Post01 - Function in Numerical Analysis

Katherine Zhou

October 29, 2017

Introduction

In class, we have learnt about function and utilized it as a tool to perform various kinds of tasks, including simple algebraic and statistical calculations. As a student majoring in engineering, I am interested in using coding to solve mathematical problems. With this motivation, in this post, I want to combine function with basics of mathematics and broaden the application areas to the numerical analysis, with a main focus on root finding, the first order derivative approximation and the Riemann integration.

Applications

Part 1 - Three Most Common Root-Finding Methods

1. Brute-force

This method approximates roots on [a,b] by dividing [a,b] into n subintervals and finds the first x_i with f(x_i)*f(x_i+1)<=0. The root is calculated as the average between x_i and x_i+1.

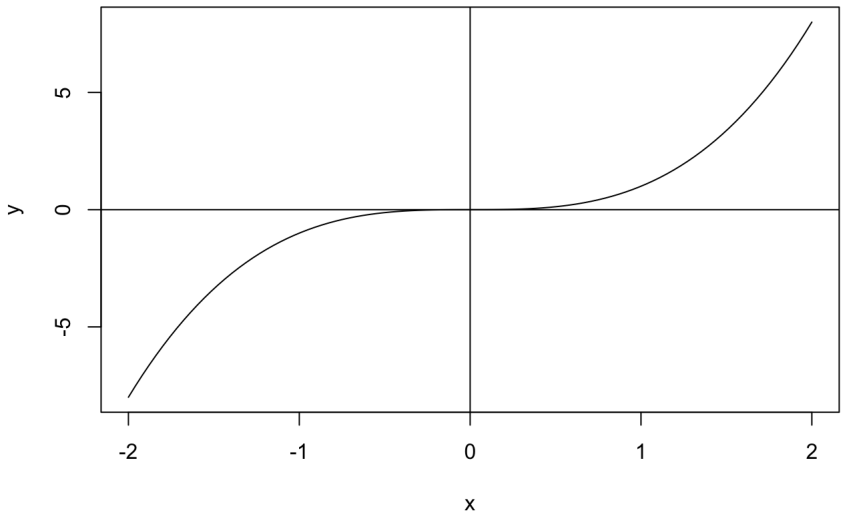
Requirements:

f is continuous over [a,b] with f(a)f(b)<=0.

```
# Define a function f that we want to find a solution x, which satisfies f(x)=0.
 f <- function(x){
    x^3
 }

# This function is continuous with f(-2)f(2)<=0.
# This can be shown from the plot of this function.
curve(x^3,from=-2,to=2,n=100,add=FALSE,ylab='y',main='Figure. 1     y vs. x')
abline(h=0,v=0)
```

Figure. 1 y vs. x




```
# Function Brute takes three inputs: a, b and n. n is the number of subintervals we want to choose.
Brute <- function(a,b,n) {
  interval <- (b-a)/n
  x <- seq(from=a,to=b,by=interval)
  i <- 1
  repeat{
     result <- f(x[i])*f(x[i+1])
     i <- i+1
     if(result <= 0) break
  }
   (f(x[i])+f(x[i-1]))/2
 }

# Testing different n values.
Brute(-2,2,10)
```

```
## [1] -0.032
```

```
Brute(-2,2,100)
```

```
## [1] -3.2e-05
```

```
Brute(-2,2,100000)
```

```
## [1] -3.2e-14
```

The larger n is, the more accurate our approximation is.

2. Bisection method

This method finds the root using an approach similar to the Brute-force. However, instead of testing x_i one by one, it changes range in each iteration and converges much faster than the previous method does.

Requirements:

f is continuous over [a,b] with f(a)f(b)<=0 The root found depends on the interval we choose. We could set a tolerance to limit the range of solutions we want to obtain.

```
# Three inputs.
Bisection <- function(a,b,tolerance){
  m <- (a+b)/2
while( abs(m) > tolerance ){
  if (f(a)*f(m)<=0){
  b <- m
 }else{
  a <- m
}
  m <- (a+b)/2
}
  m
}

Bisection(-3,2,0.01)
```

```
## [1] 0.0078125
```

```
Bisection(-3,2,0.001)
```

```
## [1] 0.0004882812
```

```
Bisection(-3,2,0.0001)
```

```
## [1] 3.051758e-05
```

We could enhance the result by lowering the tolerance based on our needs.

