Numerical Methods

Max Turgeon

STAT 3150–Statistical Computing

Lecture Objectives

- Describe the main differences between computer arithmetic and “normal” arithmetic.
- Apply root finding methods for one-dimensional problems.

Motivation

- Many estimators can be defined as solutions to a given equation or optimization problem.
- For the next few weeks, we will discuss **numerical methods** and **optimization**
- This will also serve as a good introduction to using R as a programming language.

Question

- Can you give examples of estimators defined by solving an equation $f(x) = c$?
- Can you recall an example from the notes/assignments?

Testing equality i

- To test for equality of integers, booleans or strings, we can use `==`.
    - `3 == 4`, `TRUE == FALSE`, `"hello" == "world"`.
- But with decimal numbers, the equality operator may behave in surprising ways

```
# Expected
(0.5 + 0.5) == 1
```

```
## [1] TRUE
```

Testing equality ii

```r
# Unexpected
(0.1 + 0.2) == 0.3
```

```
## [1] FALSE
```

```r
# Why?
0.3 - (0.1 + 0.2)
```

```
## [1] -5.551115e-17
```

Testing equality iii

- In computer's memory, decimal numbers are essentially represented in *binary* scientific notation.
    - Which leads to rounding errors that may be hard to predict.
- R gives us two functions to test equality more carefully:
    - `all.equal`: Tests for "near equality", i.e. within a tolerance level
    - `identical`: Tests for whether two objects are identical (including length, attributes, etc.).

Testing equality iv

```r
all.equal(0.1 + 0.2, 0.3)
```

```
## [1] TRUE
```

```r
identical(0.1 + 0.2, 0.3)
```

```
## [1] FALSE
```

Testing equality v

```r
# But be careful!
all.equal(1, 2)
```

```
## [1] "Mean relative difference: 1"
```

```r
# Better
isTRUE(all.equal(1, 2))
```

```
## [1] FALSE
```

Testing equality vi

- Another approach: check whether `abs(x - y) < epsilon`, for an `epsilon` of your choice.

```
abs(0.3 - (0.2 + 0.1)) < 10^-10
```

```
## [1] TRUE
```

Overflow and underflow

- Another way in which computer arithmetic can be surprising: very small and very large numbers.
    - Small numbers may be rounded down to zero.
    - Large numbers will be turn into `Inf`.
- In both cases, there are two strategies that can help:
    - Simplify expressions by hand as much as you can first: $\frac{n!}{(n-2)!} = n(n-1)$.
    - Compute on logarithmic scale, and convert answer back to original scale with `exp`.

Example i

- We know the Poisson mass function is

$$P(X = k) = \frac{e^{-\lambda}\lambda^k}{k!} > 0.$$

- But when k is large, we may run into underflow issues.

```
# d is for density
dpois(100, lambda = 1)
```

```
## [1] 3.941866e-159
```

Example ii

```r
dpois(200, lambda = 1)
```

```
## [1] 0
```

```r
# Use logarithms
dpois(200, lambda = 1, log = TRUE)
```

```
## [1] -864.232
```

Exercise

Using the properties of logarithms, evaluate

$$\frac{\Gamma\left(\frac{n-1}{2}\right)}{\Gamma\left(\frac{1}{2}\right)\Gamma\left(\frac{n-2}{2}\right)},$$

for $n = 400$. Use `lgamma` to evaluate the Gamma function on the logarithmic scale.

Solution

```r
n <- 400
# With gamma
(gamma(0.5*(n-1))/(gamma(0.5)*gamma(0.5*(n-2))))
```

```
## [1] NaN
```

```r
# With lgamma
exp(lgamma(0.5*(n-1)) - lgamma(0.5) - lgamma(0.5*(n-2)))
```

```
## [1] 7.953876
```

Finding the roots of a function

- The first class of numerical methods we will look at our **root finding algorithms** (in one dimension).
- Assume we have a continuous function $f(x)$ of one variable. For a given constant c, we want to find the values x such that $f(x) = c$.
    - Equivalent to replacing $f(x)$ with $f'(x) = f(x) - c$ and looking for when $f'(x) = 0$.
- We will look at two methods:
    - Bisection method
    - Brent's method

Bisection method i

- Assume that we have $f(a)$ and $f(b)$ are nonzero and have opposite sign.
    - Exactly one is negative, the other is positive.
- Because f is continuous, the *Intermediate Value Theorem* tells us that there must be a value $x \in (a, b)$ such that $f(x) = 0$.
    - It may not be unique, but there's at least one such x.

