# 2. Making Publication-Ready Outputs

Cheng Peng

Lecture Note for MAT325 Numerical Analysis

## Contents



## 1 Introduction

This note focuses on how to create publication-ready tables and graphics. We use Newton's method as an illustrative example.

## 2 R Function for Newton Method

Recall the R code developed in the lecture note.

```
# Define f(x) and f'(x)

fn = function(x) x^3 - x +3
dfn = function(x) 3*x^2 - 1

# initial values
n = 0
x = -1
M = 200
TOL = 10^(-6)
ERR = abs(fn(x)/dfn(x))
# loop begins
while(ERR > TOL){
  n = n + 1
  x = x - fn(x)/dfn(x)
  ERR = abs(fn(x)/dfn(x))
  if(ERR < TOL){
```

```
     cat("\n\nAlgorithm converges!")
     cat("\nThe approximated root:", x, ".")
     cat("\nThe absolute error:", ERR, ".")
     cat("\nThe number of iterations n =",n,".")
     break
    } else{
      cat("\nIteration n =",n, ", approximate root:",x,", absolute error:", ERR,".")
    }
    if (n ==M){
      cat("\n\nThe maximum iterations attained!")
      cat("\nThe algorithm did not converge!")
      break
    }
}
```

```
##
## Iteration n = 1 , approximate root: -2.5 , absolute error: 0.5704225 .
## Iteration n = 2 , approximate root: -1.929577 , absolute error: 0.2217111 .
## Iteration n = 3 , approximate root: -1.707866 , absolute error: 0.03530793 .
## Iteration n = 4 , approximate root: -1.672558 , absolute error: 0.0008580914 .
##
## Algorithm converges!
## The approximated root: -1.6717 .
## The absolute error: 5.002863e-07 .
## The number of iterations n = 5 .
```

Next, we simply wrap up the above code and write a function.

```
NewtonVersion01 = function(fn,     # function used to define the equation
                           dfn,    # derivative of f(x)
                           x,      # initial value
                           M,      # pre-set maximum iteration
                           TOL,    # error tolerance
                           ...
    ){
    n = 0
    ERR = abs(fn(x)/dfn(x))
    # loop begins
    while(ERR > TOL){
    n = n + 1
    x = x - fn(x)/dfn(x)
    ERR = abs(fn(x)/dfn(x))
    if(ERR < TOL){
       cat("\n\nAlgorithm converges!")
       cat("\nThe approximated root:", x, ".")
       cat("\nThe absolute error:", ERR, ".")
       cat("\nThe number of iterations n =",n,".")
       break
     } else{
       cat("\nIteration n =",n, ", approximate root:",x,", absolute error:", ERR,".")
     }
     if (n ==M){
        cat("\n\nThe maximum iterations attained!")
        cat("\nThe algorithm did not converge!")
        break
```

```
    }
  }
}
```

**Example 1**: Solve $f(x) = x^3 - x + 3 = 0$. Note that $f'(x) = 3x^2 - 1$.

```
##
fn = function(x) x^3 - x + 3
dfn = function(x) 3*x^2 - 1
##
NewtonVersion01(fn = fn,        # function used to define the equation
                dfn = dfn,      # derivative of f(x)
                x = -1,         # initial value
                M = 200,        # pre-set maximum iteration
                TOL = 10^(-6)   # error tolerance
                )
```

```
##
## Iteration n = 1 , approximate root: -2.5 , absolute error: 0.5704225 .
## Iteration n = 2 , approximate root: -1.929577 , absolute error: 0.2217111 .
## Iteration n = 3 , approximate root: -1.707866 , absolute error: 0.03530793 .
## Iteration n = 4 , approximate root: -1.672558 , absolute error: 0.0008580914 .
##
## Algorithm converges!
## The approximated root: -1.6717 .
## The absolute error: 5.002863e-07 .
## The number of iterations n = 5 .
```

