DS-288 Numerical Methods
UMC-202 Introduction to Numerical Methods
Due date: November 16, 2025 (11:59 PM)

Homework-4

Total 200 points

Weightage 20%

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.
- Only LaTeX version of the report is **acceptable** (find the attached demo LaTeX file). Use Python/Matlab for coding. Give proper annotations and comments in code wherever required.
- Submit your report as a single PDF named **LastFiveDigitsSRNo_Name.pdf**.(e.g 12345_Ram_charan.pdf)
- For each question, give a separate code file `.ipynb`, i.e., for Question 1 name **LastFiveDigitsSRNo_q1.ipynb**.(e.g 12345_q1.ipynb)
- Upload both the **PDF** and **.ipynb** files to Teams **(without zipping)**.
- Don't use any inbuilt functions for solving problems unless specified; 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.
- Using LLM's for writing the report is strictly prohibited.

1. Theoretical Derivation of Order of Convergence [30]

The asymptotic order of convergence α for a root-finding method generating the sequence $x_0, x_1, x_2, \ldots$, with error $e_n = x_n - x_{\text{true\_root}}$, is defined by,

$$\lim_{n\to\infty} \frac{e_{n+1}}{(e_n)^\alpha} = \lambda$$

where λ is the asymptotic error constant. Derive the asymptotic order of convergence α for the following root-finding methods,

(a) Secant method for the root of multiplicity m.

(b) Muller's method for simple root.

NOTE: Write the assumptions made and references properly. No partial marks will be provided in this question.

2. Chaotic world of fractals [40]

In this exercise, you need to generate following fractals as per given instruction and briefly write your observations and things that fascinated you while doing these exercises.

(a) **Mandelbrot Set:** The Mandelbrot set is a famous and visually stunning fractal pattern. The Mandelbrot set can be generated using **fixed-point iteration** 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.

- Implement a function `mandelbrot(c, max_iter=100)` that:
  - Iterates $z_{n+1} = z_n^2 + c, \quad$ with $z_0 = 0$.
  - Stops when $|z_n| > 2$ and returns the iteration count.
  - Returns `max_iter=100` if the point does not diverge.
- **Visualization:**

- Generate a grid of 1000×1000 points with

$$x \in [-2.5,\, 1], \qquad y \in [-1.5,\, 1.5].$$

- Evaluate the Mandelbrot iteration for each grid point and generate a well labeled plot of the result using `imshow` (or equivalent, play with different cmap).

(b) **Newton's Fractal:** The Newton–Raphson (NR) method can also be used to generate a fascinating class of fractals known as Newton's fractals. This fractal arises when the NR method is applied over the complex plane, where each initial point z_0 is iteratively refined toward one of the roots of a nonlinear equation. Depending on the starting value, the iteration converges to different roots, forming distinct regions known as *basins of attraction*. The boundary between these basins exhibits a fractal structure.

- Implement a function `newton_fractal(z0, tol=1e-6)` that:
  - Considers the function $f(z) = z^3 + 1$ and iterates according to the Newton–Raphson method.
  - Stops when the absolute error between z_n and the nearest true root is less than the tolerance 10^{-6}.
  - Returns an integer flag to identify the converged root:
    - Return `0` if NR converges to -1.
    - Return `1` if NR converges to $e^{i\pi/3}$.
    - Return `2` if NR converges to $e^{-i\pi/3}$.
- **Visualization:**
  - Generate a grid of 1000×1000 points over

$$x \in [-2,\, 2], \qquad y \in [-2,\, 2].$$

  - For each complex point $z_0 = x + iy$, apply the Newton iteration using the function defined above and generate a well labeled plot of the result that displays the *basins of attraction*.

(c) **Logistic Map and the Onset of Chaos:**

The logistic map is a simple yet powerful model that illustrates how complex, chaotic behavior can arise from a deterministic nonlinear system. It was first introduced as a population growth model, but it has since become a cornerstone example in chaos theory. The logistic map is defined recursively by

$$x_{n+1} = A\, x_n(1 - x_n), \qquad 0 \le x_n \le 1,$$

where A is a control parameter and x_n represents the state at iteration n. For small values of A, the sequence $\{x_n\}$ settles to a stable fixed point. As A increases, the system undergoes successive *period-doubling bifurcations* and after some iterations it becomes chaotic, where the values of x_n appear random but are still governed by a deterministic rule.

