Optimization and Root Finding

JMR Chapter 7

BTRY/STSCI 4520

The Optimization Problem

We have some function $f(x)$ and we want to find

$$x^* = \operatorname{argmax} f(x)$$

the x that produces the largest value in $f(x)$.

If f is "nice", we can work this out with algebra and calculus.

Otherwise, we can use a computer to search for the optimum.

Example

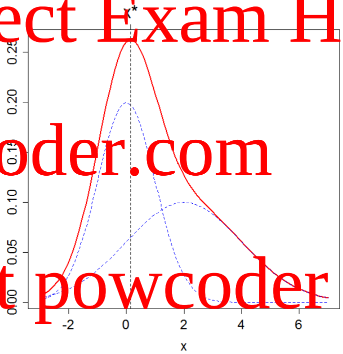
Consider the normal mixture density

$$f(x) = \frac{1}{2\sqrt{2\pi}} e^{-x^2/2} + \frac{1}{4\sqrt{2\pi}} e^{-(x-2)^2/8}$$

X comes from $N(0,1)$ with probability 0.5 and $N(2,2)$ with probability 0.5.

What is the mode of $f(x)$?

No algebraic solution available.



The Root-Finding Problem

We might also want to find where $f(x)$ has a specific value

$$x^+ = \{x : f(x) = c\} = \{x : f(x) - c = 0\}$$

Called 'root-finding' because we try to find a root of $f(x) - c$.

Frequently comes up when you want to balance an objective "How much do I have to pay now so that my return will be R?". (eg JMR's example – rather contrived)

A Classical Problem

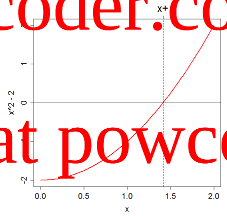
What is the value of $\sqrt{2}$?

- Classical result: no fractional expression $\sqrt{2} \neq a/b$ for all integers a and b.
- Easy expression to work with, but what are the actual digits?

Find a numerical solution of the problem

$$x^2 - 2 = 0$$

in $[0, \infty)$.



Different use of numerics: can solve the problem symbolically, but want a numerical representation of the answer.

Reducing Optimization to Root Finding and Vice Versa

You can always solve a root-finding problem with optimization:

$$f(x^+) - c = 0 \Rightarrow x^+ = \operatorname{argmax} \; -(f(x) - c)^2$$

If $f(x)$ is *differentiable*, we can do the converse:

$$x^* = \operatorname{argmax} \; f(x) \Rightarrow f'(x^*) = 0$$

but you need to check you are at a maximum.

Nonetheless, strategies specific to the problem generally work best.

In statistics, optimization is most used, but root finding provides useful motivation.

Root-Finding 1: The Bisection Method

- Suppose that $f(x)$ is monotone (increasing or decreasing) on some range $[a\ b]$.
- Then $f(x) = 0$ at at most one $x \in [a\ b]$.
- We can test whether $f(x)$ crosses zero by the sign of $f(a)f(b)$.

Bisection method searches by successively dividing intervals in two:

1. Start with $a < b$ such that $f(a)f(b) < 0$.
2. Let $c = (a + b)/2$ be the midpoint of a and b.
3. If $f(a)f(c) < 0$, $f(x)$ crosses 0 in $[a\ c]$, set $b = c$.
4. Otherwise, $f(x)$ crosses 0 in $[c\ b]$, set $a = c$.
5. Repeat.

Graphically and in Code

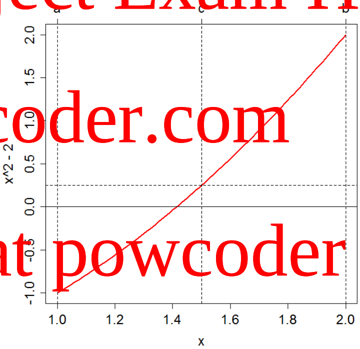
```
fn2 = function(x){ return(x^2 -2) }

a = 1   # Starting values: 1^2
b = 2   # and 2^2 > 2

fa = fn2(a); fb = fn2(b)

for(i in 1:3){
  c = (a+b)/2; fc = fn2(c)

  if( fa*fc < 0 ){
    b = c; fb = fn2(b)
  } else{
    a = c; fa = fn2(a)
  }
}
```

Step 1: set $b = c$.

