Assignment 3
MT 3X03 & CS/SE 4X03: Scientific Computation

Due at 11:59 PM on Thursday, December 5

Fall 2024

Submission Guidelines

Submit the following files on Avenue:

1. A PDF called `<FIRSTNAME>_<LASTNAME>_a3.pdf` (e.g., `matthew_giamou_a3.pdf`) containing all of your plots and written answers to mathematical and discussion questions (no need to include Julia code here). Show all steps in your solutions.

2. A file called `<FIRSTNAME>_<LASTNAME>_a3.jl` containing all of your Julia code solutions (we will run this with autograding scripts, so be sure to test it carefully). Use the `a3_template.jl` file we have provided as boilerplate: it contains function signatures for you to implement. Include all helper functions you implement as part of your solution in this file. **Do not use any additional `import` or `using` statements beyond what is provided in `a3_template.jl` - these will be detected and you will receive a grade of zero.**

The 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 easily understood). **Do not** submit the plotting scripts or test script provided to you. Read the syllabus to find the MSAF policy for this assignment.

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

There are 9 problems worth a total of 90 marks. Please read all of the files included in the Assignment 3 handout as they contain useful information.

Interpolation

Problem 1 (10 points)**:** Implement polynomial interpolation using the Newton form in the function signature `newton_int()` in `template_a3.jl`:

```
"""
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 `test_a3.jl` distributed with this assignment to check your implementation (it will be used to grade your work after submission).

Problem 2 (5 points)**:** In `template_a3.jl`, implement Horner's rule in the provided function header:

```
"""
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 `template_a3.jl`, implement the function header for `subdivide()`:

```
"""
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 is theoretically guaranteed to have 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 with Julia's 1-based indexing, $x_1 = a$ and $x_n = b$.

Problem 4 (10 points)**:** In `template_a3.jl`, implement the `chebyshev_nodes()` function header:

```
"""
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 `plot_interpolation.jl` in the same folder as your completed `test_a3.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.

Numerical Integration

Problem 6 (15 points)**:** Implement the composite midpoint rule, the composite trapezoidal rule, and the composite Simpson's rule in their respective function templates in `template_a3.jl`. Note that the composite Simpson's rule requires the input r to be an even number of subintervals, and that you should apply the basic Simpson's rule $r/2$ times (i.e., you can only evaluate the integrand f on $r+1$ points). These functions will be tested by `test_a3.jl` with slightly different inputs. Do not change the function signatures.

Problem 7 (10 points)**:** Run `plot_composite.jl` with the functions you implemented in Problem 1. This script compares your numerical quadrature methods with the analytical solution to

$$I_f = \int_0^{4\pi} e^{-x/2} \sin(x) = 0.798506\ldots \tag{4}$$

by plotting the error against $h = (a-b)/r$ for varying values of r. If your composite integration rules have been implemented correctly, you should see two parallel lines and a third line that is roughly piecewise linear in two sections. Use what we learned in lecture about the expressions for the error of each composite quadrature rule to explain:

a) (3 points) the slope of each line;

b) (3 points) the offset between the parallel lines; and

c) (4 points) the roughly piecewise continuous behaviour of the third line.

Note that the plot is displayed on logarithmic axes. Submit your answers to this question in your PDF submission.

Problem 8 (15 points)**:** Implement the adaptive Simpson's rule in its function header in `template_a3.jl`. This function will be tested by `test_a3.jl` with slightly different inputs. Do not change its function signature.

Problem 9 (10 points)**:** Compare your composite and adaptive Simpson's rule implementations using the script in `plot_adaptive.jl`, which computes the integral

$$T(k) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1-k^2\sin^2\theta}}, \tag{5}$$

where $T(k)$ is proportional to the period of a simple pendulum with starting angle θ_0 and $k = \sin\frac{\theta_0}{2}$. Submit both plots that this script outputs in your PDF submission. The blue curve on the first plot, which displays $T(k)$ approximated with the adaptive Simpson's rule, should look smooth over the plotted range $-1 < k < 1$. The red curve plots the integrals computed by the composite rule using the same number of points as the adaptive rule needed to achieve approximately satisfy the tolerance provided.

a) (4 points) Why do the integrals computed by the composite Simpson's rule differ from the adaptive solution? Which do you think is more accurate?

b) (6 points) This script outputs a second plot which compares the "nodes" used by the adaptive and composite rules for $k = 0.99$. Describe the distribution of nodes for each method. How does the distribution of these nodes support your answer to part a)?