Bisection method ii

- With the bisection method, we look at the mid-point of $[a, b]$: $x_1 = \frac{b-a}{2} + a = \frac{b+a}{2}$, and we evaluate $f(x_1)$.
  - If $f(a)$ and $f(x_1)$ have the **same sign**, then the root is in the interval (x_1, b).
  - If $f(a)$ and $f(x_1)$ have **opposite sign**, then the root is in the interval (a, x_1).
- We then repeat the process on the new interval, which gives us a sequence of “guesses” $x_1, x_2, x_3, \ldots$.
  - This sequence is **guaranteed** to converge to a root of $f(x) = 0$.
- We stop when we are “close enough”, i.e. when $|f(x_n)| < \epsilon$.

Demo

See this video: `https://youtu.be/zkd6CLfNNe8`

Example i

- We will look at the function

$$f(x) = a^2 + x^2 + \frac{2ax}{n-1} - (n-2),$$

for $a = 0.5$ and $n = 20$, on the interval $(0, 5n)$.

```
a <- 0.5
n <- 20
# First create a function
fun <- function(x) {
  a^2 + x^2 + 2*a*x/(n-1) - n + 2
}
```

Example ii

```r
# Check output at interval bounds
x_lb <- 0 # Lower bound
x_ub <- 5*n # Upper bound

c(fun(x_lb), fun(x_ub))
```

```
## [1]  -17.750 9987.513
```

Example iii

```
# Set up----
x_next <- 0.5*(x_ub - x_lb) + x_lb # Midpoint
epsilon <- 10^-10
f_lb <- fun(x_lb)
f_ub <- fun(x_ub)
f_next <- fun(x_next)
iterations <- 0
```

Example iv

```r
while(abs(f_next) > epsilon) {
  iterations <- iterations + 1
  if (f_ub*f_next > 0) {
    x_ub <- x_next # same sign, move left
    f_ub <- fun(x_ub) } else {
    x_lb <- x_next # opposite sign, move right
    f_lb <- fun(x_lb) }
  x_next <- 0.5*(x_ub - x_lb) + x_lb
  f_next <- fun(x_next)
}
```

Example v

```r
# Our estimate the solution f(x) = 0
x_next
```

```
## [1] 4.186841
```

```r
# Number of iterations
iterations
```

```
## [1] 40
```

Exercise

Use the bisection method to find the solution to the equation

$$\cos(x) = x^3.$$

Solution i

- First, we can look at the solution of $g(x) = 0$, for $g(x) = \cos(x) - x^3$.
- Based on our knowledge of these two functions, we deduce that a solution, if it exists, must be positive.
- Let's look at the interval $[0, 2]$

```r
# First create a function
g_fun <- function(x) {
  cos(x) - x^3
}
```

Solution ii

```r
# Check output at interval bounds
x_lb <- 0 # Lower bound
x_ub <- 2 # Upper bound

c(g_fun(x_lb), g_fun(x_ub))
```

```
## [1]  1.000000 -8.416147
```

Solution iii

```
# Set up----
x_next <- 0.5*(x_ub - x_lb) + x_lb # Midpoint
epsilon <- 10^-10
g_lb <- g_fun(x_lb)
g_ub <- g_fun(x_ub)
g_next <- g_fun(x_next)
iterations <- 0
```

Solution iv

```
while(abs(g_next) > epsilon) {
  iterations <- iterations + 1
  if (g_ub*g_next > 0) {
    x_ub <- x_next # same sign, move left
    g_ub <- g_fun(x_ub) } else {
    x_lb <- x_next # opposite sign, move right
    g_lb <- g_fun(x_lb) }
  x_next <- 0.5*(x_ub - x_lb) + x_lb
  g_next <- g_fun(x_next)
}
```

