MTH 373/573:
Scientific Computing

Homework 4

IIIT-Delhi, Monsoon 2022
Kaushik Kalyanaraman

Due by: December 9, 2022 (Friday) by 23:59 hours IST

Total: 70 points

Guidelines

This assignment should be answered individually.

- You are not allowed to discuss specifics of your solutions, solution strategies, or coding strategies with any other student in this class.
- Please consult with and carefully read through IIIT-Delhi's Academic Dishonesty Policy.

Please don't be a cheater!

- You shall submit your solutions via Google Classroom – there are no printed/handwritten/-paper submissions. Please type, take a picture, or scan your solutions and upload them as a single PDF file and only a single PDF file. You can find many websites that will merge your scanned images into a PDF file or, for the nerdy types, you can find a command line script to do this.
- You may post queries relating to any HW question to `#discussions` on the class's Slack channel.
- Start working on your solutions early and don't wait until close to due date.
- Each problem should have a clear and concise write-up.
- Please clearly show all steps involved in your solution.
- You will need to write a separate code for each computational problem and sometimes for some subproblems too. You should name each such file as `problem_`*n*`.py` where *n* is the problem number. For example, your file names could be `problem_5.py`, `problem_6a.py`, `problem_6b.py`, and so on.
- *Python tip:* You can import `Python` modules as follows:

```
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import numpy.linalg as npla
import scipy.linalg as spla
```

- We shall only accept `*.py` `Python` files – no Jupyter Notebook submissions are allowed. Please write your code for the machine problems in a text editor and generate appropriate output via Terminal using `Python` (directly) or `IPython` (recommended).

Problem 1: Polynomial Interpolation — 4 × 1.5 = 6 points

Provide the degree 3 interpolaing polynomial for the data points:

$$\{(-2, 15), (0, -1), (1, 0), (3, -2)\},$$

(a) using the monomial basis.

(b) using the Lagrange basis.

(c) using the Newton basis.

(d) Show that the three representations give the same polynomial by expressing each polynomial in the form $a_0 + a_1 x + a_2 x^2 + a_3 x^3$ and comparing coefficients

Problem 2: Numerical Quadrature — 4 × (1 + 1) = 8 points

You are given that:

$$\int_0^1 \frac{1}{1+x^2} dx = \frac{\pi}{4} \qquad \text{and} \qquad \int_0^1 \sqrt{x} \log x dx = -\frac{4}{9}.$$

Compute each of these integrals using the midpoint, trapezoid, Simpson's and 2-point Gaussian quadrature rules as discussed in the class notes. (*Note*: You have to compute these "by hand" and not using a computer although you can perform evaluations using Python.)

Note: For changing the limits of integration and hence the quadrature rule, if we wish to use a quadrature rule that is tabulated on the interval $[\alpha, \beta]$,

$$\int_\alpha^\beta f(x)dx \approx \sum_{i=1}^n w_i f(x_i),$$

to approximate an integral on the interval $[a, b]$,

$$I(f) = \int_a^b f(t)dt,$$

then we must use a change of variable from x in $[\alpha, \beta]$ to t in $[a, b]$. Many such transformations are possible, but a simple linear transformation:

$$t = \frac{(b-a)x + a\beta - b\alpha}{\beta - \alpha}$$

has the advantage of preserving the degree of the quadrature rule. The integral and its quadrature

are then given by:

$$I(f) = \frac{b-a}{\beta-\alpha}\int_{\alpha}^{\beta} f\left(\frac{(b-a)x + a\beta - b\alpha}{\beta-\alpha}\right)dx$$
$$= \frac{b-a}{\beta-\alpha}\sum_{i=1}^{n} w_i f\left(\frac{(b-a)x_i + a\beta - b\alpha}{\beta-\alpha}\right).$$

Problem 3: Interpolating Runge's Function — 24 points

Compute (a) polynomial, and (b) cubic spline interpolants to Runge's function:

$$f(t) = \frac{1}{1+25t^2},$$

using $n = 11, 21$ equally spaced points on the interval $[-1, 1]$. Compare your results graphically by plotting for each case the interpolants and the original functon for each value of n.

Note 1: You have to write your own functions/code to compute the polynomial and spline interpolants and not use any library APIs like those in `scipy.interpolate`.

Note 2: In order to plot the Runge's function and the interpolating polynomials, you will have to evaluate them over more points than simply the interpolation points. For example, you can use the code below to plot the Runge's function. You should similarly plot the interpolating polynomials.

```
def f(t):
    return 1/(1 + 25 * t**2)

T = np.linspace(-1, 1, 100)
f_at_T = np.array([f(t_) for t_ in T])
plt.plot(T, f_at_T)
plt.grid(True)
plt.xlabel('t', fontsize=12)
plt.ylabel('f(t)', fontsize=12)
plt.title('Plot of Runge\'s function', fontsize=14)
```

Problem 4: Composite Gaussian Quadrature — 20 points

In this problem, you shall understand how to extend the 2-point Gaussian quadrature rule we have studied in class to a more realistic computational scenario by doing *composite quadrature*.

For this, we suppose that the interval $[a, b]$ is subdivided into an even number of subintervals, say, n each of width $h = (b-a)/n$. Then, the partition points are $x_i = a + ih$ for $0 \le i \le n$ where n is

divisible by 2. Now, from basic calculus, we have that:

$$\int_a^b f(x)dx = \sum_{i=1}^{n/2} \int_{a+(2(i-1)h}^{a+2ih} f(x)dx \approx \sum_{i=1}^{n/2} Q(f)\Big|_{a+(2(i-1)h}^{a+2ih},$$

where the quadrature $Q(f)$ is over the specified interval. You can use the change of variables specified in Problem 2 to compute these integrals.

Write a Python code to implement composite quadratue. Test your code to evaluate the following integral using $n = 2, 4, \ldots, 64$:

$$\int_1^3 \frac{100}{x} \sin\left(\frac{10}{x}\right) dx.$$

The best estimate for this integral is -18.79829683678703 (for example, via Wolfram Alpha). Compute also the relative error for each value of n.

Problem 5: Numerical Differentiation — 12 points

Using the forward difference approximation of the derivative of a function $f(x)$ at $x = x_0$ as being:

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h},$$

compute the approximation of the derivative of $f(x) = e^{-\sin(x^3)/4}$ at $x = 1$ for $h = 10^{-1}, 10^{-2}, \ldots, 10^{-15}$. For this function, the exact derivative is given by the expression:

$$f'(x) = -\frac{3}{4}x^2 \cos(x^3) e^{-\sin(x^3)/4}.$$

Compute therefore also the absolute error in your approximation for the derivative for each h, and plot this absolute error on a log-log plot. Does this error go to 0 as h becomes smaller? Explain your observations based on your understanding of floating point errors as we studied them in the first chapter of this course.

Note: Going over Example 1.3 in [Heath, 2018] should help you answer this.