7. Secant Method

Cheng Peng

Lecture Note for MAT325 Numerical Analysis

Contents

1 Introduction

Recall that Newton method uses Taylor expansion to derive the functional recursive relationship between adjacent approximated roots

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \text{ for } n = 0, 1, \cdots .$$

where $f'(x_n)$ is the slope of the tangent line passing through $x = x_n$. If we use the slope of a secant line that passes through points $(x_n, f(x_n))$ and $(x_{n-1}, f(x_{n-1}))$, we can use the x-coordinates of the intersection between the secant line and x-axis to approximate the root of $f(x) = 0$.

2 Secant Method

Assume we have two distinct initial values $x = x_0$ and $x = x_1$. Then slope of the secant line passing through A$(x_0, f(x_0))$ and B$(x_1, f(x_1))$ is

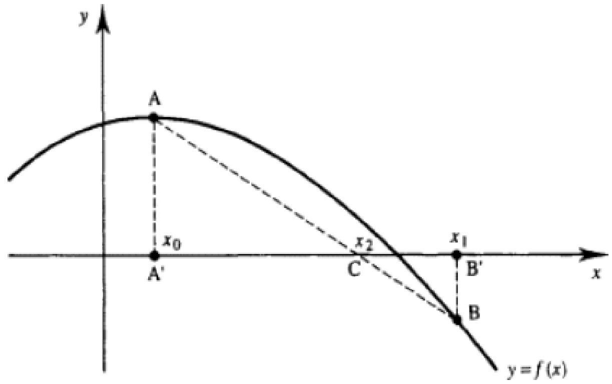
$$\frac{f(x_1) - f(x_0)}{x_1 - x_0} \approx f'(x_1) \text{ when } |x_1 - x_0| \text{ is small.}$$

The secant method uses the x-coordinate of the intersection of the secant line

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

and $f(x) = 0$ (equation of the x-axis). Solving for x, we have

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



In general, the recursive relationship between approximated roots of the **secant method** is given by

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n), \text{ for } n = 0, 1, \cdots$$

https://github.com/pengdsci/MAT325/raw/main/w04/img/w04-betterSecantAnimation.gif

- **Secant Algorithm**

We develop the following pseudo-code of the secant method.

```
INPUT:  f(x)            (satisfying f(x) = 0)
        x0              (initial value 1)
        x1              (initial value 2)

STEP 1: x0
        x1              (f(x0)*f(x1) must be negative)
        M = 200
        TOL = 10^(-6)
        n = 0
        ERR = |x1 - x0|
STEP 2: WHILE ERR > TOL DO
           n = n + 1
           new.x = x1 - ((x1-x0)/(f(x1)-f(x0)))*f(x1)
           ERR = |new.x - x1|
           IF ERR < TOL DO:
              OUTPUT        (results and optional relevant info)
              STOP
           ENDIF
           IF ERR >= TOL DO:
              OUTPUT        (message or intermediate outputs)
              x1 = new.x    (update)
              x0 = x1
           ENDIF
           IF n == M DO:
             OUTPUT         (warning messages)
             STOP
           ENDIF
        ENDWHILE
```

- **Implementation with R**

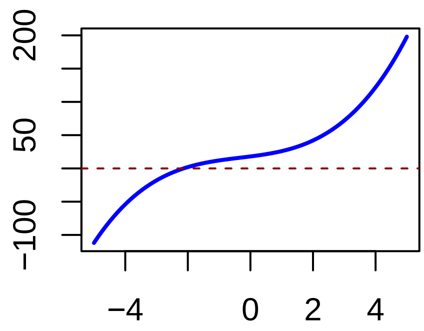
We next write an R function to implement the secant method.

```r
#########################################
##     Root Finding: Secant Method
##########################################
Secant = function(fn,                    # input function
                  TOL,                   # error tolerance
                  max.iter = 100,        # max allowed iterations
                  x1,                    # initial value #1
                  x2,                    # initial value #2
                  detail=TRUE            # output intermediate output
                  ){
  ctr = 0      # counter of iteration
  ERR = abs(x2 - x1)
  while(ERR > TOL){
      ctr = ctr + 1
      new.x = x2 - fn(x2) * (x2 - x1) / (fn(x2) - fn(x1))
      ERR = abs(new.x - x2)
      if(ERR < TOL){
         cat("\n\nThe algorithm converges!")
         cat("\nThe approximated root is", new.x,".")
         cat("\nThe absolute relative error ERR=",ERR,".")
         cat("\nThe number of iterations:", ctr, ".")
         break
         } else{
            if(detail == TRUE){
               cat("\nIteration:", ctr,", approximated root:", new.x,", absolute relative error:", ERR,
            }
            # updating the two values. CAUTION: order matters
            x1 = x2
            x2 = new.x
         }
      if(ctr == max.iter){
         cat("\n\nThe maximum number of iterations attained!")
         cat("\nThe algorithm did not converge!")
         break
       }
   } # close the while-loop
}    # close the function environment
```

Example 1: Find a root of equation $x^3 + x^2 + 6x + 18 = 0$.