Solution v

```
# Our estimate the solution g(x) = 0
x_next
```

```
## [1] 0.865474
```

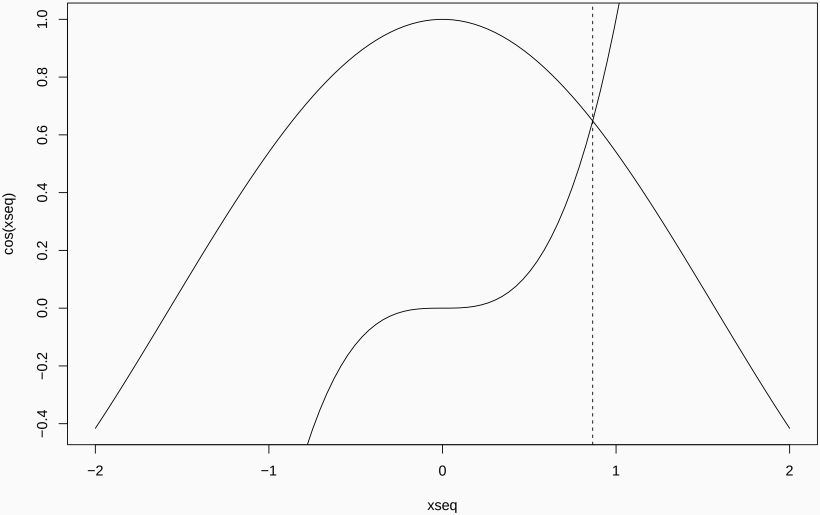
```
# Number of iterations
iterations
```

```
## [1] 34
```

Solution vi

```r
# Plot functions to check
xseq <- seq(-2, 2, length.out = 100)
plot(xseq, cos(xseq), type = "l")
lines(xseq, xseq^3)
abline(v = x_next, lty = 2)
```

Solution vii



Brent's method i

- The bisection method is guaranteed to converge.
    - Intermediate Value Theorem
- But convergence can be slow...
    - For an initial interval of length L, after n step the bracketing interval has length $L/2^n$.
- Other methods (e.g. secant method) can converge faster, but they're not guaranteed to converge...
- **Brent's method** combines the convergence speed of these methods, but guarantees convergence by keeping the root within a shrinking interval.

Brent's method ii

- I will give a general description the algorithm, but today we will use R's implementation.
  - **Next lecture**: you will implement it.

