Computational Physics, FC 2020-1

Homework 3

October 8, 2019

Carlos Gerardo Malanche Flores & Víctor Alfredo Milchorena González

There is a part that will get evaluated by the server, and another one that you need to hand in in paper. For the digital one, you have to create (again) a folder inside your `Homeworks` folder, called `Homework3`. Due date: **October 23rd 2019, 23:59:59**.

Task 1: Probabilistic analysis (*)

As we saw in class, an algorithm has a worst case scenario, a best case scenario, and given certain information about the distribution of the input, an expected run time bound. Recall the example from the lectures about an algorithm to find the student that arrived first to the class. We derived that the expected complexity was $\mathcal{O}(n)$ (but with a 0.5 multiplier advantage, comparing to the overall worst case), asking each student starting with the first chair.
Your goal is to compute the expected time complexity of the same algorithm, given that the students choose places (among an infinite number of chairs) with the following distribution

$$p(\text{student sits on empty chair number } i) = \frac{e^{-5}}{i!}5^i$$

Task 2: Master Theorem (*)

You will be presented by 3 algorithms. The goal is that you identify the recurrence relation, so that you can derive the time complexity for each of the algorithms.

- The goal is to compute the matrix product $R = AB$, with $R, A, B \in \mathbb{R}^{n\times n}$. For simplicity, we assume that $n = 2^k$ for some $k > 0$. Now, we split the matrices into submatrices with $n/2$ rows and columns:

$$A = \begin{bmatrix} \boldsymbol{A}_{1,1} & \boldsymbol{A}_{1,2} \\ \boldsymbol{A}_{2,1} & \boldsymbol{A}_{2,2} \end{bmatrix}, \quad B = \begin{bmatrix} \boldsymbol{B}_{1,1} & \boldsymbol{B}_{1,2} \\ \boldsymbol{B}_{2,1} & \boldsymbol{B}_{2,2} \end{bmatrix}, \quad R = \begin{bmatrix} \boldsymbol{R}_{1,1} & \boldsymbol{R}_{1,2} \\ \boldsymbol{R}_{2,1} & \boldsymbol{R}_{2,2} \end{bmatrix}$$

  which tells us that the result of the product AB looks like

$$\begin{aligned} \boldsymbol{R}_{1,1} &= \boldsymbol{A}_{1,1}\boldsymbol{B}_{1,1} + \boldsymbol{A}_{1,2}\boldsymbol{B}_{2,1} \\ \boldsymbol{R}_{1,2} &= \boldsymbol{A}_{1,1}\boldsymbol{B}_{1,2} + \boldsymbol{A}_{1,2}\boldsymbol{B}_{2,2} \\ \boldsymbol{R}_{2,1} &= \boldsymbol{A}_{2,1}\boldsymbol{B}_{1,1} + \boldsymbol{A}_{2,2}\boldsymbol{B}_{2,1} \\ \boldsymbol{R}_{2,2} &= \boldsymbol{A}_{2,1}\boldsymbol{B}_{1,2} + \boldsymbol{A}_{2,2}\boldsymbol{B}_{2,2} \end{aligned}$$

  Your first task is to find the complexity of this divide and conquer sratEgy, using the master theorem and the recurrence relation for this algorithm. After you are done, then work with the next situation: given the following 7 $n/2$-matrices

$$\begin{aligned} \boldsymbol{M}_1 &= (\boldsymbol{A}_{1,1} + \boldsymbol{A}_{2,2})(\boldsymbol{B}_{1,1} + \boldsymbol{B}_{2,2}) \\ \boldsymbol{M}_2 &= (\boldsymbol{A}_{2,1} + \boldsymbol{A}_{2,2})\boldsymbol{B}_{1,1} \\ \boldsymbol{M}_3 &= \boldsymbol{A}_{1,1}(\boldsymbol{B}_{1,2} - \boldsymbol{B}_{2,2}) \\ \boldsymbol{M}_4 &= \boldsymbol{A}_{2,2}(\boldsymbol{B}_{2,1} - \boldsymbol{B}_{1,1}) \\ \boldsymbol{M}_5 &= (\boldsymbol{A}_{1,1} + \boldsymbol{A}_{1,2})\boldsymbol{B}_{2,2} \\ \boldsymbol{M}_6 &= (\boldsymbol{A}_{2,1} - \boldsymbol{A}_{1,1})(\boldsymbol{B}_{1,1} + \boldsymbol{B}_{1,2}) \\ \boldsymbol{M}_7 &= (\boldsymbol{A}_{1,2} - \boldsymbol{A}_{2,2})(\boldsymbol{B}_{2,1} + \boldsymbol{B}_{2,2}) \end{aligned}$$