Solution: we use the above R function of Newton method to find the approximated root of the equation.

```r
# define the function f(x) that satisfies f(x) = 0
test_func = function(x){x^3+x^2+6*x+18 }
###
xx = seq(-5,5, length=500)
yy = test_func(xx)
plot(xx, yy, type = "l", xlab ="", ylab="", main="", lwd = 2, col = "blue")
abline(h=0, col = "darkred", lty = 2)
```




```
##
# call the function
Secant(fn = test_func,          # input function
       TOL = 10^(-8),           # error tolerance
       max.iter = 100,          # max allowed iterations
       x1=1,                    # initial value #1
       x2=2,                    # initial value #2
       detail=TRUE              # output intermediate output
       )
```

```
##
## Iteration: 1 , approximated root: -0.625 , absolute relative error: 2.625 .
## Iteration: 2 , approximated root: -1.994056 , absolute relative error: 1.369056 .
## Iteration: 3 , approximated root: -2.225656 , absolute relative error: 0.2315996 .
## Iteration: 4 , approximated root: -2.131567 , absolute relative error: 0.09408908 .
## Iteration: 5 , approximated root: -2.135926 , absolute relative error: 0.004358849 .
## Iteration: 6 , approximated root: -2.136065 , absolute relative error: 0.0001395923 .
## Iteration: 7 , approximated root: -2.136065 , absolute relative error: 2.201139e-07 .
##
## The algorithm converges!
## The approximated root is -2.136065 .
## The absolute relative error ERR= 1.076161e-11 .
## The number of iterations: 8 .
```

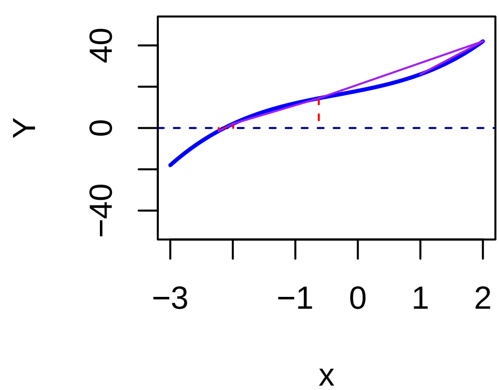
- **Modified Code with Graphic Information**

```
########################################
##     Root Finding: Secant Method
#########################################
SecantMethod = function(fn,                      # input function
                        TOL,                     # error tolerance
                        max.iter = 100,          # max allowed iterations
                        x1,                      # initial value #1
                        x2,                      # initial value #2
```

```
                        detail = TRUE,        # output intermediate output
                        ## adding controls for the graphic
                        graphic = TRUE,        # default
                        xlimit,                # x-limits: a vector of lower and upper limits
                        ylimit,                # y-limits: a vector of lower and upper limits
                        ...                    # other optional arguments if any
                  ){
  ctr = 0      # counter of iteration
  ERR = abs(x2 - x1)
  ## base graph
  if(graphic ==TRUE){
     xx = seq(xlimit[1], xlimit[2], length = 500) # x-coordinates
     yy = fn(xx)                                  # y-coordinates
     plot(xx, yy, type = "l",
                  xlab = "x",
                  ylab = "Y",
                  xlim = xlimit,
                  ylim = ylimit,
                  lwd = 2,
                  col = "blue",
                  lty = 1)
    }
    abline(h = 0, lty = 2, col = "navy")
  ##
  while(ERR > TOL){
      ctr = ctr + 1
      new.x = x2 - fn(x2) * (x2 - x1) / (fn(x2) - fn(x1))
      ERR = abs(new.x - x2)
      if(ERR < TOL){
         cat("\n\nThe algorithm converges!")
         cat("\nThe approximated root is", new.x,".")
         cat("\nThe absolute relative error ERR=",ERR,".")
         cat("\nThe number of iterations:", ctr, ".")
         break
         } else{
            if(graphic == TRUE){
              segments(x2, fn(x2), x1, fn(x1), lty = 1, col = "purple")
              segments(new.x, fn(new.x), new.x, 0, col = "red", lty = 2)
            }
            if(detail == TRUE){
               cat("\nIteration:", ctr,", approximated root:", new.x,", absolute relative error:", ERR,
             }
            # updating the two values. CAUTION: order matters
            x1 = x2
            x2 = new.x
         }
      if(ctr == max.iter){
         cat("\n\nThe maximum number of iterations attained!")
         cat("\nThe algorithm did not converge!")
         break
       }
   } # close the while-loop
}    # close the function environment
```

Example 1 Revisited:

```
# define the function f(x) that satisfies f(x) = 0
test_func = function(x){x^3+x^2+6*x+18 }
###
# call the function
SecantMethod(fn = test_func,       # input function
       TOL = 10^(-8),        # error tolerance
       max.iter = 100,       # max allowed iterations
       x1=1,                 # initial value #1
       x2=2,                 # initial value #2
       detail=TRUE,           # output intermediate output
       ## adding controls for the graphic
       graphic = TRUE,        # default
       xlimit=c(-3,2),                # x-limits: a vector of lower and upper limits
       ylimit = c(-50, 50)                 # y-limits: a vector of lower and upper limits
       )
```

