Project 1: ADMI 6807

Dr. A. Nieves-González

University of Puerto Rico
Río Piedras Campus
Institute of statistics

Due date: April 27.

This project consists on solving some of the problems listed below. In any problem that asks to write or develop an algorithm, it must be shown in pseudocode, and implemented in either R or MATLAB®.

1 Problem set 1

Do all of the following four problems.

1. Write an algorithm that given the price of an item and the local sales tax, it returns the price of the item sales tax included.

2. Develop an algorithm that given the hours worked per week, and the hourly pay rate, it returns the gross pay. The rules for determining gross pay are as follows: Regular pay rate for the hours up to 40; time-and-a-half for hours over 40 up to 54; and double time for hours over 54. After displaying the output the algorithm must ask the user wheter he/she wants to do another computation. Repeat the entire set of operations until the user says no.

3. *Median.*
   Write an algorithm to compute the median of a set of observations. The median of a set of observations $\{x_1, x_2, \ldots, x_n\}$ is the value for which half of the observations is less than such value. Compare the your result and the performance of your function with the function that computes the median that is already built-in in R or MATLAB®. You should make plots and tables of the running time of each program as a function of the size of the list of numbers. Hint: you can use the Selection Sort function discussed in class or the Bubble Sort as part of the algorithm.

4. *Fibonacci sequence.*
   The Fibonacci sequence is a sequence of integers defined as follows: the first two numbers are both 1. After that, each number in the sequence is sum of the two preceding numbers. Thus, if $F(n)$ stands for the nth value in the sequence then we have

$$F(n) = \begin{cases} F(n-1) + F(n-2) & \text{if } n > 2 \\ 1 & \text{if } n = 1 \text{ or } n = 2 \end{cases} \tag{1}$$

   It can be demonstrated (by strong induction) that a formula for $F(n)$ is

$$F(n) = \frac{\sqrt{5}}{5}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{\sqrt{5}}{5}\left(\frac{1-\sqrt{5}}{2}\right)^n \tag{2}$$

   The first few values of the sequence are: $1, 1, 2, 3, 5, 8, 13, 21, \ldots$ Do the following:

(a) Implement two functions: one that computes the nth value of the Fibonacci sequence using the definition (equation 1), and other that uses equation 2.

(b) Compute the value of the sequence for different n, e.g., $5, 10, 15, 20, 25, 30$. And compute the running time for each calculation using each one of the functions (make a table). Comment on the running time of each of the functions. Also, what are the advantages and disavantages of using one equation versus the other (also think about $\sqrt{5}$)?

(c) Verify numerically that the ratio of $F(n+1)/F(n)$ converges to the golden ratio: $(1+\sqrt{5})/2$. You can use the R-script given at the end of this document or you can write your own. In the given script $FiboSeq(n)$ is the function that you wrote to compute the nth value un the sequence. What happens if you uncomment the three lines below the Slow comment and comment the lines below the Fast comment inside the for loop? Give an explanation of the observed phenomenon.

Hint: Equation 1 can be implemented in a way akin to the way we implemented the factorial function ($n!$) in class.

2 Problem set 2

Do at least three of the following four problems.

1. *Insertion Sort.*
   Sorting is a very important problem in computer science. This problem deals with a simple, important, but not very efficient sorting algorithm: the insertion sort. Given a list of integers of size n the insertion sort is:

Algorithm 1 Insertion Sort.

Input: List of integers $u = (u_1, \ldots, u_n)$ of length n
Output: Ordered list of integers u (ascending order)
for $i = 1, \ldots, n$ **do**
  $j \leftarrow i$
  while $j > 1$ AND $u_{j-1} > u_j$ **do**
    $swap(u_{j-1}, u_j)$
    $j \leftarrow j - 1$
  end while
end for

Observe that the swap function (or procedure) just exchange the value of its arguments.

(a) Implement the insertion sort.

(b) Compare the insertion sort, the selection sort (discussed in class), and the built-in sort functions in terms of performance. You should make plots and tables of the running time of each program as a function of the size of the list of numbers. Note that in R you can choose between quick sort and shell sort as the sorting algorithm (in MATLAB a quick sort is the available method) which are very efficient, you should try them both. You can use a script simmilar to the one given at the end of this document to plot your results.

2. *Bubble Sort.*
   Sorting is a very important problem in computer science. This problem deals with a simple, important, but not very efficient sorting algorithm: the bubble sort. Given a list of integers of size n the bubble sort is:

Algorithm 2 Bubble Sort.