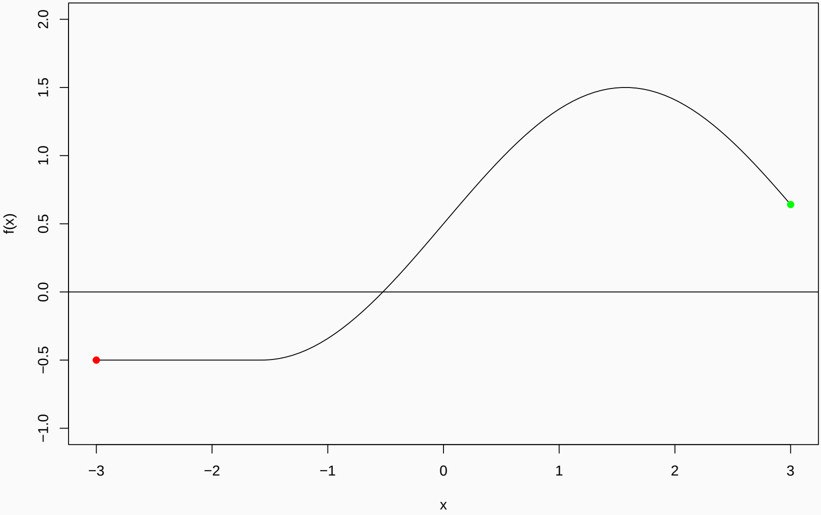
Brent's method iii

Algorithm

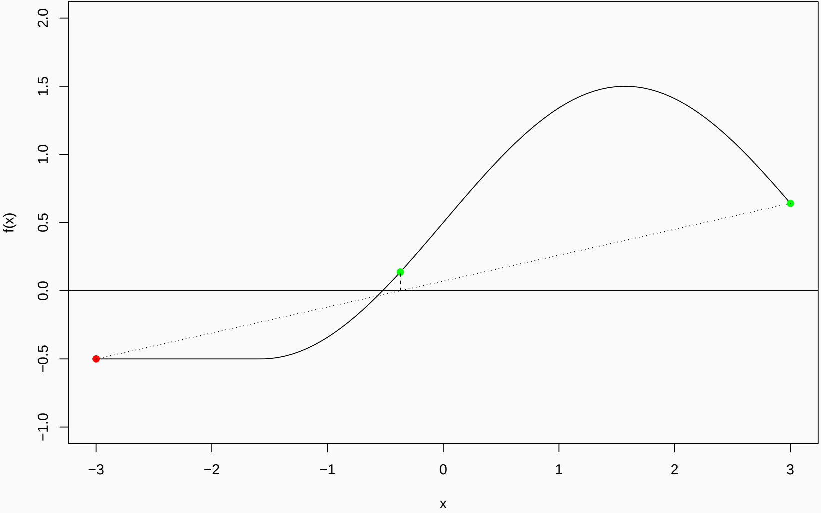
Start with interval $[a, b]$ and continuous function $f(x)$. The values $f(a), f(b)$ have opposite signs.

1. Define a third point $(c, f(c))$, where c is the value at which a linear interpolation crosses the x-axis. Depending on the sign of $f(c)$, we know the solution $f(x) = 0$ falls inside the interval (a, c) or (c, b).
2. Fit a sideways parabola to all three points, and find the intersection x_1 with the x-axis. If x_1 falls outside the interval from Step 1, replace x_1 by the midpoint of the interval (i.e. bisection).
3. Repeat until convergence.

