STATS 102A Homework 5

Winter 2022

Your Name
Your UID

Due 03/17/2022

General Notes

1. For the main functions (written in bold letters), please provide their flowcharts or algorithms that are sufficiently complete, clear, and concise enough to enable a person to accurately implement the function in any programming language they are adept with using.
2. Write the functions which accurately implements the algorithms as described or requested.
3. Include the error-handling to ensure your functions work properly.

Sourcing the Functions

```
source("205615894_stats102a_hw5.R")
```

1 Root-Finding Problem

1.1 Bisection Method

Write R code using the Bisection method to solve the following equations within an error of 10^{-8}. For all three cases, please state your starting interval and the estimated number of iterations required.

(a) $x^3 + 23 = 0$

```
function1 <- function(x){
  x^3 + 23
}
tol = 1e-8
x_l = -3; x_r = 0
f_l = function1(x_l); f_r = function1(x_r)
it = 0
while(x_r - x_l > tol){
  x_m <- (x_l + x_r) / 2
  f_m <- function1(x_m)
  if (identical(all.equal(f_m, 0), TRUE)) break
  if (f_l * f_m < 0){
    x_r <- x_m
  }else{
    x_l <- x_m
  }

it <- it + 1

}
cat("Root = ", x_m, ", ",
"iterations = ", it, "\n", sep="")
```

```
## Root = -2.843867, iterations = 29
```

Starting interval is (-3,0) and the estimated number of iterations required is 29.

(b) $x^x - 18 = 0$

```
function2 <- function(x){
  x^x - 18
}
tol = 1e-8
x_l = 1; x_r = 3
f_l = function2(x_l); f_r = function2(x_r)
it = 0
while(x_r - x_l > tol){
  x_m <- (x_l + x_r) / 2
  f_m <- function2(x_m)
  if (identical(all.equal(f_m, 0), TRUE)) break
  if (f_l * f_m < 0){
    x_r <- x_m
  }else{
    x_l <- x_m
  }

it <- it + 1

}
cat("Root = ", x_m, ", ",
"iterations = ", it, "\n", sep="")
```

```
## Root = 2.803663, iterations = 28
```

Starting interval is (1,3) and the estimated number of iterations required is 28.

(c) $e^{-x^2} - \frac{1}{10} = 0$

```
function3 <- function(x){
  exp(-(x^2)) - (1 / 10)
}
tol = 1e-8
x_l = -2; x_r = -1
```

```
f_l = function3(x_l); f_r = function3(x_r)
it = 0
while(x_r - x_l > tol){
  x_m <- (x_l + x_r) / 2
  f_m <- function3(x_m)
  if (identical(all.equal(f_m, 0), TRUE)) break
  if (f_l * f_m < 0){
    x_r <- x_m
  }else{
    x_l <- x_m
  }

it <- it + 1

}
cat("Root = ", x_m, ", ",
"iterations = ", it, "\n", sep="")
```

```
## Root = -1.517427, iterations = 22
```

Starting interval is (-2, -1) and the estimated number of iterations required is 22.

1.2 Fixed Point Iteration Method

Write R code using the Fixed point iteration method to solve the equation (b). Explain your algorithm prior to the code. Note: You may answer and write code for Question 1.1 and 1.2 directly in your .Rmd file.

(b) $x^x - 18 = 0$

We initially want to find g(x) such that g is defined as the solution to the equation g(x) = x. Thus, in this example, g(x) will equal log(18) / log(x). We start with x0 = 3 and this value goes into the g function. The value that goes into the g function changes based on the iteration and iterations are done until the difference between the two iterations are less than the tolerence level.

```
g <- function(x) log(18) / log(x)
it <- 0
tol <- 1e-8
N = 1e4
# initial guess
x_0 <- 3
x_1 <- g(x_0)
while((abs(x_1 - x_0) > tol) & (it <= N)){
  x_0 <- x_1
  x_1 <- g(x_0)
  it <- it + 1
}
cat("Root = ", x_1, ", ",
"iterations = ", it, "\n", sep="")
```

```
## Root = 2.803663, iterations = 573
```

1.3 Newton's Method

Use the Newton's method to compute the square root by using only simple arithmetic operations.