Input: List of integers $u = (u_1, \ldots, u_n)$ of length n
Output: Ordered list of integers u (ascending order)
$IsSwapped \leftarrow$ TRUE
while $IsSwapped =$ TRUE **do**
  $IsSwapped \leftarrow$ FALSE
  for $i = 2, \ldots, n$ **do**
    if $u_{i-1} > u_i$ **then**
      $swap(u_{i-1}, u_i)$
      $IsSwapped \leftarrow$ TRUE
    end if
  end for
  $n \leftarrow n - 1$
end while

(a) Implement the bubble sort.

(b) Also, compare the bubble sort, the selection sort, and the built-in sort functions in terms of performance. You should make plots and tables of the running time of each program as a function of the size of the list of numbers. Note that in R you can choose between quick sort and shell sort as the sorting algorithm (in MATLAB a quick sort is the available method) which are very efficient, you should try them both. You can use a script simmilar to the one given at the end of this document to plot your results.

3. *Bisection method.*
The bisection method is a root finding method based upon the intermediate value theorem. Let f be a function $f : [a, b] \to \mathbb{R}$. The root of a function is a value x^* such that $f(x^*) = 0$. Suppose that f is continuous on $[a, b]$. The bisection method is defined as:

Algorithm 3 Bisection Method

Input: A continuous function $f : [a, b] \to \mathbb{R}$, the endpoints of interval $[a, b]$ such that $f(a)f(b) < 0$, the maximum number of iterations $n_{\max}$, and a specified tolerance $tol > 0$.
Output: x_k, an approximation of a root of f.
$k \leftarrow 0$, $err \leftarrow tol + 1$
$a_0 \leftarrow a$, $b_0 \leftarrow b$, $x_0 \leftarrow (a_0 + b_0)/2$
while $k \leq n_{\max}$ AND $err > tol$ **do**
  if $f(a_k)f(x_k) < 0$ **then**
    $a_{k+1} \leftarrow a_k$
    $b_{k+1} \leftarrow x_k$
  else
    if $f(a_k)f(x_k) > 0$ **then**
      $a_{k+1} \leftarrow x_k$
      $b_{k+1} \leftarrow b_k$
    else
      $a_{k+1} = x_k - tol$
      $b_{k+1} = x_k + tol$
    end if
  end if
  $x_{k+1} \leftarrow (a_{k+1} + b_{k+1})/2$
  $err \leftarrow 0.5|b_k - a_k|$
  $k \leftarrow k + 1$
end while

Use the bisection method to find the roots of the following continuous functions:

(a) $f(x) = x^2 - 1$

(b) $f(x) = x^3 - 3x^2 + 2x$. Whose roots lie within $[0, 2]$.

(c) The Legendre polynomial of degree 5

$$f(x) = \frac{x}{8}(63x^4 - 70x^2 + 15),$$

whose roots lie within interval $(-1, 1)$. You can start with $a_0 = 0.6$, $b_0 = 1$, $n_{\max} = 100$, and $tol = 10^{-10}$ and you should get the approximation of the root $x^* \approx 0.9062$.

Try different initial guesses for each of the different problems, and make a table with the sucessive approximations (one table for each initial guess). Try to find the all the roots, if you cannot, explain what difficulties you encounter.

4. *The Chord Method.*
The chord method is a root finding method based upon a truncation of a Taylor's series expansion around a root of the function f. Let f be a function $f : [a, b] \to \mathbb{R}$. The root of a function is a value x^* such that $f(x^*) = 0$. Suppose f is differentiable on $[a, b]$. The chord method is defined as:

Algorithm 4 Chord Method

Input: A differentiable function $f : [a, b] \to \mathbb{R}$, an initial guess x_0, the endpoints of interval $[a, b]$, the maximum number of iterations $n_{\max}$, and a specified tolerance $tol > 0$.
Output: x_k, an approximation of a root of f.
$k \leftarrow 0$, $err \leftarrow tol + 1$
while $k \leq n_{\max}$ AND $err > tol$ **do**
  $x_{k+1} \leftarrow x_k - \left(\frac{f(b)-f(a)}{b-a}\right)^{-1} f(x_k)$
  $err \leftarrow |x_{k+1} - x_k|$
  $k \leftarrow k + 1$
end while

Use the chord method to find the roots of the following continuous functions:

(a) $f(x) = x^2 - 1$

(b) $f(x) = x^3 - 3x^2 + 2x$. Whose roots lie within $[0, 2]$.

