FIRST MIDTERM

Scientific Computing II
Fundación Universitaria Konrad Lorenz

- The exam will be available from September 06th, 2024. From that time on you will have one whole week to complete it.
- The solution must be turned in only through the Aula virtual. Midterms sent through email will not be graded.
- Full score will only be given to correct and completely justified answers. Miraculous, obtuse and unnecessarily complex solutions will receive partial or null score.
- You can ask any question of this assignment during class or through email: julian.jimenezc@konradlorenz.edu.co.

Finite Differences Method

The Taylor series expansion of a univariate function f, centered at x_0, is given by

$$f(x_0+h)=\sum_{n=0}^{\infty}\frac{f^{(n)}(x_0)}{n!}h^n=f(x_0)+\frac{f'(x_0)}{1!}h+\frac{f''(x_0)}{2!}h^2+\frac{f^{(3)}(x_0)}{3!}h^3+\frac{f^{(4)}(x_0)}{4!}h^4+\cdots, \tag{1}$$

and reciprocally, if we go backwards, we get

$$f(x_0-h)=f(x_0)-\frac{f'(x_0)}{1!}h+\frac{f''(x_0)}{2!}h^2-\frac{f^{(3)}(x_0)}{3!}h^3+\frac{f^{(4)}(x_0)}{4!}h^4+\cdots. \tag{2}$$

Subtracting (1)-(2) gives

$$f(x_0+h)-f(x_0-h)=2f'(x_0)h+\frac{2h^3}{3!}f^{(3)}(x_0)+\cdots, \tag{3}$$

from where we can get an approximation for the first-order derivative of f, namely

$$f'(x_0)=\frac{f(x_0+h)-f(x_0-h)}{2h}+h^2\xi=\frac{f(x_0+h)-f(x_0-h)}{2h}+\mathcal{O}\left(h^2\right), \tag{4}$$

where ξ is an error term which we can summarize by saying that our approximation is the first quotient plus a function whose order is at least h^2, written as $\mathcal{O}\left(h^2\right)$. This is important, because if we take h sufficiently small, h^2 (the error magnitude) becomes negible. Moreover, note that (4) is a better approximation than the usual one

$$f'(x_0)\approx\frac{f(x_0+h)-f(x_0)}{h}. \tag{5}$$

as the latter is of order $\mathcal{O}(h)$ (you should be familiar with this derivation).

If we do a similar procedure to the one we did for the first-order derivative of f, namely, if we sum (1)+(2), we get the following approximation for the second-order derivative.

$$f''(x_0) = \frac{f(x_0+h) - 2f(x_0) + f(x_0-h)}{h^2} + \mathcal{O}\left(h^2\right). \tag{6}$$

Note that this approximating method is easily generalizable for partial differential equations (why?).

1. (0/10) Find an approximation for the third-order derivative of f at x_0 whose error is $\mathcal{O}\left(h^2\right)$. *Hint:* Taylor expand $f(x_0+2h)$ and $f(x_0-2h)$ and use it with (1)-(2) to find a linear combination that gives you the approximation with the desired error.

Korteweg–De Vries equation

The KdV equation is a partial differential equation which models the waves on shallow water surfaces, or more generally, it model one-dimensional nonlinear dispersive waves described by a function $\phi(x,t)$ adhering to:

$$\frac{\partial \phi}{\partial t} + \frac{\partial^3 \phi}{\partial x^3} - \phi \frac{\partial \phi}{\partial x} = 0, \quad x \in \mathbb{R}, t \geq 0. \tag{7}$$

2. (0/10) Provide a numerical method based on finite differences to numerically integrate this partial differential equation. Do not forget to explain the order of the error of your method (which should be $\mathcal{O}(h^2)$, where h is the size of the discretization of your grid).

3. (0/10) Numerically integrate the differential equation in Python or C/C++ in a finite spatial interval using a meaningful initial condition $\phi(x,0)$. Provide your solution as a simulation where you display the wave evolution as a function of time.

Do not forget to optimize and vectorize your code, as not doing so will lead to grade penalizations. Document your code extensively, and explain your optimization strategies. Moreover, make sure that your numerical method indeed converges.

Parallel Sorting

A sorting algorithm is used to arrange elements of a list in a specific order. For example, such algorithm will transform $[1,-1,2,0,10,-9]$ to $[-9,-1,0,1,2,10]$. Your task is to implement a parallel sorting algorithm that divides an array in p (where p is the number of processors to be used) parts, sorting them separately and then using a processor to **optimally** merge these parts in a sorted array.

4. (0/20) Take an array of randomly generated uniform integers from 0 to 100 of size $N = 10^6$ and sort them using your parallel implementation in Python. Compare the performance of your implementation with sequential sorting.