Assignment 2
MECHTRON 3X03: Scientific Computation

Due at 11:59 PM on Friday, November 3

Fall 2023

Submission Guidelines

There are 5 problems worth a total of 40 marks. Please read all of the files included in the Assignment 2 handout as they contain useful information. Submit the following files on Avenue:

1. A completed version of `assignment2_handout.jl` containing your solutions to Problems 1-4. Do **not** change the name of this file. Other than `LinearAlgebra`, do **not** include (via `import` or `using` or any other means) any libraries as part of your submission (they will be detected and removed by the grading script and you will receive a mark of zero). Do **not** change the name or function signatures of any of the functions you are asked to implement in Problems 1-4. You may write helper functions to simplify your solution (include these in your completed version of `assignment_handout.jl`).

2. A PDF called `assignment2.pdf` containing your plot and analysis for Problem 5 (no need to include Julia code here). This PDF can be any combination of typed/scanned/handwritten, so long as it is legible (you will receive a grade of zero on any section that cannot be understood).

The handout also contains a file called `assignment2_test.jl`. This contains some (but not all) of the code that we will be using to grade your submission. Do **not** hand this file in. You are advised to use this script to test your code and modify it while debugging.

The handout contains one last file called `assignment2_plot.jl`. You will use this code (along with your completed functions in `assignment2_handout.jl`) to produce the plot you need for Problem 5. Do **not** hand this file in. You may find it useful to modify this code while working on your solution to Problem 5, but the plot you submit should be the result of running this script without making any modifications.

Use of Generative AI Policy

If you use it, treat generative AI as you would a search engine: you may use it to answer general queries about scientific computing, but any specific component of a solution or lines of code must be cited (see the syllabus for citation guidelines).

This is an individual assignment. All submitted work must be your own, or appropriately cited from scholarly references. Submitting all or part of someone else's solution is an academic offence.

Problems

Problem 1 (10 points)**:** In `assignment2_handout.jl`, implement the following Julia function:

```
"""
Computes the coefficients of Newton’s interpolating polynomial.
Inputs
    x: vector with distinct elements x[i]
    y: vector of the same size as x
Output
    c: vector with the coefficients of the polynomial
    """
function newton_int(x, y)
    return c
end
```

The output $c \in \mathbb{R}^n$ contains coefficients such that

$$p_n(x) = c_1 + c_2(x - x_1) + c_3(x - x_1)(x - x_2) + \ldots + c_n(x - x_1)(x - x_2)\cdots(x - x_{n-1}), \tag{1}$$

where we have indexed starting with 1 to mach Julia’s convention (note that we indexed from 0 in lecture). Use the test script distributed with this assignment to check your implementation (it will be used to grade your work after submission).

Problem 2 (5 points)**:** In `assignment2_handout.jl`, implement the following Julia function:

```
"""
Evaluates a polynomial with Newton coefficients c
defined over nodes x using Horner’s rule on the points in X.
Inputs
    c: vector with n coefficients
    x: vector of n distinct points used to compute c in newton_int
    X: vector of m points
Output
    p: vector of m points
"""
function horner(c, x, X)
    return p
end
```

The output vector p should contain m elements equal to

$$p_i = c_1 + c_2(X_i - x_1) + \ldots + c_n(X_i - x_1)\cdots(X_i - x_{n-1}), \tag{2}$$

for $X_i,\ i = 1, \ldots, m$, where we have once again indexed starting with 1 to mach Julia’s convention. Note that you cannot just implement Eq. 2 naively: you must use Horner’s rule to reduce the number of floating point operations required. This function will be used with coefficients computed by `newton_int()`. Once again, use the test script distributed with this assignment to check your implementation.

Problem 3 (5 points)**:** In `assignment2_handout.jl`, implement the following Julia function:

```
"""
Computes the number of equally spaced points to use for
interpolating cos(ω*x) on interval [a, b] for an absolute
error tolerance of tol.

Inputs
    a: lower boundary of the interpolation interval
    b: upper boundary of the interpolation interval
    ω: frequency of cos(ω*x)
    tol: maximum absolute error
Output
    n: number of equally spaced points to use
"""
function subdivide(a, b, ω, tol)
    return n
end
```

Your function should return the smallest positive integer n such that the interpolating polynomial $p_{n-1}(x)$ constructed with n evenly-spaced points x_i has maximum absolute error less than `tol` for all $x \in [a, b]$; i.e.,

$$|\cos(\omega * x) - p_{n-1}(x)| \leq \texttt{tol} \ \forall x \in [a, b]. \tag{3}$$

Note that $x_1 = a$ and $x_n = b$.

Problem 4 (10 points)**:** In `assignment2_handout.jl`, implement the following Julia function:

```
"""
Computes Chebyshev nodes in the interval [a, b] for the function
cos(ω*x) for a maximum absolute error of tol.

Inputs
    a: lower boundary of the interpolation interval
    b: upper boundary of the interpolation interval
    ω: frequency of cos(ω*x)
    tol: maximum absolute error
Output
    x: distinct Chebyshev nodes in [a, b]
"""
function chebyshev_nodes(a, b, ω, tol)
    return x
end
```

Your function should return the fewest Chebyshev nodes in $[a, b]$ that ensure a maximum absolute error of `tol` for the interpolating polynomial constructed with those nodes (i.e., satisfying the bound in Eq. 3).

Problem 5 (10 points)**:** Run `assignment2_plot.jl` in the same folder as your completed `assignment2_handout.jl` file. Comment on the two strategies for selecting interpolation points x_i: do they both satisfy the absolute error bound? Which requires more points to satisfy this bound? Give a qualitative description of the Chebyshev nodes' distribution.