Graphically and in Code

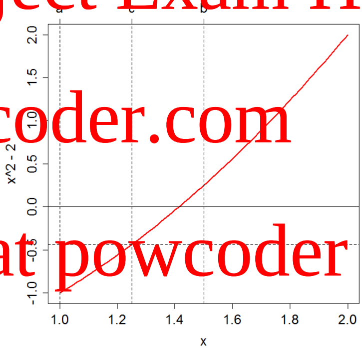
```
fn2 = function(x){ return(x^2 -2) }

a = 1   # Starting values: 1^2 < 2
b = 2   # and 2^2 > 2

fa = fn2(a); fb = fn2(b)

for(i in 1:3){
  c = (a+b)/2; fc = fn2(c)

  if( fa*fc < 0 ){
    b = c; fb = fc
  } else{
    a = c; fa = fc
  }
}
```



Step 2: set $a = c$.

Graphically and in Code

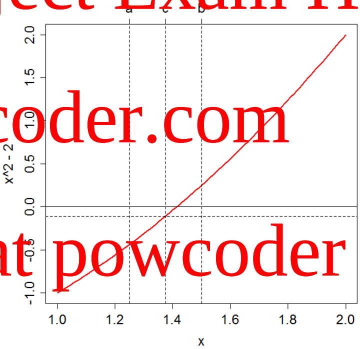
```
fn2 = function(x){ return(x^2 -2) }

a = 1   # Starting values: 1^2 < 2
b = 2   # and 2^2 > 2

fa = fn2(a); fb = fn2(b)

for(i in 1:3){
  c = (a+b)/2; fc = fn2(c)

  if( fa*fc < 0 ){
    b = c; fb = fc
  } else{
    a = c; fa = fc
  }
}
```



Step 3: set $a = c$.

Graphically and in Code

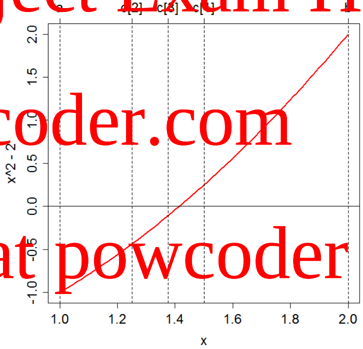
```
fn2 = function(x){ return(x^2 -2) }

a = 1   # Starting values: 1^2
b = 2   # and 2^2 > 2

fa = fn2(a); fb = fn2(b)

for(i in 1:3){
  c = (a+b)/2; fc = fn2(c)

  if( fa*fc < 0 ){
    b = c; fb = fc
  } else{
    a = c; fa = fc
  }
}
```



Sequence of approximations to $\sqrt{2}$.

Convergence Criteria

- It's unlikely that we will ever find $f(x) = 0$ exactly.
- So we need some way to decide that our solution is "good enough".

- Usually decided by a tolerance.
  - For root finding, stop when $|f(c)| < \epsilon$.
  - ϵ chosen based on required accuracy and machine tolerance (default is often around `1e-8`).
  - Note: $f(x)$ small does not necessarily guarantee $|x - x^*|$ small.

- We also usually set a maximum number of iterations, so we know we will terminate sometime.

In Code

```
BisectionSearch = function(fn,a,b,tol=1e-8,maxit=100){
  fa = fn(a); fb = fn(b);    # Initialization

  tol.met = FALSE     # No tolerance met
  iter = 0            # No iterations

  while(!tol.met){
    c = (a+b)/2; fc = fn(c)

    if( fa*fc < 0 ){ b = c; fb = fc }
    else{ a = c; fa = fc }

    iter = iter + 1 # Update iterations and tolerance
    if( abs(fc) < tol | iter > maxit ){ tol.met=TRUE }
  }
  return(list(sol=c,iter=iter))
}
```

Output

Including some `print` commands in the function:

```
> sol = BisectionSearch(fn2,1,2)
[Iter Value Convergence
[1] 1    1.5    0.25
[1] 2    1.25   -0.4375
[1] 3    1.375  -0.109375
[1] 4    1.4375 0.06640625
...
[1] 26   1.4142 -2.63102e-08
[1] 27   1.4142 -5.23681e-09
```

And if we compare this to R's value:

```
> sol$sol
[1] 1.414214
> sqrt(2)
[1] 1.414214
> sqrt(2)-sol$sol
[1] 1.851493e-09
```

Analysis of Convergence of Root Finding Methods

So how large is our error?