(a) Write a function, **`get_sqrt()`**, that computes the square root of an arbitrary positive number. The function should take five arguments, where the last three have default values:

- `a`, a non-negative number to compute the square root of,
- `x0`, the initial guess,
- `tol` determines when you consider two iterates to be equal,
- `iter_max`, gives the maximum number of iterations, and
- `verbose`, determines if you want to display intermediate results.

In .R code

```
knitr::include_graphics("get_sqrt.png")
```

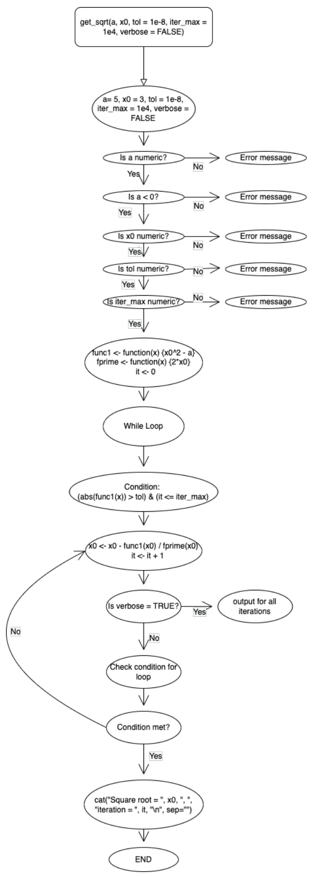

get_sqrt(a, x0, tol = 1e-8, iter_max = 1e4, verbose = FALSE)
a= 5, x0 = 3, tol = 1e-8, iter_max = 1e4, verbose = FALSE
Is a numeric?
No
Error message
Yes
Is a < 0?
No
Error message
Yes
Is x0 numeric?
No
Error message
Yes
Is tol numeric?
No
Error message
Yes
Is iter_max numeric?
No
Error message
Yes
func1 <- function(x) {x0^2 - a}
fprime <- function(x) {2*x0}
it <- 0
While Loop
Condition:
(abs(func1(x)) > tol) & (it <= iter_max)
x0 <- x0 - func1(x0) / fprime(x0)
it <- it + 1
Is verbose = TRUE?
Yes
output for all iterations
No
Check condition for loop
Condition met?
No
Yes
cat("Square root = ", x0, ", ", "iteration = ", it, "\n", sep="")
END

Figure 1: General stucture of a flowchart.

(b) Give an example of your function by computing the square root of 5.

```
get_sqrt(a= 5, x0 = 3, tol = 1e-8, iter_max = 1e4, verbose = FALSE)
```

```
## Square root = 2.236068, iteration = 4
```

```r
get_sqrt(a= 5, x0 = 3, tol = 1e-8, iter_max = 1e4, verbose = TRUE)
```

```
## Approximated square root = 2.333333, iteration = 1
## Approximated square root = 2.238095, iteration = 2
## Approximated square root = 2.236069, iteration = 3
## Approximated square root = 2.236068, iteration = 4
## Square root = 2.236068, iteration = 4
```

(c) How would the program change to compute an arbitrary, positive integer, root? Please use calculus to determine an appropriate iterative equation for approximating this arbitrary root.

The program will differ because in this new case, we will not know the exact f function and the f prime function when writing the code since we will know exactly what the functions are after getting the user's input. The f function and f prime function must be expressed using the root variable since we do not know the exact functions in these scenarios. For example, if we wanted to do the same thing in the example in part b, but using cube root, we must take the variable root = 3, and the f function will be x^3 - a and correspondingly, the f prime function will be 3x^2 - a because this is the derivative of the f function. Thus, where we used 3 to represent the cube of x, we want to substitute it with the variable root so that the user's input can be correctly reflected on the functions.