A key quantity that characterizes chaotic dynamics is the **Lyapunov exponent** (λ), which measures the average exponential rate at which nearby trajectories diverge. For a discrete map $x_{n+1} = f(x_n)$, small perturbations evolve according to the local stretching factor $|f'(x_n)|$. The finite-time Lyapunov exponent is defined as

$$\lambda_n = \frac{1}{n}\sum_{k=0}^{n-1} \ln\left|f'(x_k)\right|,$$

which approaches a constant value λ as $n \to \infty$. A positive Lyapunov exponent indicates *sensitive dependence on initial conditions*, the defining feature of chaotic behavior.

- Implement a function `logistic_map(A, x0, n_iter)` that:
  - Iterates the map $x_{n+1} = A\,x_n(1 - x_n)$ for a given A and initial condition x_0.
  - Returns the sequence $\{x_n\}$ for $n = 1, 2, \ldots, n_{\text{iter}}$.
- **Bifurcation Diagram:**
  - Vary the control parameter A from 0.89 to 3.995 in increments of 0.0125.
  - For each A, iterate the logistic map for 200 steps, discarding the first 15 iterations to eliminate transients.
  - Plot the remaining values of x_n versus A to obtain the *bifurcation diagram*, illustrating the transition from periodic to chaotic behavior.
- **Lyapunov Exponent and Sensitivity to Initial Conditions:**
  - Fix $A = 4$, choose an initial value x_0 and iterate the logistic map.
  - Compute and plot the finite-time Lyapunov exponent as a function of n.
  - Report λ_n for large n, Observe whether λ_n approaches a constant value (appearing as a straight line).

For those who are interested in knowing more about chaos and fractals: Link 1 Link 2

3. Optimizing a multi-link robotic arm [40]

In many engineering applications, we often encounter optimization problems where an explicit analytical solution is difficult or impossible to obtain. Such cases arise in robotic control, fluid dynamics, machine learning, and nonlinear systems. When closed-form equations cannot be inverted, **iterative numerical optimization methods** such as *Gradient Descent* provide an efficient way to find approximate solutions.

Gradient descent is an iterative algorithm that updates parameters in the direction of steepest descent of a cost (or loss) function until convergence. Unlike methods such as Newton–Raphson, which require explicit computation of higher-order derivatives (Hessian matrices), gradient descent relies only on first-order derivative information and is therefore computationally simpler and more stable for high-dimensional or nonlinear systems.

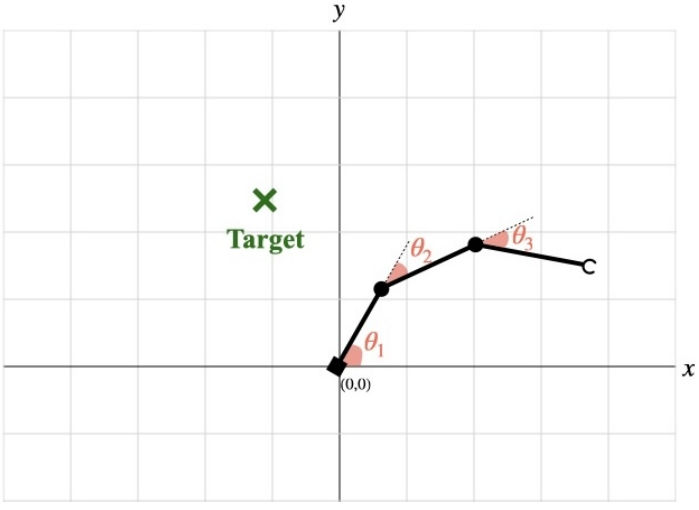

Figure 1: A 3-linked Robotic Arm fixed at origin.

Problem Description:

Consider a planar robotic manipulator that consists of three revolute joints with each arm length $l_1 = l_2 = l_3 = 1.0\,\text{m}$. The joint angles are denoted as $\boldsymbol{\theta} = [\theta_1, \theta_2, \theta_3]^T$. The end-hook position (x, y) in Cartesian coordinates is determined by the forward kinematics:

$$\begin{aligned} x &= l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3), \\ y &= l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3) \end{aligned} \tag{1}$$

...

The objective is to move the arm such that its end-hook reaches a fixed target position:

$$\mathbf{p}_t = \begin{bmatrix} -0.8 \\ 1.2 \end{bmatrix} m.$$

We define the loss function (or cost function) as the squared distance between the current end-hook position and the target:

$$L(\boldsymbol{\theta}) = \frac{1}{2}\|\mathbf{p}(\boldsymbol{\theta}) - \mathbf{p}_t\|^2.$$

Tasks:

(a) Derive the coordinate Equation 1 with the help of the schematic Figure 1.

