DS-288 Numerical Methods
UE-201 Introduction to Scientific Computing
Due date: October 8, 2023 (Sunday 11:59 PM)

Homework-3

Total 100 points

Weight 10%

Read the following instructions carefully.

- Write your NAME and SR. NUMBER on the first page of the report(only one PDF for all questions in order). Start each question on a new page.
- In coding exercises, also give algorithm/background theory along with code and discuss attached plots briefly to get full credit for that question.
- LaTeX is recommended for the report. Use Python/Matlab for coding. Give proper annotations and comments in code wherever required. Name code file according to question number (i.e., q1,q2,q4...).
- Put all codes and the report in a folder, with the folder named **DS288_LastFiveDigitsSRNo_Name**, compress it into a zip file, and submit that zip file to the teams.
- Don't use any inbuilt functions for solving problems (i.e., np.linalg.solve, np.gradient, scipy); use a proper algorithm to get credits. Marks will be deducted if plagiarism is found in the report or codes. Late submissions won't be accepted.

Answer **All** Questions.

1. (a) **Programming Exercise:** The Mandelbrot set is a famous and visually stunning fractal pattern. The Mandelbrot set can be generated using FPI on complex numbers. To determine whether a complex number $\mathbf{c}$ belongs to the Mandelbrot set, you perform an iterative calculation starting with $\mathbf{z_0 = 0 + 0j}$ for each point in the complex plane. The formula for the iteration is $\mathbf{z_{n+1} = z_n^2 + c}$, where $\mathbf{z_n}$ is the complex number at the nth iteration. If $\mathbf{z_n}$ diverges in these iterations then $\mathbf{c}$ doesn't belong to the Mandelbrot set. Follow the following instructions to generate the Mandelbrot set.
   i. Generate a complex plane by dividing the X-axis (-2.5,1) and the Y-axis (-1.5,1.5) into 4000 equally spaced points.
   ii. Call Mandelbrot function using complex numbers, generated by each point in the above set of the X-axis and the Y-axis.
   iii. Define Mandelbrot function that takes a complex number c and iterates $z_{n+1} = z_n^2 + c$, starting with $z_0 = 0$.
   iv. Use **100** iterations for above steps. If $|z_n| > 2$, then c doesn't belong to the Mandelbrot set. In such case stop iterating and return the number of iterations it took so far.

...

v. If $|z_n| < 2$ for all 100 iterations then return max no. of iterations that is 100.

Follow the attached code snippet on the last page.

(b) In a similar way NR can be used to generate Newton's fractal. The way you worked in part(a), following instructions, will lead you to generate Newton's fractal.

i. Consider $f(z) = z^3 + 1$, it has three roots $-1, e^{i\pi/3}, e^{-i\pi/3}$.

ii. Generate a complex plane by dividing the X-axis (-2,2) and the Y-axis (-2,2) into 3000 equally spaced points.

iii. Call Newton's function using complex numbers c generated by the above set of x and y axes as an initial guess.

iv. If NR converges to -1, return 0, if it converges to $e^{i\pi/3}$, return 1 and if it converges to $e^{-i\pi/3}$, return 2.

v. Use absolute error between z_n and the actual root as stopping criteria, with tolerance 10^{-6}.

Give imgplot for these fractals. Briefly write your observations and things that fascinated you while doing these exercises.

[10+10]

2. **Theory Exercise:** Let f be a real-valued function on an interval J and $x_0, \ldots, x_n$ be $n+1$ distinct points in J. Suppose $p_k \in \mathcal{P}_k$ denotes the Lagrange interpolating polynomial for f at the first $k+1$ points $x_0, \ldots, x_k$. [20]

(a) Prove that $p_n(x) - p_{n-1}(x) = c\,(x - x_0)\cdots(x - x_{n-1})$ for some constant c. Let us call this constant the n-th divided difference of f at the x_i and denote it by $f\,[x_0, \ldots, x_n]$.

(b) Prove that $f\left[x_{\sigma(0)}, \ldots, x_{\sigma(n)}\right] = f\,[x_0, \ldots, x_n]$ for any σ that permutes the set $\{0, 1, \ldots, n\}$. In other words, show that permuting the interpolation points does not change the divided difference value.