# 3 Output Tables

```
NewtonVersion02 = function(fn,          # function used to define the equation
                           dfn,         # derivative of f(x)
                           x,           # initial value
                           M,           # pre-set maximum iteration
                           TOL,         # error tolerance
                           out.table,   # intermediate output table
                           ...
    ){
    n = 0
    ERR = abs(fn(x)/dfn(x))
    # Result table
    Result = NULL
    # Intermediate Table
    Intermediate.output = data.frame(Iteration = 1:M,
                                     Estimated.root = rep(NA,M),
                                     Absolute.error = rep(NA,M))
    # loop begins
    while(ERR > TOL){
    n = n + 1
    x = x - fn(x)/dfn(x)
    ERR = abs(fn(x)/dfn(x))
    if(ERR < TOL){
       Intermediate.output[n, ] = c(n, x, ERR)
       #Result =c(Total.Iteration = n, Estimated.Root = x, Absolute.Error = ERR)
```

```
      break
    } else{
      Intermediate.output[n, ] = c(n, x, ERR)   # store intermediate outputs
    }
    if (n ==M){
       cat("\n\nThe maximum iterations attained!")
       cat("\nThe algorithm did not converge!")
       break
    }
   }
   # out of the loop
   Intermediate.output = na.omit(Intermediate.output)
   if (out.table == TRUE){
     pander(Intermediate.output)
   }
    #pander(Result)
}
```

```
##
fn = function(x) x^3 - x + 3
dfn = function(x) 3*x^2 - 1
##
NewtonVersion02(fn = fn,        # function used to define the equation
                dfn = dfn,      # derivative of f(x)
                x = -1,         # initial value
                M = 200,        # pre-set maximum iteration
                TOL = 10^(-6),  # error tolerance
                out.table = TRUE
                )
```

| Iteration | Estimated.root | Absolute.error |
|---|---|---|
| 1 | -2.5 | 0.5704 |
| 2 | -1.93 | 0.2217 |
| 3 | -1.708 | 0.03531 |
| 4 | -1.673 | 0.0008581 |
| 5 | -1.672 | 5.003e-07 |

# 4 Creating Subtasks - Computational Thinking

Ideally, we would like to have a function to output more than one object so that we can see tables, graphics, etc. Unfortunately, one R function can output a single object. However, we can use a single object such as vectors, matrices, and data frames, and then use the output object to create other objects such as tables and figures.

One clean way for obtaining different pieces of information in coding is to make several tidy functions/routines and call these functions to get the desired information.

For all root-finding methods, all desired summarized information is based on iterations, estimated roots, and errors in each step.

The following **Newton.Method()** returns a data frame that contains information about *number of iterations, estimated roots, and absolute errors*. We also write two graphic functions to extract information from the Newton method.

```
Newton.Method = function(fn,          # function used to define the equation
                         dfn,         # derivative of f(x)
                         x,           # initial value
                         M,           # pre-set maximum iteration
                         TOL,         # error tolerance
                         ...){
    n = 0
    ERR = abs(fn(x)/dfn(x))
    # Result table
    Result = NULL
    # Intermediate Table
    Intermediate.output = data.frame(Iteration = 1:M,
                                     Estimated.root = rep(NA,M),
                                     Absolute.error = rep(NA,M))
    # loop begins
    while(ERR > TOL){
    n = n + 1
    x = x - fn(x)/dfn(x)
    ERR = abs(fn(x)/dfn(x))
    if(ERR < TOL){
       Intermediate.output[n, ] = c(n, x, ERR)
       #Result =c(Total.Iteration = n, Estimated.Root = x, Absolute.Error = ERR)
       break
     } else{
       Intermediate.output[n, ] = c(n, x, ERR)   # store intermediate outputs
     }
     if (n ==M){
        cat("\n\nThe maximum iterations attained!")
        cat("\nThe algorithm did not converge!")
        break
     }
    }
    # out of the loop
    Intermediate.output = na.omit(Intermediate.output)
    Intermediate.output
}
```

It is always a good idea to write a function to draw the curve of a given function on a given interval $[a, b]$.

```
curve.fun = function(fn, a, b){
  xx = seq(a,b, length = 200)   # 200 x-values evenly spread over [a,b]
  yy = fn(xx)                   # corresponding y values
  plot(xx, yy, type = "l", lwd = 2, col = "blue",
               xlab = "",  ylab = "",  main = "")
  abline(h = 0, lty = 1, col = "red")
}
```