(d) Write an R function, **`get_abroot()`**, like the one for square root which calculates the arbitrary root of a number. I.e., in addition to the arguments in `get_sqrt()`, the `root` argument is also required.

In .R Code

```r
knitr::include_graphics("get_abroot.png")
```

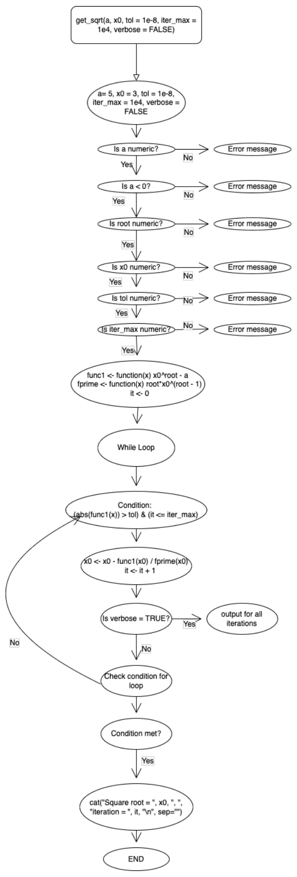

get_sqrt(a, x0, tol = 1e-8, iter_max = 1e4, verbose = FALSE)
a= 5, x0 = 3, tol = 1e-8, iter_max = 1e4, verbose = FALSE
Is a numeric?
No
Error message
Yes
Is a < 0?
No
Error message
Yes
Is root numeric?
No
Error message
Yes
Is x0 numeric?
No
Error message
Yes
Is tol numeric?
No
Error message
Yes
Is iter_max numeric?
No
Error message
Yes
func1 <- function(x) x0^root - a
fprime <- function(x) root*x0^(root - 1)
it <- 0
While Loop
Condition:
(abs(func1(x)) > tol) & (it <= iter_max)
x0 <- x0 - func1(x0) / fprime(x0)
it <- it + 1
Is verbose = TRUE?
Yes
output for all iterations
No
No
Check condition for loop
Condition met?
Yes
cat("Square root = ", x0, ", ", "iteration = ", it, "\n", sep="")
END

Figure 2: General stucture of a flowchart.

(e) Give an example of your function by computing the seventh root of 13.

```
get_abroot(13, 7, 4)
```

```
## Square root = 1.442563, iteration = 11
```

```
get_abroot(13, 7, 4, verbose = TRUE)
```

```
## Approximated square root = 3.429025, iteration = 1
## Approximated square root = 2.940307, iteration = 2
## Approximated square root = 2.523137, iteration = 3
## Approximated square root = 2.169886, iteration = 4
## Approximated square root = 1.877695, iteration = 5
## Approximated square root = 1.651826, iteration = 6
## Approximated square root = 1.507275, iteration = 7
## Approximated square root = 1.450326, iteration = 8
## Approximated square root = 1.442686, iteration = 9
## Approximated square root = 1.442563, iteration = 10
## Approximated square root = 1.442563, iteration = 11
## Square root = 1.442563, iteration = 11
```

(f) Let $e^k = |x_k - \sqrt[7]{13}|$, where k indicates the kth iteration. Please print from e_1 to e_4, and specify your findings.

```r
e1 <- abs(3.42902483259 - 1.44256291944)
e1
```

```
## [1] 1.986462
```

```r
e2 <- abs(2.940306551 - 1.44256291944)
e2
```

```
## [1] 1.497744
```

```r
e3 <- abs(2.52313677705 - 1.44256291944)
e3
```

```
## [1] 1.080574
```

```r
e4 <- abs(2.16988647887 - 1.44256291944)
e4
```

```
## [1] 0.7273236
```

As we can see, the error is getting less and less as e1 is 1.9864619 and this value goes down to 0.7273236 for e4. Additionally, we can see how the errors are decreasing at a non-linear pace.