- Let $D = b - a$ for the starting guess.
- If we estimate $\hat{x} = (a + b)/2$, we know that $|\hat{x} - x^{+}|$ is at most $D/2$.
- Next iteration, we halve the size of the interval again.
- After k steps, the error can be at most $2^{-k}D = O(2^{-k})$.
- In this case, we can control error by controlling the number of iterations.

Properties (unlike Newton-Raphson, next)

- Doesn't require derivatives.
- Explicit convergence error from number of steps.
- Very simple to implement.
- **But** slow and doesn't generalize to more dimensions.

Root-Finding 2: Newton-Raphson

Suppose that $f(x)$ has a derivative.

- Start at initial guess x_0 and take a linear approximation

$$f(x) \approx f(x_0) + (x - x_0)f'(x_0)$$

- Now we can set the linear approximation to 0

$$f(x_0) + (x - x_0)f'(x_0) = 0 \Rightarrow x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

- Now set x_1 to be our guess and start again.

As before, stop when $|f(x)| < \epsilon$ or too many iterations.

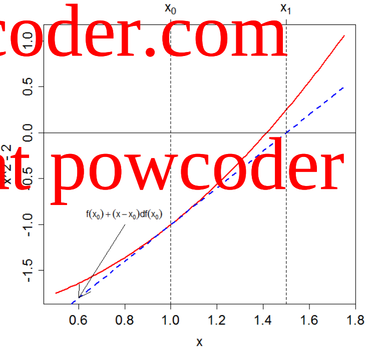
Graphically and in Code

We first need to define a derivative

$$\frac{d}{dt}(x^2 - 2) = 2x$$


```
dfn2 = function(x){ return( 2*x ) }
```

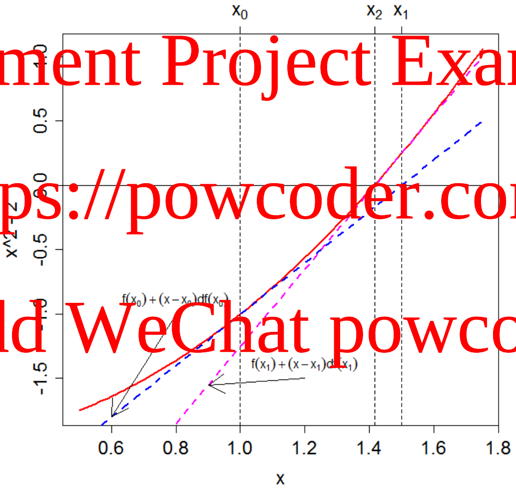
Start at 1 as an initial guess




```
x0 = 1
f0 = fn2(x0)
df0 = dfn2(x0)

x1 = x0 - f0/df0
```

The Next Iteration



And continue.

A Formal Function

```
NewtonRaphson = function(fn,dfn,x0,tol=1e-8,maxit=100){
  f0 = fn(x0); df0 = dfn(x0);     # Initialization

  tol.met = FALSE     # No tolerance met
  iter = 0            # No iterations


  while(!tol.met){
    x0 = x0 - f0/df0
    f0 = fn(x0); df0 = dfn(x0)

    iter = iter + 1 # Update iterations and tolerance
    if( abs(f0) < tol | iter > maxit ){
      tol.met=TRUE
    }
  }
  return(list(sol=x0,iter=iter))
}
```

Validation

Very fast convergence.


```
> sol = NewtonRaphson(fn2,dfn2,1)
Iter   Estimate Convergence
[1] 1     1.5      0.25
[1] 2     1.41667 0.00694
[1] 3     1.41422 6.007e-06
[1] 4     1.41421 4.511e-12
```



```
> sol$sol
[1] 1.414214
> sqrt(2)
[1] 1.414214
> sqrt(2)-sol$sol
[1] -1.594724e-12
```

Convergence Analysis

A bit of mathematics:

Write a Taylor-series approximation to f near x_n as

$$f(x) = f(x_n) + (x - x_n)f'(x_n) + R_1$$

where the remainder R_1 is

$$R_1 = \frac{1}{2}(x - x_n)^2 f''(\tilde{x})$$

for some $\tilde{x}$ between x and x_0.

