S&DS663 Computational Mathematics for Data Science

Assignment 0

August 28, 2024

Homework Policy

- You may use any resource about syntax and general programming in C (C99) or Fortran (77). You may not use additional packages.
- You should not use any books, materials from previous years or other resources discussing these specific problems.
- No AI/LLMs.
- You must mention any resource you use (even if it is a resource that you are not supposed to use!).
- It is very important that the functions you write have EXACTLY the same structure as described here. You should submit one source file. We may try to compile tests with your functions, and it would fail if it is not exactly in this form.
- Your functions should be documented, including a general description of each function, functions inputs and function outputs, and comments throughout the code. There is no need to overdocument: just make sure the code functions and code are clear enough for other students and staff to read.

Other Comments

I recommend that you solve all problems, or at least share your thoughts about them with us (you don't have to be right!).

Those of you who are not familiar with C/ Fortran should concentrate on getting the basic functionality (look for the † sign); Once you do that, you can continue to other parts if you have time. Those of you who are familiar with C would ideally get to all the different parts.

1 Powers of x

Write a function that takes x and n, where x is a real number and n is a non-negative integer, and returns x^n.

The function should look like this: `double mypow(double x, int n)` if you are using C, and `subroutine mypow(v, x, n)`, where `v` and `x` are `real *8` and `n` is `integer` if you are using Fortran (the output will be returned in v). It is very important that you use these exact structures, because we may try to call your function in our code, and if you use a different form, our test code will not compile.

1. † Implement the functions. In your implementation use a loop that multiplies by x n times. Do not use the built in powers in your implementation.
2. How would you test your function? Here are some things you can look at.
   (a) What can go wrong when you compute the powers?
   (b) How long does it take to compute high powers (try different n). How would you time that?
   (c) How does the time scale when you increase n?
   (d) Suppose you wanted to do this faster, how could you do that? (Just some ideas, you don't have to implement them)
   (e) Compare your functions to the built in powers in C or Fortran.
   (f) If you made a mistake in this question - where would it be? Report (briefly) on challenges and mistake you made while working on this question.

You may collaborate via github on this question.

2 Factorial and Binomial Coefficients

The factorial $n!$ is defined by the formula:

$$n! = n \cdot (n-1) \cdot ...2 \cdot 1, \tag{1}$$

with $0! = 1$.

The binomial coefficient $\binom{n}{k}$ ("*n choose k*") is defined by the formula:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{2}$$

1. † Implement the factorial. The fuction should look like this `int myfact(int n)` if you are using C, and `subroutine myfact(v, n)`, where `n` and `v` are `integer`s if you are using Fortran (the output will be returned in v).

2. Implement the binomial. The function should look like this `int mybinom(int n, int k)` if you are using C, and `subroutine mybinom(v, n, k)`, where `n`, `k` and `v` are `integer` if you are using Fortran (the output will be returned in v).

3. How would you test your function?

If you find it necessary to use a different function definition, give the function a different name and explain the motivation for your choice.

You may collaborate via github on this question.

3 Bubble Sort†

Suppose that we have an array of n real numbers $a_1, a_2, ..., a_n$, which we would like to sort from the smallest to the largest. Formally, we want to have a new array $a_{\sigma(1)}, a_{\sigma(2)}, ..., a_{\sigma(n)}$, which has the exact same numbers as our original array, sorted such that $a_{\sigma(k)} \leq a_{\sigma(k+1)}$ (in other words, σ is a permutation).

One sort algorithms is Bubble Sort.

Algorithm 1: Bubble Sort

```
input  : integer n, array of n real numbers A
for $j \leftarrow 1$ to $n$ do
    for $k \leftarrow n$ down to $j+1$ do
        if $A[k] < A[k-1]$ then
            Swap $A[k]$ and $A[k-1]$
return A
```

1. Analyze the Bubble Sort algorithm (terminates? timing? correctness?). Hint: What can you say about the j-th element at the end of the j-th iteration?

2. Suppose that the array is already sorted. How can you make the algorithm terminate faster? How would you prove that version?

3. It is a nice exercise in arrays to implement the Bubble Sort algorithm.

4. If you made a mistake in this question - where would it be? Report (briefly) on challenges and mistake you made while working on this question.