2 Optimization Problem

(a) Using calculus, find the minimum of the function

$$f(x) = x^n - n\alpha \log(x), n \in \{1, 2, ..., \},\ \alpha > 0$$

$$f(x) = x^n - n\alpha \log(x), \quad n \in \{1, 2, \cdots\}, \quad \alpha > 0$$

First, we take derivative of $f(x)$

$$f'(x) = n x^{n-1} - n\alpha \frac{1}{x}$$

Next we want $f'(x) = 0$

$$0 = n x^{n-1} - n\alpha \frac{1}{x}$$

$$\cancel{n} x^{n-1} = \cancel{n}\alpha \frac{1}{x}$$

$$x^n = \alpha$$

$$x = \sqrt[n]{\alpha}$$

Next, we want to find $f''(x)$

$$f''(x) = (n)(n-1)(x^{n-2}) + \frac{n\alpha}{x^2}$$

Using calculus, we know that there is a local maximum or local minimum where f'(x) = 0. Since the given f(x) is a convex function, we get that the local minimum where f'(x) = 0.

(b) Using your formulas for the derivatives of f, write down Newton's method to find the minimum.

In this case, to find the local minimum indicates finding x such that f'(x) = 0 and this also indicates finding the root of f'(x).

Using Newton's Method, we attempt to find local min.

$$f(x) = x^n - n\alpha \log(x)$$

$$f'(x) = n x^{n-1} - \frac{n\alpha}{x}$$

$$f''(x) = n*(n-1)*x^{(n-2)} + \frac{n\alpha}{x^2}$$

$$x_1 = x_0 - \frac{n x_0^{n-1} - \frac{n\alpha}{x_0}}{n(n-1)x_0^{n-2} + \frac{n\alpha}{x_0^2}}$$

$$x_1 = x_0 - \frac{x_0^{n-1} - \frac{\alpha}{x_0}}{(n-1)x_0^{n-2} + \frac{\alpha}{x_0^2}}$$

$$x_1 = x_0 - \frac{x_0^{n+1} - x_0\alpha}{(n-1)x_0 + \alpha}$$

$$\vdots$$

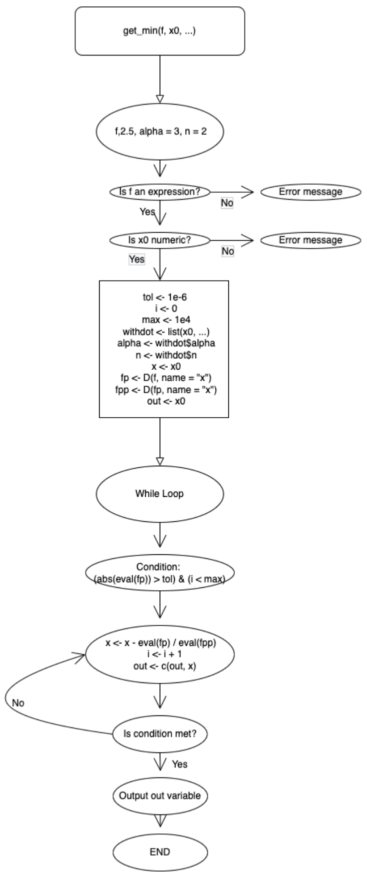
Continue until x converges to a specific value or if max iteration is reached

(c) Use R to write a function, `get_min()`, which takes the following arguments:

- `f`, an expression to minimize with respect to `x`,
- `x0`, an initial guess
- `...`, any additional variables needed by `f`

Choose an appropriate stopping rule and, using your function, return the minimum of f for $n = 2$ and $\alpha = 3$.

```
knitr::include_graphics("get_min.png")
```



```
tol <- 1e-6
i <- 0
max <- 1e4
withdot <- list(x0, ...)
alpha <- withdot$alpha
n <- withdot$n
x <- x0
fp <- D(f, name = "x")
fpp <- D(fp, name = "x")
out <- x0
```

```
x <- x - eval(fp) / eval(fpp)
i <- i + 1
out <- c(out, x)
```