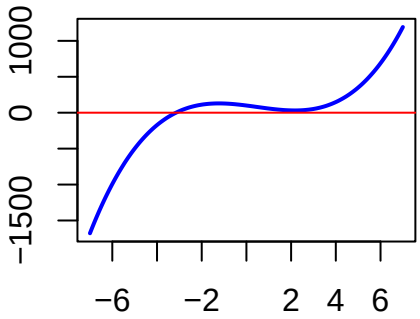
```
ErrorPlot = function(errorMatrix){
     niter = dim(errorMatrix)[1]  # total number of iterations
     iteration = 0:(niter + 1)
     abs.error = errorMatrix[,3]
```

```
    plot(1:niter, abs.error, type = "b", lwd = 2, col = "blue", pch = 16,
                             xlab = "Number of Iteration",
                             ylab = "Absolute Error",
                             xlim = c(0,niter+1),
                             ylim = c(0, max(abs.error)),
                             main = "Error Pattern")
}
```

**Example 2**: draw the curve of $f(x) = 5x^3 - 7x^2 - 40x + 100$ over interval $[-7, 7]$

**Solution**: We simply call the above function to curve the given function in the following.

```
fn = function(x) 5 * x^3 - 7 * x^2 - 40 * x + 100
curve.fun(fn=fn, a = -7, b = 7)
```




The derivative of $f(x)$ is

$$f'(x) = 15x^2 - 14x - 40$$

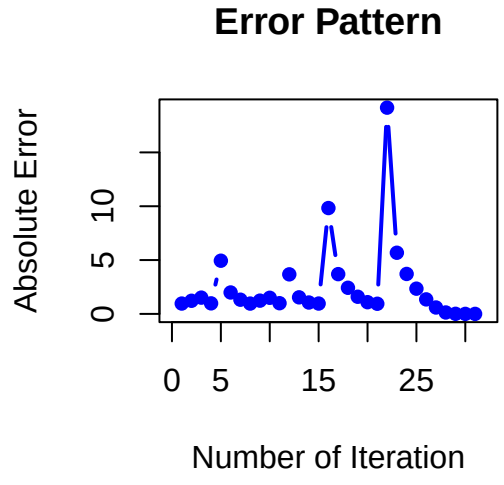
From the above figure, we can see that one of the roots is in $[-7, 7]$. We choose the initial value $x = 6$

```
##
fn = function(x) 5*x^3 - 7*x^2 - 40*x + 100
dfn = function(x) 15*x^2 - 14*x - 40
##
Intermediate.output = Newton.Method(fn = fn, dfn = dfn,  x = 5, M = 200, TOL = 10^(-6))
pander(Intermediate.output)
```

| Iteration | Estimated.root | Absolute.error |
|---|---|---|
| 1 | 3.679 | 0.9602 |
| 2 | 2.719 | 1.218 |
| 3 | 1.501 | 1.51 |
| 4 | 3.011 | 0.9769 |

| Iteration | Estimated.root | Absolute.error |
|---|---|---|
| 5 | 2.034 | 4.939 |
| 6 | 6.972 | 1.987 |
| 7 | 4.985 | 1.316 |
| 8 | 3.669 | 0.9585 |
| 9 | 2.711 | 1.231 |
| 10 | 1.48 | 1.496 |
| 11 | 2.975 | 0.9918 |
| 12 | 1.984 | 3.673 |
| 13 | 5.656 | 1.538 |
| 14 | 4.119 | 1.058 |
| 15 | 3.061 | 0.9599 |
| 16 | 2.101 | 9.831 |
| 17 | 11.93 | 3.692 |
| 18 | 8.24 | 2.424 |
| 19 | 5.816 | 1.591 |
| 20 | 4.224 | 1.086 |
| 21 | 3.138 | 0.9415 |
| 22 | 2.197 | 19.16 |
| 23 | -16.96 | 5.679 |
| 24 | -11.28 | 3.709 |
| 25 | -7.571 | 2.342 |
| 26 | -5.229 | 1.347 |
| 27 | -3.882 | 0.5938 |
| 28 | -3.288 | 0.1305 |
| 29 | -3.158 | 0.006164 |
| 30 | -3.152 | 1.348e-05 |
| 31 | -3.152 | 6.443e-11 |

```
ErrorPlot(errorMatrix = Intermediate.output)
```