3. Newton-Raphson method

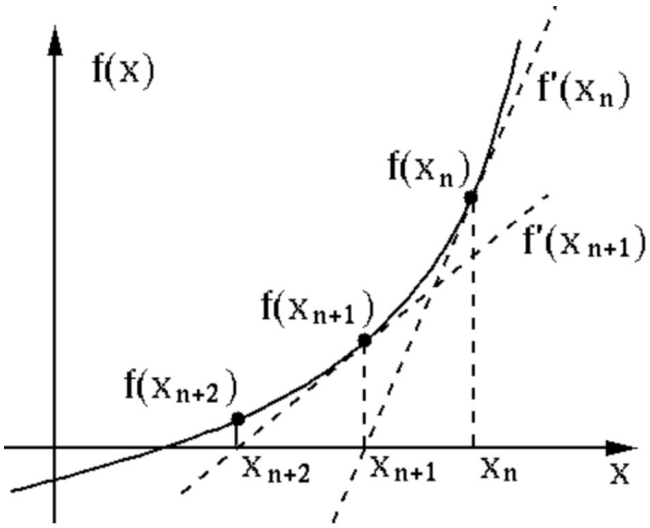
This method starts from an initial guess, x_n. The tangent line of f at x_n intersects with x axis, which gives us the next x, x_n+1. It is demonstrated in the figure below.

Requirements and limitations:

It converges to the solution very fast. However, it requires a good initial guess and may fail in some conditions. For example, when df(x_n) is approximately zero, the tangent is horizontal, which results in an infinite x_n+1 and the solution does not converge in this case.

```
library(knitr)
include_graphics('images/newton.png',auto_pdf = TRUE)
```

Figure. 2 Newton-Raphson Approximation




```
# Find the derivative df.
df <- function(x)3*x^2

# Two inputs: r_0, the initial guess, and tolerance.
Newton_Raphson <- function(r_0, tolerance){
  r<- r_0
  while (abs(r)>tolerance){
  r <- r-f(r)/df(r)
  }
  r
}

Newton_Raphson(2,0.001)
```

```
## [1] 0.000902186
```

```
Newton_Raphson(2,0.0001)
```

```
## [1] 7.920426e-05
```

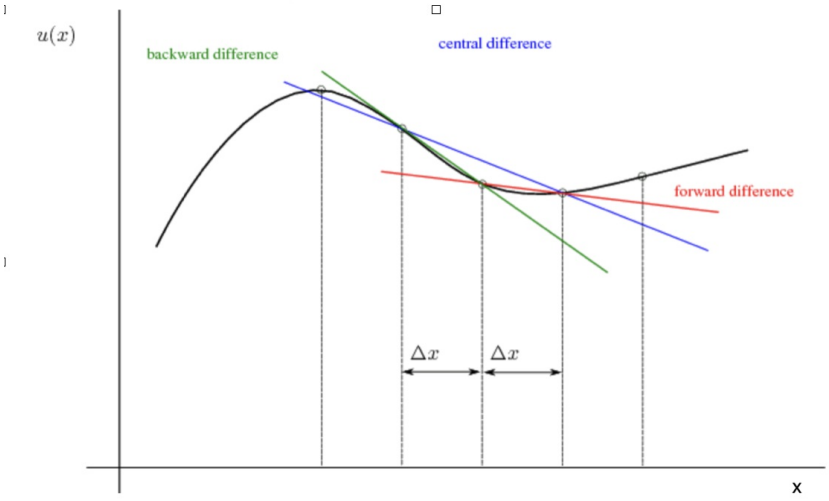
The solution converges very fast.