  Show that

$$\begin{aligned}
\boldsymbol{R}_{1,1} &= \boldsymbol{M}_1 + \boldsymbol{M}_4 - \boldsymbol{M}_5 + \boldsymbol{M}_7\\
\boldsymbol{R}_{1,2} &= \boldsymbol{M}_3 + \boldsymbol{M}_5\\
\boldsymbol{R}_{2,1} &= \boldsymbol{M}_2 + \boldsymbol{M}_4\\
\boldsymbol{R}_{2,2} &= \boldsymbol{M}_1 - \boldsymbol{M}_2 + \boldsymbol{M}_3 + \boldsymbol{M}_6
\end{aligned}$$

and compute the complexity of this algorithm using the Master Theorem. The algorithm is known as *Strassen algorithm*.

- You have a sorted array $\{a_i\}_{i=1}^n$ with n entries, the algorithm finds an element e in this array by looking at the element in the middle of the array. If $e \neq a_{n/2}$, then the algorithm decides whether the search should continue in the upper half or in the lower half of the array, the former if $a_{n/2} < e$ and the latter if $a_{n/2} > e$. We already know the complexity of such a binary search, but the goal here is to write it as a recurrence relation and use the master theorem to find the time complexity.

- You have n points $\{\vec{p_i}\}_{i=1}^n$ in $\mathbb{R}^2$. The goal of the algorithm is to find the minimum distance between two points in the set. The algorithm goes with the following steps

  1. You sort the points by using the x coordinate only.
  2. Now you split the points into two equal sized sets (the first one contains the first $n/2$ points with the lowest x value), which defines a line in p_{mid} the middle point, and now you solve the problem in a recursive fashion for each half.
  3. You will end up with the minimum distance between points in the left set (d_{left}), and the minimum distance between points in the right set (d_{right})
  4. Now you compute the minimum distance between all the points in the left, and all the points in the right, which yields d_{center}
  5. The answer is the minimum between d_{left}, d_{right} and d_{center}.

  Compute the time complexity of this algorithm by writing the recurrence relation.

  Note: The step number 4 in the algorithm looks like it needs to be $\mathcal{O}(n^2)$, which comes from computing the distance from each of the $n/2$ points on the left to each of the $n/2$ points in the right. But this can be accomplished in linear time with the following observation: as we are looking for the smallest distance between points in the left and in the right, we may not exceed the minimum between d_{left} and d_{right} (let's call it d_{rlmin}). So given a point p_i on the left, we should not search further than the box of width d_{rlmin} and height $2d_{\text{rlmin}}$, centered vertically in p_i's y coordinate, and touching with the left side the divisory line of the sets. Think why this box cannot contain more than 6 points.

Task 3: `sortingalgos.cpp` (***)

The file `SortMachines.h` contains the definition of 4 sorting methods, *Insertion Sort*, *Bubble Sort*, *Heap Sort* and *Merge Sort*, although they are empty.

Your task is simply to fill in the `sort` method that the 4 classes inherit from the class `SortMachine`. You can use the helper method `print` to print the vector at the end of each iteration.

The file `Sorting.cpp` contains minimal code to test your sort implementations with the 4 sorting algorithms, using a randomized 10-entry vector, and using the default (not implemented) Bubble Sort class.

Note: You can look up other sorting algorithms that are not so hard to implement and try to add a 5th sorting algorithm!

Task 4: `legendre.cpp` (***)

Legendre polynomials are quite popular among some physicists, they are the polynomial solutions[1] for the so called **Legendre equation**:

$$\frac{d}{dx}\left[(1-x^2)\frac{dP(x)}{dx}\right]+\lambda P(x)=0$$

These polynomials can be defined by mean of the following recursive relations:

$$P_0(x)=1,\ P_1(x)=x,\ P_{n+2}=\frac{2n+3}{n+2}xP_{n+1}-\frac{n+1}{n+2}P_n$$

- First of all, you ought to write a code that computes the coefficients of the n-th Legendre polynomial (which is a polynomial of degree n).
- Using the previous result, compute the roots of the first 10 P_n polynomials, so that they are correct up to the sixth decimal place, and write the output in the file `roots.txt`.