# 5 The source() Function

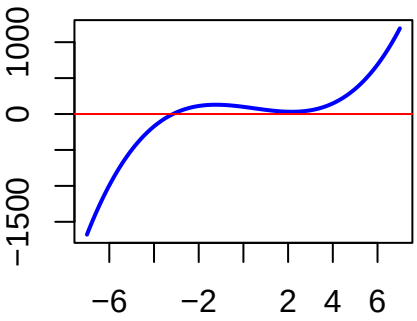
We can also put all R functions in a single file and store that file in web space. The function of the file can be called directly from R using function **source()**.

```
source("https://raw.githubusercontent.com/pengdsci/MAT325/main/w04/NewtonMethodCode.txt")
```

Note that the names of the three functions are different from the one used in **Example 2**! We redo Example 2 using the functions in https://raw.githubusercontent.com/pengdsci/MAT325/main/w04/NewtonMethodCode.txt

**Example 3**: draw the curve of $f(x) = 5x^3 - 7x^2 - 40x + 100$ over interval $[-7, 7]$

**Solution**: We simply call the above function to curve the given function in the following.



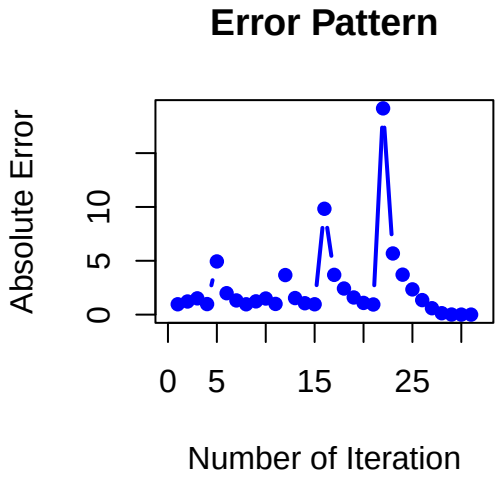

The derivative of $f(x)$ is

$$f'(x) = 15x^2 - 14x - 40$$

From the above figure, we can see that one of the roots is in $[-7, 7]$. We choose the initial value $x = 6$

| Iteration | Estimated.root | Absolute.error |
|---|---|---|
| 1 | 3.679 | 0.9602 |
| 2 | 2.719 | 1.218 |
| 3 | 1.501 | 1.51 |
| 4 | 3.011 | 0.9769 |
| 5 | 2.034 | 4.939 |
| 6 | 6.972 | 1.987 |
| 7 | 4.985 | 1.316 |
| 8 | 3.669 | 0.9585 |
| 9 | 2.711 | 1.231 |
| 10 | 1.48 | 1.496 |
| 11 | 2.975 | 0.9918 |
| 12 | 1.984 | 3.673 |

| Iteration | Estimated.root | Absolute.error |
|---|---|---|
| 13 | 5.656 | 1.538 |
| 14 | 4.119 | 1.058 |
| 15 | 3.061 | 0.9599 |
| 16 | 2.101 | 9.831 |
| 17 | 11.93 | 3.692 |
| 18 | 8.24 | 2.424 |
| 19 | 5.816 | 1.591 |
| 20 | 4.224 | 1.086 |
| 21 | 3.138 | 0.9415 |
| 22 | 2.197 | 19.16 |
| 23 | -16.96 | 5.679 |
| 24 | -11.28 | 3.709 |
| 25 | -7.571 | 2.342 |
| 26 | -5.229 | 1.347 |
| 27 | -3.882 | 0.5938 |
| 28 | -3.288 | 0.1305 |
| 29 | -3.158 | 0.006164 |
| 30 | -3.152 | 1.348e-05 |
| 31 | -3.152 | 6.443e-11 |




## 6 This Week's Lab Assignment

Use the fixed-point code in the lecture note as the baseline to create three functions (with detailed comments on your code) and solve $f(x) = 5x^3 - 7x^2 - 40x + 100 = 0$. Please submit a PDF version of your completed work that includes both code and outputs.