Part 2 - First Order Derivative Approximation

1. Forward finite
2. Backward finite
3. Central finite

```
include_graphics('images/Finite difference.png',auto_pdf = TRUE)
```

Figure. 3 Finite Difference Approximation




```
# This figure gives us a good demonstration on how the three methods work.
```

```
Derivative <- function(x,r){
forward <- (f(x[r+1])-f(x[r]))/(x[r+1]-x[r])
backward <- (f(x[r])-f(x[r-1]))/(x[r]-x[r-1])
central <- (f(x[r+1])-f(x[r-1]))/(x[r+1]-x[r-1])
c(forward,backward,central)
}

n <- 0.001
x <- seq(from=-2,to=2,by=n)
Derivative(x,3)
```

```
## [1] 11.97002 11.98201 11.97601
```

```
df(x[3])
```

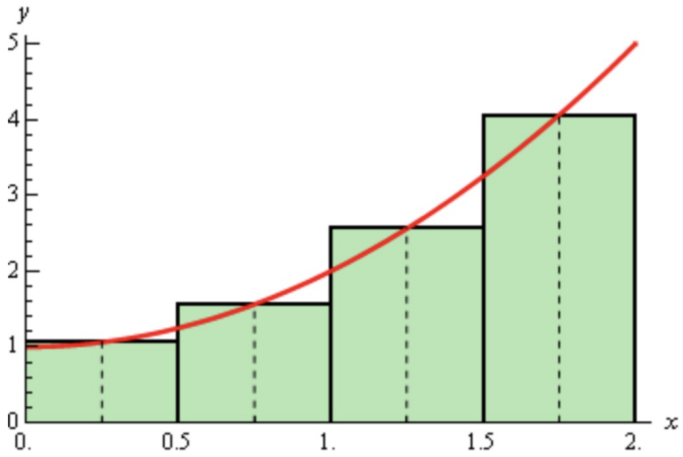
```
## [1] 11.97601
```

```
# Choose n value and compare the result with the true value.
# Notice that the central finite difference works the best, which, however, it's not necessarily true in other cas
es.
```

Part 3 - Midpoint Riemann Integral

```
include_graphics('images/Mid Riemann.png',auto_pdf = TRUE)
```

Figure. 4 Midpoint Riemann Integral




```
# This method approximates the area under the curve, the integral, using the f(midpoint) as the height of each sub
interval and the subinterval as the width.
# The figure below shows how it works.
```

```
Integral <- function(a,b,n){
    interval <- (b-a)/n
    x <- seq(from=a,to=b,by=interval)
    h <- rep(0,length(x))
  for (i in 1:(length(x)-1)){
    h[i] <- f((x[i]+x[i+1])/2)
    }
sum(h*interval)
}

Integral(0,3,100)
```

```
## [1] 20.24899
```

```
Integral(0,3,1000)
```

```
## [1] 20.24999
```

```
Integral(0,3,10000)
```

```
## [1] 20.25
```

The larger n is, the closer it is to the real integration value.

Conclusion

As the above examples show, function can combine with loops to perform numerical analysis. It could be tailored to needs and produce good approximations. In addition, by choosing methods wisely, one could maximize the calculation efficiency and save time.

Although R is relatively inefficient with large iteration loops compared to other programming languages, it is still worth a close look, since understanding basics could give us a solid knowledge foundation and prepare us for working on more challenging projects.

Reference

Bisection Method (https://en.wikipedia.org/wiki/Bisection_method)
Newton-Raphson Figure (http://fourier.eng.hmc.edu/e176/lectures/NM/node20.html)
Newton-Raphson Method (https://en.wikipedia.org/wiki/Newton%27s_method)
Finite-Difference (https://en.wikipedia.org/wiki/Finite_difference)
Finite-Difference Figure (http://www.iue.tuwien.ac.at/phd/heinzl/node27.html)
Riemann Integral (https://en.wikipedia.org/wiki/Riemann_integral)
Riemann Integral Figure (http://tutorial.math.lamar.edu/Classes/CalcI/AreaProblem.aspx)