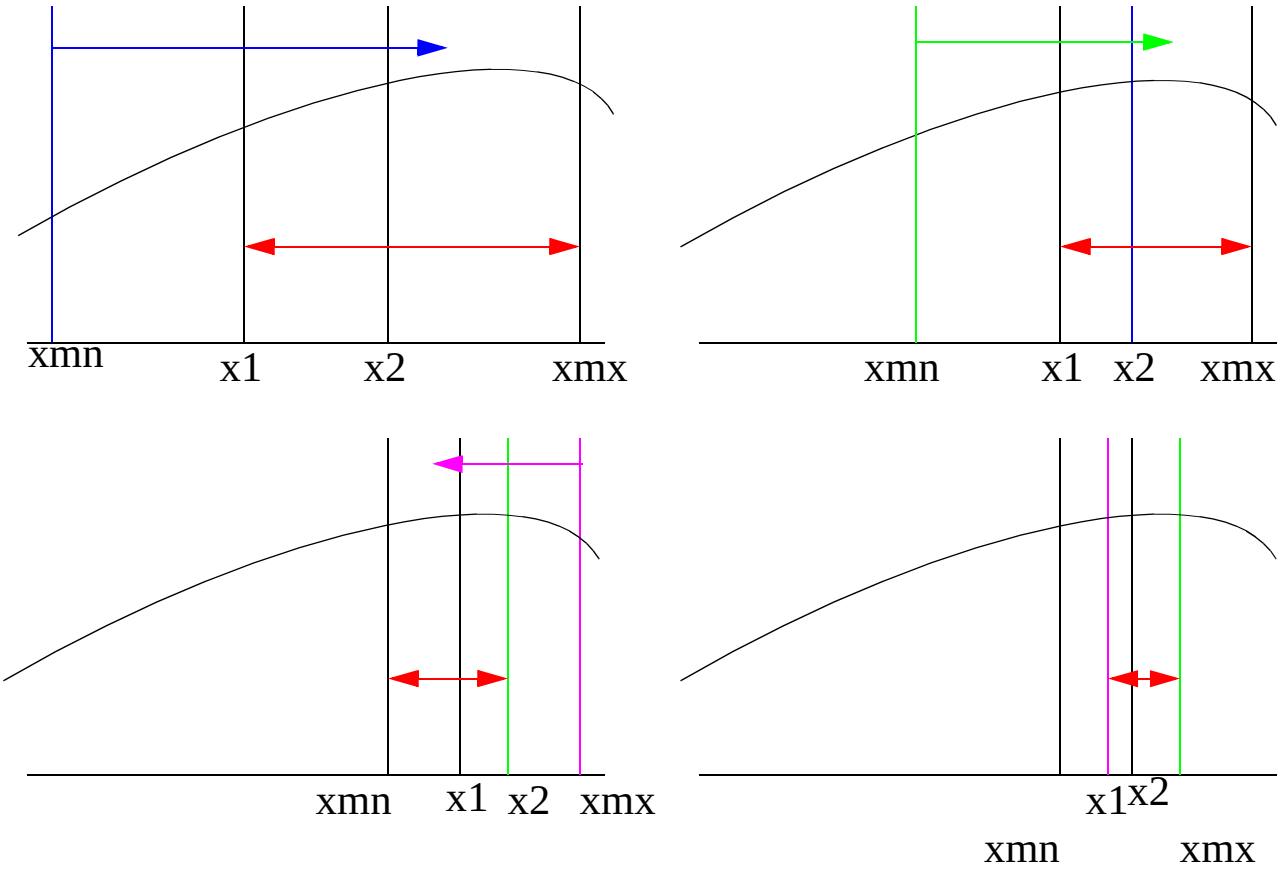
> We can't use err = |f(x)| like we did with root finding algorithms,
> since the value of f(x) says nothing about whether its a maximum value.

We'll instead evaluate error with respect to the x-axis



The maximum x distance between the current guess and the current range is called the error.
ERR = (1-R) * (xmx - xmn)

So if we set the tolerance to .001
the final x guess is within .001 of the x location of the actual maximum
A few iterations



A working algorithm

GoldenSearch (f , a , b , tol) :=

```
grat ← (√5 − 1) / 2
d ← grat · ( b − a )
x2 ← a + d
x1 ← b − d
err ← 100
numit ← 0
while err > tol
    numit ← numit + 1
    if f ( x1 ) > f ( x2 )
        xopt ← x1
        err ← ( 1 − grat ) · | ( b − a ) / xopt |
        if err > tol
            b ← x2
            x2 ← x1
            d ← grat · ( b − a )
            x1 ← b − d
    otherwise
        xopt ← x2
        err ← ( 1 − grat ) · | ( b − a ) / xopt |
        if err > tol
            a ← x2
            x1 ← x2
            d ← grat · ( b − a )
            x2 ← a + d
( xopt )
( f ( xopt ) )
```

$$f(x) := 2\cdot\sin(x) - \frac{x^2}{10} \qquad out := GoldenSearch(f, 0, 4, .01) \qquad out = \begin{pmatrix} 1.528 \\ 1.765 \end{pmatrix}$$

Reference:C:\Mine\Mathcad\Tutorials\MyFunctions.mcd

$x := Create\_Vector(0, 4, .1)$