# Computing for Data Sciences – 2017

PGDBA, First Year, First Semester, Indian Statistical Institute

## Assignment 1

Discussed during 2–4 August 2017 | Posted on 7 August 2017 | Submit by 15 August 2017

### Problem 1

[20 points]

**Background :** Recall the `LinSearchPeak1D()` and `BinSearchPeak1D()` functions you wrote in Assignment 0 for 1D-Peak search. We will hereby try to estimate the runtime of the algorithms.

**Task :** Assume that the input list $L$ contains random integers between $-999$ to 999 (both inclusive) in all the following cases. Vary $n$, the size of the list $L$, as `[1, 10, 100, 1000, 10000, 100000, 1000000]`, and for each such $n$, perform 1000 experimental runs of `LinSearchPeak1D()` and `BinSearchPeak1D()`, with randomly generated list $L$, and store (in a txt/csv file) the total number of integer-to-integer comparisons made in each such experiment against the chosen $n$.

### Problem 2

[20 points]

**Background :** Recall the `SearchPeak2D()` function you wrote in Assignment 0 for 2D-Peak search. We will hereby try to estimate the runtime of the algorithm you implemented.

**Task :** Assume that the input matrix $M$ contains random integers between $-999$ to 999 (both inclusive) in all the following cases. Vary $n$, the dimension of the matrix $M$, as `[1, 10, 100, 1000, 10000]`, and for each such $n$, perform 1000 experimental runs of `SearchPeak2D()`, with randomly generated $n \times n$ matrix $M$, and store (in a txt/csv file) the total number of integer-to-integer comparisons made in each such experiment against the chosen dimension $n$.

### Problem 3

[20 points]

**Background :** Recall the *Binary Search* approach and the *Newton-Raphson* approach to compute the square root of an integer, as discussed in class. We will study the errors and convergence.

**Task :** Write two Python functions, `BinarySquareRoot()` and `NewtonSquareRoot()`, each of which takes as input a positive integer $n$, an initial guess $x_0$, a fixed number of iterations $s$, and outputs – (a) the approximate value of the square root $x_{i+1} \approx \sqrt{n}$ after each iteration of the algorithm, and (b) the relative error $\epsilon_{i+1} = |(x_{i+1} - x_i)/x_{i+1}|$ after each iteration of the algorithm, where $i = 0, 1, 2 \ldots, s-1$. Try to notice which of the two methods converges faster.

**Bonus :** You may want to input a *tolerance* parameter $t$ in the functions, and stop iterating when relative error $\epsilon_{i+1} = |(x_{i+1} - x_i)/x_{i+1}|$ is less than $t$. This eliminates the requirement of $s$.

## Problem 4

[20 points]

**Background :** Recall the generic *Newton-Raphson* approach to find a solution (also called a root) of an equation $f(x) = 0$, as discussed in class. We will study the errors and convergence.

$$\text{Iteration for Newton-Raphson :} \qquad x_{i+1} \;=\; x_i \;-\; \frac{f(x_i)}{f'(x_i)} \qquad \text{for } i = 0, 1, 2, \ldots$$

**Task :** Write a Python function, `NewtonFindRoot()`, which takes as input a function $f(x)$, the derivative of the function $f'(x)$, an initial guess for the root $x_0$, a fixed number of iterations $s$, and outputs – (a) the approximate value of the root $x_{i+1}$ after each iteration of the algorithm, such that $f(x_{i+1}) \approx 0$, and (b) the relative error $\epsilon_{i+1} = |(x_{i+1} - x_i)/x_{i+1}|$ after each iteration of the algorithm, where $i = 0, 1, 2 \ldots, s-1$. Try to notice the converge rate of the algorithm.

**Bonus :** You may want to input a *tolerance* parameter $t$ in the functions, and stop iterating when relative error $\epsilon_{i+1} = |(x_{i+1} - x_i)/x_{i+1}|$ is less than $t$. This eliminates the requirement of $s$.

## Problem 5

[20 points]

**Background :** Recall the *Secant Method*, a variant of the Newton-Raphson Method, to find a root of an equation $f(x) = 0$, as discussed in class. We will study the errors and convergence.

$$\text{Iteration for Secant Method :} \qquad x_{i+1} \;=\; x_i \;-\; f(x_i) \cdot \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} \qquad \text{for } i = 1, 2, 3, \ldots$$

**Task :** Write a Python function, `SecantFindRoot()`, which takes as input a function $f(x)$, two initial guesses for the root $x_0$ and $x_1$, preferably with $f(x_0)$ and $f(x_1)$ of opposite signs so that the guesses are guaranteed to contain a root within, a fixed number of iterations $s$, and outputs – (a) the approximate value of the root $x_{i+1}$ after each iteration of the algorithm, such that $f(x_{i+1}) \approx 0$, and (b) the relative error $\epsilon_{i+1} = |(x_{i+1} - x_i)/x_{i+1}|$ after each iteration of the algorithm, where $i = 1, 2 \ldots, s-1$. Try to notice the converge rate of the algorithm.

**Bonus :** You may want to input a *tolerance* parameter $t$ in the functions, and stop iterating when relative error $\epsilon_{i+1} = |(x_{i+1} - x_i)/x_{i+1}|$ is less than $t$. This eliminates the requirement of $s$.

---

**Submission :** This is an individual assignment, and everyone must submit their own code. You should submit a single Python file – `rollXXassignment1.py` – containing all the functions mentioned above (and any other associated code), where `XX` is your roll number, as in `17BM6JPXX`.

Your submission should be emailed to `sg.sourav@gmail.com` by midnight of 15 August 2017.

---

*Properly acknowledge every source of information that you referred to, including discussions with your friends. Verbatim copy from any source is strongly discouraged, and plagiarism will be heavily penalized. It is strongly recommended that you write the codes completely on your own.*