(c) The Legendre polynomial of degree 5

$$f(x) = \frac{x}{8}(63x^4 - 70x^2 + 15),$$

whose roots lie within interval $(-1, 1)$. You can start with $x_0 = 0.85$, $n_{\max} = 100$, and $tol = 10^{-10}$ and you should get the approximation of the root $x^* \approx 0.9062$.

Try different initial guesses for each of the different problems, and make a table with the sucessive approximations (one table for each initial guess). Try to find the all the roots, if you cannot, explain what difficulties you encounter.

3 Additional remarks and R scripts

3.1 Remarks on how two measure running time and pass functions as arguments of another function.

- How to measure running time?
  - In MATLAB®: Use the functions tic and toc. For example:

$$\begin{aligned} \text{starttime} &= \text{tic;} \\ \text{Some MATLAB® Code} & \\ \text{elapsedtime} &= \text{toc;} \\ \text{RunTime} &= \text{elapsedtime} - \text{starttime} \end{aligned}$$

  - In R: Use the proc.time() function. For example:

$$\begin{aligned} \text{starttimes} &= \text{proc.time()} \\ \text{Some R Code} & \\ \text{EndTimes} &= \text{proc.time()} - \text{starttimes} \end{aligned}$$

- How to pass a function as an argument (input) to another function?
  - In MATLAB®: Use function handle.
  - In R: Define the function in another file as shown in class, issue the source('filename.r') where 'filename.r' is where the function where the function is defined, pass the function as a standard argument.

3.2 R Scripts

3.2.1 R script number 1.

```
# We are going to verify that the ratio of sucessive terms of the
# Fibonacci sequence converges to the golden ratio.
source("FiboSeq.r")
# An approximation to the golden ratio.
golden_rat = (1 + sqrt(5))/2

N = 1000
Rat = rep(0,N)
i = 2
starttimes = proc.time()
Fprev = FiboSeq(1)
Fnext = FiboSeq(2)
for (i in 2:N)
{ # Slow
  # Fnext = FiboSeq(i)
  # Rat[i-1] = Fnext/Fprev
  # Fprev = Fnext
  # Fast
  Rat[i-1] = Fnext/Fprev
  temp = Fnext
  Fnext = Fnext + Fprev
```

```
  Fprev = temp

}
Rat[N] = Fnext/Fprev
endtimes = proc.time() - starttimes
Iterations = 1:N
err = abs(golden_rat - Rat)

# Plots
par(mfrow=c(2,1))
plot(Iterations,Rat)
abline(a=golden_rat,b=0)
plot(Iterations,err)
```

3.2.2 R script number 2.

```
# Script to compare sorting programs
source("InserSort.r")
source("BubbleSort.r")
source("FindLargest.r")
source("SelectionSort.r")
p = 3 # the list of numbers are of sizes 10^1,...,10^p
m = seq(1,p,by=1)
times = matrix(rep(0,5*p),nrow=5,ncol=p,byrow=TRUE)
N = as.integer(rep(0,length(m)))
proctimeindex = 3

for (i in 1:length(m))
{ N[i] = as.integer(10^m[i])

  uarr = seq(N[i],1,by=-1)
  oarr = uarr
  st = proc.time()
  oarr = InserSort(uarr)
  tend = proc.time()
  telap = tend - st
  times[1,i] = telap[proctimeindex]

  uarr = seq(N[i],1,by=-1)
  oarr = uarr
  st = proc.time()
  oarr = SelectionSort(uarr)
  tend = proc.time()
  telap = tend - st
  times[2,i] = telap[proctimeindex]

  uarr = seq(N[i],1,by=-1)
  oarr = uarr
  st = proc.time()
  oarr = BubbleSort(uarr)
  tend = proc.time()
  telap = tend - st
  times[3,i] = telap[proctimeindex]

  uarr = seq(N[i],1,by=-1)
```

```
  oarr = uarr
  st = proc.time()
  oarr = sort.int(uarr,method="quick")
  tend = proc.time()
  telap = tend - st
  times[4,i] = telap[proctimeindex]

  uarr = seq(N[i],1,by=-1)
  oarr = uarr
  st = proc.time()
  oarr = sort.int(uarr,method="shell")
  tend = proc.time()
  telap = tend - st
  times[5,i] = telap[proctimeindex]
}

# Plots
names = c("Insert Sort","Selection Sort", "Bubble Sort",
  "Quick Sort","Shell Sort" )
par(mfrow=c(2,3))
for (j in 1:5)
{
  plot(N,times[j,],main=names[j])
}
```