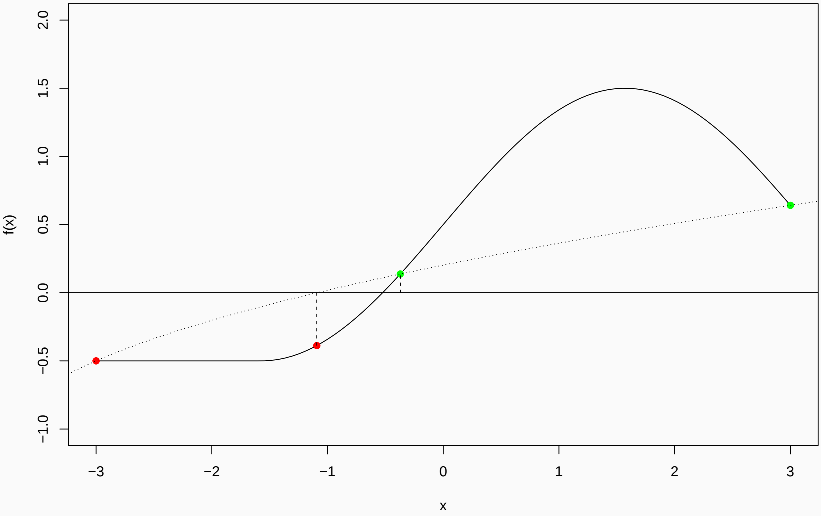
Demo i



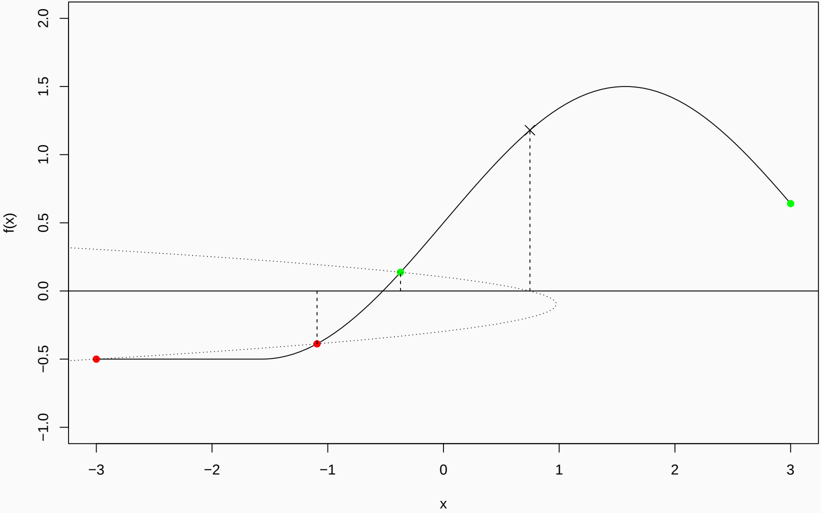
Demo ii



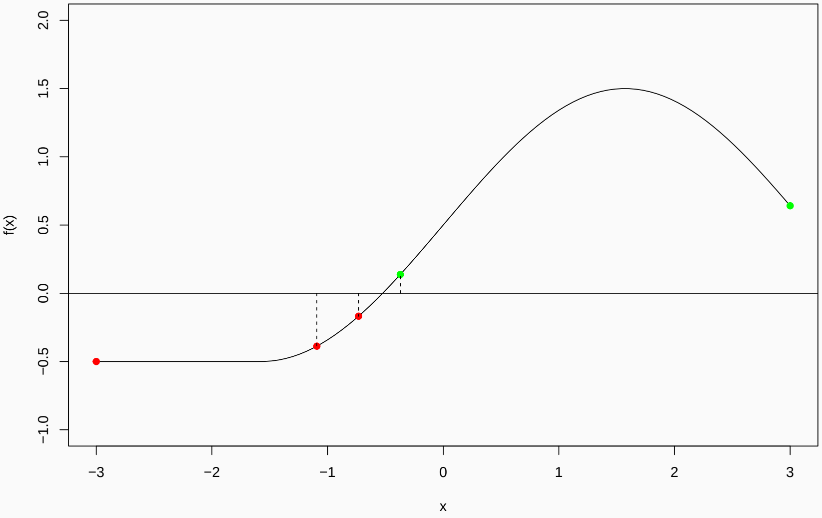
Demo iii



Demo iv



Demo v



Example i

- We will use the same example as above:

$$f(x) = a^2 + x^2 + \frac{2ax}{n-1} - (n-2),$$

for $a = 0.5$ and $n = 20$, on the interval $(0, 5n)$.

```
a <- 0.5
n <- 20
# Create a function
fun <- function(x) {
  a^2 + x^2 + 2*a*x/(n-1) - n + 2
}
```

Example ii

- We will use the function `uniroot` in R:
  - The first argument is the function $f(x)$.
  - The second argument is the interval $[a, b]$.
  - The argument `tol` controls the convergence.

```
output <- uniroot(f = fun,
                  interval = c(0, 5*n),
                  tol = 10^-10)
names(output)
```

```
## [1] "root" "f.root" "iter" "init.it"
"estim.prec"
```

Example iii

```r
output$root
```

```
## [1] 4.186841
```

```r
output$iter
```

```
## [1] 16
```

Exercise

Use Brent's method to find the root of

$$f(x) = e^{-x}\left(3.2\sin(x) - 0.5\cos(x)\right),$$

on the interval $[3, 4]$.

Solution

```r
result <- uniroot(function(x) {
  exp(-x)*(3.2*sin(x) - 0.5*cos(x))
  }, interval = c(3, 4))

result$root
```

```
## [1] 3.296589
```

Summary

- We discussed some important differences between computer arithmetic and "normal" arithmetic.
    - Rounding errors
    - Overflow and underflow
- We introduced two methods for finding roots $f(x) = 0$ in one-dimension.
    - Why can't we apply these methods in higher dimensions?
- On Thursday, we will see how this can be applied to **Maximum Likelihood Estimation**.