Now let's look at this approximation at the root x^+

$$0 = f(x^+) = f(x_n) + (x^+ - x_n)f'(x_n) + \frac{1}{2}(x^+ - x_n)^2 f''(\tilde{x})$$

Convergence Analysis

$$0 = f(x_n) + (x^+ - x_n)f'(x_n) + \frac{1}{2}(x^+ - x_n)^2 f''(\tilde{x})$$

Re-arrange this and divide by $f'(x_n)$:

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

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

$$|x^+ - x_{n+1}| = \left|\frac{f''(\tilde{x})}{f'(x_n)}\right|(x^+ - x_n)^2$$

- Error $\epsilon_n = x^+ - x_n$ is squared each iteration: $\epsilon_{n+1} = O(\epsilon_n^2)$ (bisection search just halves it).
- But only works if $f''(\tilde{x})/f'(x_n)$ stays small *and* we start close to x^+.

Convergence Issues

Newton-Raphson can fail in a number of ways:

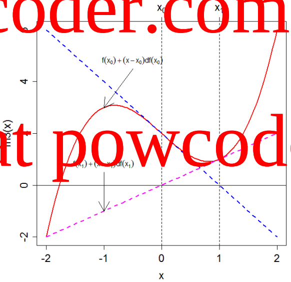
- Function not smooth enough (needs second derivatives to be fast).
- $f'(x)$ is zero at the root.
- Infinite cycles.

Consider $f(x) = x^3 - 2x + 2$,
$f'(x) = 3x^2 - 2$:

- Start $x_0 = 0$,
  $x_1 = 0 - (1/-1) = 1$.

- $x_2 = 1 - 1/1 = 0 = x_0$

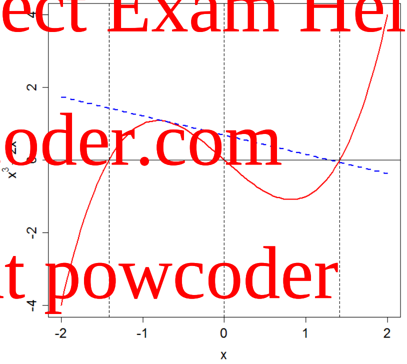
Never ends!



But you usually have to work hard to find these examples.

What If $f(x)$ Crosses 0 Multiple Times?

- Optimizers will give you one of the zeros.
- Newton-Raphson usually gives you the root closest to where you started.
- But not always!
- Common strategy: try starting from multiple places.
- Bisection search harder to analyze in this case.



Secant Method

Calculating derivatives can sometimes be inconvenient (and users often get them wrong).

Instead, use two initial guesses x_0, x_1.

- Draw a line through $(x_0, f(x_0))$ and $(x_1, f(x_1))$:

$$f(x_1) + (x - x_1)\frac{f(x_0) - f(x_1)}{x_0 - x_1}$$

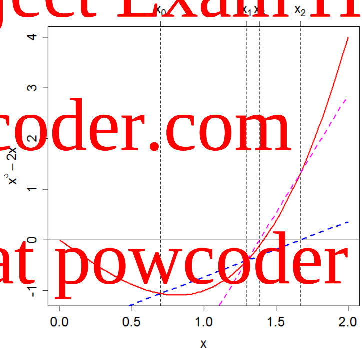
- Find the point where this crosses zero

$$x_2 = x_1 - f(x_1)\frac{x_0 - x_1}{f(x_0) - f(x_1)}$$

- Iterate.

Graphically

- Same convergence properties as Newton-Raphson.
- Same problems with local solutions, infinite loops.
- Still requires smoothness.
- Trade-off need for $f'(x)$ with extra calculation and two starting points.



Optimization 1: Newton-Raphson

More frequently (in statistics) we want to optimize.