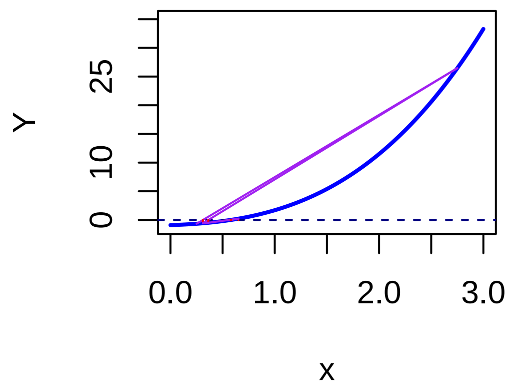



```
##
## Iteration: 1 , approximated root: -0.625 , absolute relative error: 2.625 .
## Iteration: 2 , approximated root: -1.994056 , absolute relative error: 1.369056 .
## Iteration: 3 , approximated root: -2.225656 , absolute relative error: 0.2315996 .
## Iteration: 4 , approximated root: -2.131567 , absolute relative error: 0.09408908 .
## Iteration: 5 , approximated root: -2.135926 , absolute relative error: 0.004358849 .
## Iteration: 6 , approximated root: -2.136065 , absolute relative error: 0.0001395923 .
## Iteration: 7 , approximated root: -2.136065 , absolute relative error: 2.201139e-07 .
##
## The algorithm converges!
## The approximated root is -2.136065 .
## The absolute relative error ERR= 1.076161e-11 .
## The number of iterations: 8 .
```

Example 3: find the solution to $0.8(x + 0.5)^3 - 1 = 0$ 0n $[0.25, 2.75]$.

```
# define the function f(x) that satisfies f(x) = 0
test_func = function(x){0.8*(x+0.5)^3-1}
###
```

```r
# call the function
SecantMethod(fn = test_func,       # input function
       TOL = 10^(-8),        # error tolerance
       max.iter = 100,       # max allowed iterations
       x1=0.25,                 # initial value #1
       x2=2.75,                 # initial value #2
       detail=TRUE,           # output intermediate output
       ## adding controls for the graphic
       graphic = TRUE,        # default
       xlimit=c(0,3),                # x-limits: a vector of lower and upper limits
       ylimit = c(-1, 35)                 # y-limits: a vector of lower and upper limits
       )
```




```
##
## Iteration: 1 , approximated root: 0.3110599 , absolute relative error: 2.43894 .
## Iteration: 2 , approximated root: 0.3627672 , absolute relative error: 0.05170731 .
## Iteration: 3 , approximated root: 0.651921 , absolute relative error: 0.2891538 .
## Iteration: 4 , approximated root: 0.5610572 , absolute relative error: 0.09086383 .
## Iteration: 5 , approximated root: 0.5761365 , absolute relative error: 0.01507935 .
## Iteration: 6 , approximated root: 0.5772337 , absolute relative error: 0.0010972 .
## Iteration: 7 , approximated root: 0.5772173 , absolute relative error: 1.640501e-05 .
## Iteration: 8 , approximated root: 0.5772173 , absolute relative error: 1.645382e-08 .
##
## The algorithm converges!
## The approximated root is 0.5772173 .
## The absolute relative error ERR= 2.502443e-13 .
## The number of iterations: 9 .
```

3 Error Analysis

Let $e_n = x_n - p$, then $e_n - e_{n-1} = x_n - x_{n-1}$. From the definition of the secant method we have

$$e_{n+1} = e_n + x_{n+1} - x_n = e_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n).$$

With some algebraic manipulation, we can express e_{n+1} as

$$e_{n+1} = \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \frac{f(x_n)/e_n - f(x_{n-1})/e_{n-1}}{x_n - x_{n-1}} e_n e_{n-1}.$$

Note that

$$\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \approx \frac{1}{f'(p)}$$

After expanding $f(x_n)$ and $f(x_{n-1})$ at p, we have

$$\frac{f(x_n)/e_n - f(x_{n-1})/e_{n-1}}{x_n - x_{n-1}} \approx \frac{f''(p)}{2}$$

Therefore,

$$e_{n+1} \approx \frac{f''(p)}{2f'(p)} e_n e_{n-1}$$

Consequently,

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n e_{n-1}} = \frac{f''(p)}{2f'(p)} = C_0$$

To find the order of convergence, we assume that $e_{n+1} = C_n e_n^\alpha$ where $\lim_{n\to\infty} C_n = C$. Then

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n e_{n-1}} = \lim_{n\to\infty} \frac{C[Ce_{n-1}^\alpha]^\alpha}{Ce_{n-1}^\alpha e_{n-1}} = \lim_{n\to\infty} C^\alpha e_{n-1}^{\alpha^2-\alpha-1} = C_0.$$

This implies that

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

The positive root of the above equation is

$$\alpha = \frac{1+\sqrt{5}}{2} \approx 1.62$$

Therefore, the convergence order for the secant method is between linear and quadratic orders – we call this **super-linear** convergence!