(c) Show that

$$p_n(x) = f\,[x_0] + f\,[x_0, x_1]\,(x - x_0) + \cdots + f\,[x_0, \ldots, x_n]\,(x - x_0)\cdots(x - x_{n-1})\,.$$

Prove that the error is given as

$$f(x) - p_n(x) = f\,[x_0, \ldots, x_n, x]\,(x - x_0)\cdots(x - x_n)\,.$$

(d) If $f \in C^n(J)$, show that there exists a point ξ in the interior of J such that

$$f\,[x_0, \ldots, x_n] = \frac{1}{n!} f^{(n)}(\xi).$$

...

3. **Programming Exercise:** It is suspected that the high amounts of tannin in mature oak leaves inhibit the growth of the winter moth (Operophtera bromata L., Geometridae) larvae that extensively damage these trees in certain years. The following table lists the average weight of two samples of larvae at times in the first 28 days after birth. The first sample was reared on young oak leaves, whereas the second sample was reared on mature leaves from the same tree. [30]

Day	0	6	10	13	17	20	28
Sample I average weight (mg)	15.67	20.33	30.67	25.33	35.10	30.31	28.5
Sample 2 average weight (mg)	15.67	17.11	20.89	15.00	10.56	10.44	8.5

(a) Use a natural cubic spline to approximate the average weight curve for each sample and fill in the following table. Compare the spline interpolant and Lagrange polynomial interpolation by plotting from day 0 to 28 (in a gap of **12hrs**).
To solve the system of equations, you can use crout factorization as tridiagonal solver(Use algorithm 3.4 given in the textbook).

	Sample 1				Sample 2			
x_i	a_i	b_i	c_i	d_i	a_i	b_i	c_i	d_i
0			0				0	
6								
10								
13								
17								
20								

(b) Find an approximate **maximum average weight and corresponding day** for each sample by determining the maximum of the spline. Use forward divided difference with step size 10^{-4} days for derivatives.
Don't infer any information from plots and the data table of part (a). You need to report the answers up to 2 decimal places.

4. **Programming Exercise:** A car traveling along a straight road is clocked at a number of points. The data from the observations are given in the following table, where the time is in seconds, the distance is in feet, and the speed is in feet per second. [30]

Time	0	3	5	8	13
Distance	0	225	383	623	993
Speed	75	77	80	74	72

...

(a) Use a Hermite polynomial to predict the position of the car and its speed when $t = 10$ s. You can use the divided difference(take step size 10^{-4}) to find the approximate derivative of the above polynomial for estimating speed at $t = 10$. Plot this Hermite polynomial from $t = 0s$ to $t = 13s$ (take a gap of $1s$).

(b) Use the derivative of the Hermite polynomial to determine whether the car ever exceeds a 55mi/h speed limit on the road. If so, what is the first time the car exceeds this speed?

(c) What is the predicted maximum speed for the car?

Note: Use algorithm 3.3 from the textbook.

Snippet for Q1

Python

```
# Set the resolution
n = 4000
X = np.linspace(-2.5,1,n)
Y = np.linspace(-1.5,1.5,n)

# Compute the Mandelbrot set
# Write your definition for mandalbrot(c)
MandalbrotSet = np.array([mandalbrot(x + 1j*y) for x
in X for y in Y])

MandalbrotSet = MandalbrotSet.reshape(n,n)

# Display the Mandelbrot set
fig = plt.figure(figsize=(10,10))

# Try different cmap
plt.imshow(MandalbrotSet.T, cmap='binary', extent =
[-2.5,1,-1.5,1.5])
plt.savefig('Mset.png')
plt.show()
```

MatLab

```
% Set the resolution
n = 4000;
X = linspace(-2.5, 1, n);
Y = linspace(-1.5, 1.5, n);

% Compute the Mandelbrot set
% Write your definition for mandalbrot(c) in
mandalbrot.m
MandelbrotSet = zeros(n, n);
for x = 1:n
   for y = 1:n
      c = X(x) + 1i*Y(y);
      MandelbrotSet(x, y) = mandalbrot(c); % Call the
mandalbrot function
   end
end

% Display the Mandelbrot set
figure;
imagesc(X, Y, MandelbrotSet');
colormap('jet');
axis equal;
axis([-2.5 1 -1.5 1.5]);
colorbar;
title('Mandelbrot Set');
```

*Codes may take 1-3 min of execution time, depending on your machine.

***Additional for python users:** Use **numba** to speed up execution.

*from **numba** import **njit***
write **@njit** just before **mandalbrol(c)** definition
i.e.,

```
@njit
def mandalbrot(c):
```

Notice speedup yourself after using this **just in time** compiler

To learn more about ***MANDELBROT***.

https://youtu.be/NGMRB4O922I?si=1RqxMftN_nO9oZGe