Newton-Raphson for Optimization: (usually just "Newton's Method")

- To find the maximum of $f(x)$, look for $f'(x) = 0$.
- Use $f'(x)$ in place of $f(x)$ in algorithms above.

- Iteration changes to

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

- Need a maximum: check $f''(x_n) < 0$ (or conversely for a minimum).
- If at the wrong sort of stationary point, try again.

The Mode of a Mixture Distribution

$$f(x) = \frac{1}{2\sqrt{2\pi}} e^{-x^2/2} + \frac{1}{4\sqrt{2\pi}} e^{-(x-2)^2/8}$$

$$f'(x) = -\frac{x}{2\sqrt{2\pi}} e^{-x^2/2} - \frac{(x-2)}{16\sqrt{2\pi}} e^{-(x-2)^2/8}$$

$$f''(x) = \frac{x^2-1}{2\sqrt{2\pi}} e^{-x^2/2} + \frac{\frac{(x-2)^2}{4} - 1}{16\sqrt{2\pi}} e^{-(x-2)^2/8}$$

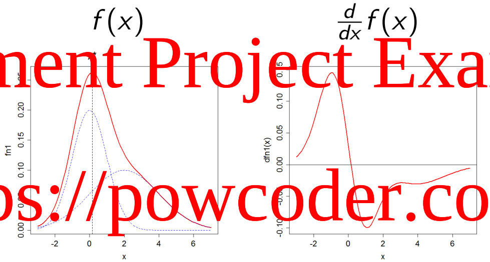
Expressed in terms of the normal density:

```
fn1 = function(x){
  return( 0.5*dnorm(x) + 0.5*dnorm(x,sd=2,mean=2) )
}

dfn1 = function(x){
  return( -x*dnorm(x)/2 - (x-2)*dnorm(x,mean=2,sd=2)/8 )
}

d2fn1 = function(x){
  return( (x^2-1)*dnorm(x)/2 + ((x-2)^2/4-1)*dnorm(x,mean=2,sd=2)/8 )
}
```

The Usual Problems Occur



Newton-Raphson will only converge close to the truth.

```
> est = NewtonRaphson(dfn1,d2fn1,0)
[1] 1 0.1516 0.00017
[1] 2 0.1525 2.701e-08
[1] 3 0.1525 7.008e-16
```

```
> est = NewtonRaphson(dfn1,d2fn1,2)
[1] 1 2.9632  -0.028714
[1] 2 16.098  -5.71e-12
```

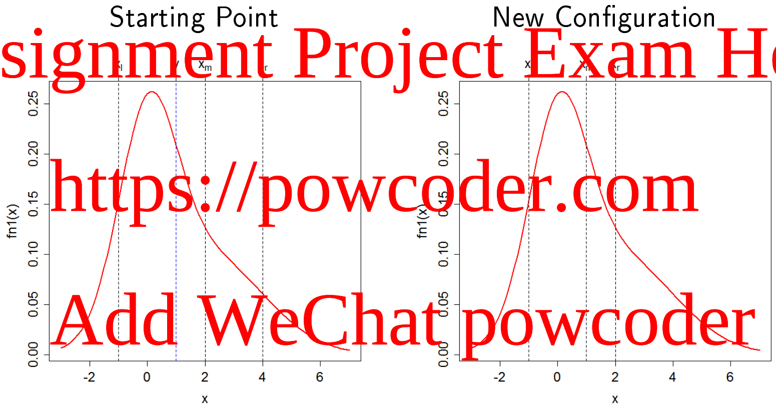
If $f(x) \to -\infty$ for $|x| \to \infty$, we must at least get a local maximum.

Golden Section Search

Analogue to bisection search for zeros. Do not want to require derivatives.

- Begin with left point x_l, right point x_r and middle x_m.
- Assume $f(x_m) > f(x_l)$ and $f(x_m) > f(x_r)$.
- Choose a new point y in the larger of $[x_l \; x_m]$ and $[x_m \; x_r]$.
- Suppose $y \in [x_l \; x_m]$,
  - If $f(y) > f(x_m)$ the maximum is in $[x_l \; y]$; set $x_m = y$ and $x_r = x_m$.
  - Otherwise, the maximum is in $[y \; x_r]$; set $x_l = y$.
- Conversely for $y \in [x_m \; x_r]$.