Figure 3: General stucture of a flowchart.

```
f <- expression(x^n - alpha * n * log(x))

steps <- get_min(f, 2.5, alpha = 3, n = 2)
steps
```

```
## [1] 2.500000 1.621622 1.728299 1.732047 1.732051
```

```
steps <- get_min(f, 1, alpha = 3, n = 2)
steps
```

```
## [1] 1.000000 1.500000 1.714286 1.731959 1.732051
```

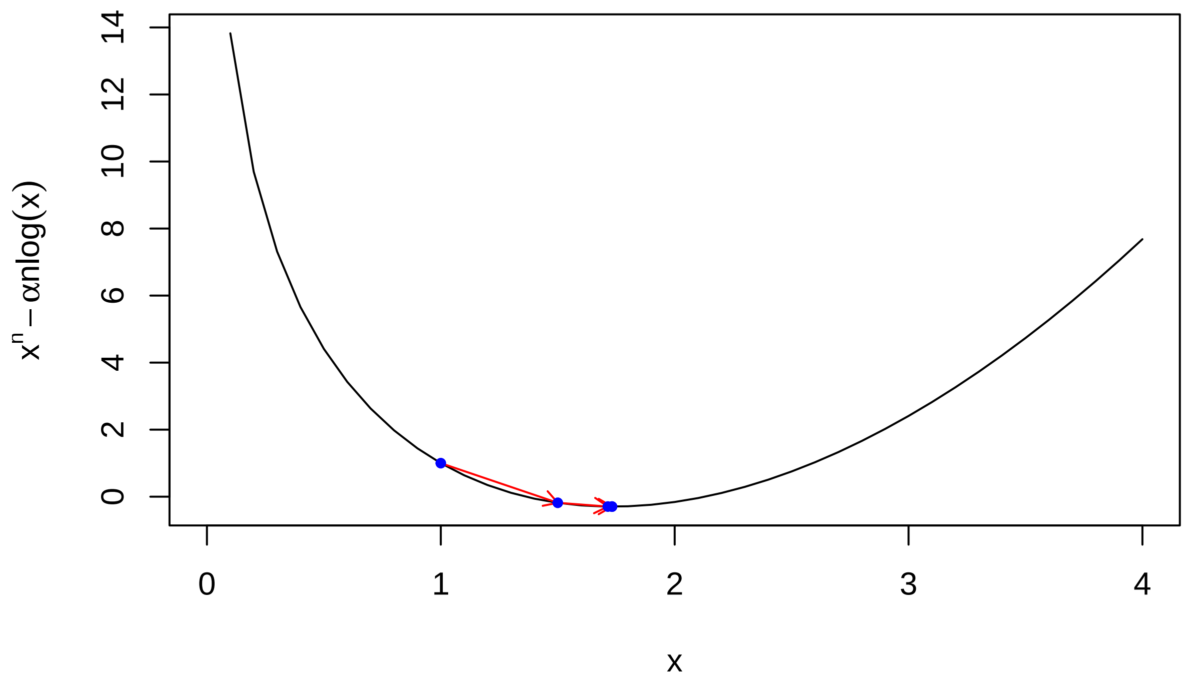
(d) Plot the function and the steps taken by your method starting from $x_0 = 1$.

```
f <- expression(x^n - alpha * n * log(x))
x <- seq(0, 4, 0.1)
y <- eval(f, list(x=steps, alpha=3, n=2))
```

```
plot(x, eval(f, list(x=x, alpha=3, n=2)), type="l", ylab=f)
for (i in 2:length(steps)){
  arrows(steps[i-1], y[i-1], steps[i], y[i], length=0.08, col="red")
}
```

```
## Warning in arrows(steps[i - 1], y[i - 1], steps[i], y[i], length = 0.08, : zero-
## length arrow is of indeterminate angle and so skipped
```

```
points(steps, y, col="blue", pch=19, cex=0.6)
```


$x^n - \alpha n \log(x)$
x