> **Note:** These roots belong to the interval $[-1,1]$, and the polynomials have a definite symmetry with respect to the origin, so you just have to be careful to pick an appropriate partition of the interval and the right initial points depending on the method of your choice; you must use at least once every method we mentioned in the lecture.[a]
>
> [a]Bisection, Secant, Newton and Fixed Point

For $n\in\mathbb{N}$, the previously obtained roots $a_k^{(n)}$, $k=1,2,...,n$ of $P_n(x)$ may be used as nodes for a quadrature method with weights given by:

$$w_k=\frac{1}{P_n'(a_k)}\int_{-1}^{1}\frac{P_n(x)}{x-a_k}dx,$$

such a method is known as **Gauss-Legendre quadrature**. So any integral that's meant to be evaluated in the interval $[-1,1]$ may be numerically approximated by:

$$\int_{-1}^{1}f(x)dx\sim\sum_{k=1}^{n}w_kf(a_k)$$

- For evaluating an integral with arbitrary limits $[\alpha,\beta]$, prove that:

$$\int_{\alpha}^{\beta}f(x)dx\sim\left(\frac{\beta-\alpha}{2}\right)\sum_{k=1}^{n}w_kf\left(a_k\left(\frac{\beta-\alpha}{2}\right)+\left(\frac{\alpha+\beta}{2}\right)\right),$$

- and use that method, with orders $1,2,...,10$ to compute the integrals:

$$\int_{-1}^{1}\frac{dx}{\sqrt{1-x^2}},\quad\int_{1}^{2}\frac{dx}{x}$$

  with 6 decimal places.[2]

  Compare with the results you'd get, had you used the trapezoidal and Simpson's rules.

> **Note:** You may use Gauss-Legendre quadrature in the following tasks, should you decide to do so.

[1]As opposed to solutions that must be expressed as infinite series.
[2]Btw, these are definitions of π and $\ln 2$, respectively.

Task 5: `bessel.cpp` (**)

Bessel functions (of first kind) are popular among physicists who enjoy painful experiences; they are solutions for the so called **Bessel equation**:

$$x^2y'' + xy' + (x^2 - \nu^2) = 0$$

Due to it's relevant role in solving Laplace equation in cylindrical coordinates, ν is usually an integer n. For such an election, it is possible to index the Bessel functions according to its *order*, n, and get the following integral representation:

$$J_n(x) = \frac{1}{\pi}\int_0^{\pi} \cos\left(n\theta - x\sin(\theta)\right)d\theta$$

With the equation above, and using a nice quadrature method, you may approximate the value of J_n at x.
Doing so, and using any numerical method, find the first 4 roots of the first 5 Bessel functions.

Task 6: `erf.cpp` (*)

A common function in statistics is the *error function*, which is defined as:

$$Erf(x) := \frac{1}{\sqrt{\pi}}\int_{-x}^{x} e^{-t^2}dt.$$

- Using any quadrature method, compute $Erf(x)$, for $x = -3 + hk, k = 0, 1, 2, ..., 6/h$, and $h = 0.01$
- However, this function may also be obtained as the solution of the following second order ICP:

  $$y'' + 2xy' = 0, \quad y(0) = 0, \quad y'(0) = \frac{2}{\sqrt{\pi}}$$

  Solve the equation using any of the methods we discussed during the lectures and compare with the previous computations.

Task 7: `vanderpol.cpp` (**)

A distant relative of the classic harmonic oscillator is a non-linearly damped harmonic oscillator that goes by the name of *Van der Pol oscillator*, which is the system described by the following differential equation:

$$\frac{d^2x}{dt^2} - \mu(1 - x^2)\frac{dx}{dt} + x = 0$$

This system doesn't have an exact analytic solution so our only hope is to solve it numerically...
Obtain both x and dx/dt from $t = 0$ to $t = 10$, given $\mu = 1$ and the initial conditions:

1. $x(0) = 0; \quad \frac{dx}{dt}(0) = 2.16108$
2. $x(0) = 1; \quad \frac{dx}{dt}(0) = 0$
3. $x(0) = -2.161; \quad \frac{dx}{dt}(0) = 2.53$

Plot the results and save each plot as `vanderpolX.png` (where X=1,2,3)

Task 8: RK4 (**)

Do the math in order to obtain the 4th order Runge Kutta weights as seen in the lecture notes.

$$y_{n+1} = y_n + hK_4(x_n, y_n, h), n = 0, 1, 2, ...$$

$$K_4(x_n, y_n, h) = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = f(x_n, y_n), k_2 = f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right), k_3 = f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right), k_4 = f\left(x_n + h, y_n + hk_3\right)$$