Graphical Golden Section

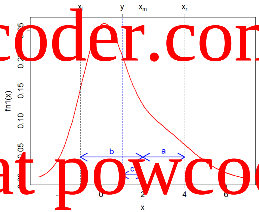


- Unimodality crucial – allows us to conclude where maximum *must* lie.
- y in largest interval = most efficient exploration.

The Golden Section: Choosing y

- Place y so that we always reduce the interval by the same amount.
- Choose the initial interval so that the ratio of smaller to larger interval doesn't change.

- Shift x_r to x_m; ratio is $(b-c)/c$
- Shift x_l to y; ratio is c/a.
- Both should be equal to $b/a = \rho$.



$$\frac{a}{c} = \frac{b}{a} \rightarrow c = \frac{a^2}{b} \text{ substitute in } \frac{b-c}{c} = \frac{b}{a} \text{ yields } \rho^2 - 1 = \rho$$

The Golden Section

$$\rho^2 - \rho - 1 = 0$$

is solved for

$$\rho = \frac{1+\sqrt{5}}{2};$$

the "Golden ratio".

- To work out how large c is, note that $a = b - c$ (we shrink the same amount in either case).
- $\frac{c}{b} = 1 - \frac{a}{b} \Rightarrow c = \frac{b}{1+\rho}$ (algebra is a bit involved).
- Or $y = x_m - \frac{x_m - x_l}{1+\rho}$.
- Note: notation here goes right to left, book goes left to right.

Pseudo-Code

Because R code doesn't fit when still readable - see lecture code.

Start with x_l, x_r, $x_m = x_l + (x_r - x_l)/(1 + \rho)$.

Repeat:

1. If $x_r - x_m > x_m - x_l$, $y = x_m + (x_r - x_m)/(1 + \rho)$
   - If $f(y) > f(x_m)$, set $x_l = x_m$ and $x_m = y$.
   - Else set $x_r = y$
2. Else $y = x_m - (x_m - x_l)/(1 + \rho)$
   - If $f(y) > f(x_m)$, set $x_r = x_m$ and $x_m = y$.
   - Else set $x_l = y$

until $x_r - x_l < \epsilon$ or too many iterations.

In practice, just update $f(x_l)$, $f(x_m)$, $f(x_r)$ from $f(y)$ or $f(x_m)$ as appropriate to avoid re-evaluating.

Some Notes

- Tolerance criteria can be any of

$$|x_{n+1} - x_n| < \epsilon, \quad |f(x_{n+1}) - f(x_n)| < \epsilon, \quad |f'(x_{n+1})| < \epsilon$$

  Step size in x, improvement in $f(x)$ and local rate of change of $f(x)$.

  None guarantee $|x_n - x^*| < \epsilon$; often require all to be met.
- Convergence is local – with multiple maxima in a function, each of these will find just one.
- What if $f''(x) > 0$ or the maximum is at the edge of the interval? Try expanding the interval in the upward direction (more later).
- Some strategies switch back and forth between optimizers.

Why?

Optimization has multiple scientific uses.

In statistics; most important is maximum likelihood estimation.

$X_1, \ldots, X_n \sim f(x; \theta)$, estimate θ.

Choose θ to make $X_1, \ldots, X_n$ *most probable*.

$$\hat{\theta} = \operatorname{argmax} P(X_1, \ldots, X_n | \theta) = \operatorname{argmax} \prod_{i=1}^{n} f(X_i | \theta)$$

is the *maximum likelihood estimator*.

Usually work with the log probability

$$\hat{\theta} = \operatorname{argmax} \sum_{i=1}^{n} \log f(X_i | \theta)$$

Sometimes calculable analytically, but not always.

Summary

- Root finding and optimization as crucial numerical tools.
- Can be converted into each other.
- Locally, Newton-Raphson methods make convergence towards the truth very fast.
- Bisection/Golden Section methods don't require derivatives.
- You always run the risk of not converging, or only finding a local optimum.
- Next: optimization over multiple quantities.