(b) Using the concepts of numerical differentiation discussed in class, compute the gradient of $L(\boldsymbol{\theta})$ with respect to each θ using the **central difference method** with a step size of $\mathbf{h} = \mathbf{10^{-5}}$.

(c) Implement the **Gradient Descent** method to iteratively minimize $L(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \nabla L(\boldsymbol{\theta}_k),$$

where the learning rate is $\alpha = 0.01$
The process should continue until the Euclidean distance between the end-hook and the target is less than 0.01 m.

(d) Extend your implementation to include **Gradient Descent with Momentum**, defined as:

$$\mathbf{v}_{k+1} = \beta \mathbf{v}_k + \alpha \nabla L(\boldsymbol{\theta}_k),$$
$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mathbf{v}_{k+1},$$

with momentum coefficient $\beta = 0.9$.

(e) Compare and discuss the performance of both methods:

- Report the number of iterations required for convergence.
- Provide the final joint angles $[\theta_1, \theta_2, \theta_3]$ (in degrees).
- Plot the end-hook paths for both methods on the same figure.
- Write your observations on the effect of the momentum on optimization.

Initial conditions:

$$\boldsymbol{\theta}_0 = \begin{bmatrix} 0.2 \\ 0.1 \\ -0.3 \end{bmatrix} rad, \qquad \mathbf{v_o} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

4. Fourier Series Reconstruction of Letter Outlines [40]

The representation of periodic functions through sums of sinusoidal components is made possible by Fourier series. This question explores how complex Fourier series can be used to represent and reconstruct the outlines of letters.

Any periodic function can be expressed as a combination of sine and cosine functions with different frequencies. This classical form of the Fourier series can be written as:

$$f(x) = a_0 + \sum_{k=1}^{\infty} \left[a_k \cos\left(\frac{2\pi kx}{T}\right) + b_k \sin\left(\frac{2\pi kx}{T}\right) \right],$$

where a_k and b_k are real coefficients that must be computed using separate integrals for each coefficient.

To avoid calculating multiple integrals for both sine and cosine components, we employ the **complex Fourier series** representation using **Euler's identity**:

$$e^{i\theta} = \cos(\theta) + i\sin(\theta).$$

The expression of any periodic function with period T as an infinite sum of complex exponentials is therefore possible. This expression, called the **Fourier series**, takes the form:

$$z(t) = \sum_{k=-\infty}^{\infty} c_k e^{i2\pi kt/T}, \tag{2}$$

where the **Fourier coefficients** c_k quantify the contribution of each complex exponential (frequency component) to the function $z(t)$.

The calculation of Fourier coefficients uses the integral:

$$c_k = \frac{1}{T} \int_0^T z(t) e^{-i2\pi kt/T}\, dt. \tag{3}$$

When we normalize the period to $T = 1$ for simplicity, this formula simplifies to:

$$\boxed{c_k = \int_0^1 z(t) e^{-i2\pi kt}\, dt.} \tag{4}$$

Since computing infinitely many coefficients is impractical, we use a finite number of modes to approximate the function. This approximation, called the **truncated Fourier series**, is expressed as:

$$\boxed{z_M(t) = \sum_{k=-M}^{M} c_k e^{i2\pi kt},} \tag{5}$$

...

where the **mode number** is denoted by M. Finer details of the original curve are captured by higher values of M. As M increases, the approximation error decreases, though the convergence rate depends on how smooth the underlying function is.

Part 1: Orthogonality Question [Theoretical] Consider a set of complex exponential functions be $\{e^{i2\pi kt}\}_{k\in\mathbb{Z}}$ defined on the interval $t \in [0, 1]$. Consider the inner product

$$\langle f, g\rangle = \int_0^1 f(t)\,\overline{g(t)}\,dt,$$

where $\overline{g(t)}$ denotes the complex conjugate of $g(t)$.

Prove that the set $\{e^{i2\pi kt}\}_{k\in\mathbb{Z}}$ is orthogonal with respect to this inner product. That is, show that for any integers k and m:

$$\langle e^{i2\pi kt}, e^{i2\pi mt}\rangle = \begin{cases} 1, & \text{if } k = m, \\ 0, & \text{if } k \neq m. \end{cases}$$

Part 2: Data Loading and Preprocessing Load the outline points for the **initial capital letter** of your name (i.e., for Shubham, use 'S') and convert them to a complex representation. For discrete samples (x_j, y_j) where $j = 0, 1, \ldots, N-1$, construct the complex values as $z_j = x_j + i \cdot y_j$. Parameter values corresponding to these samples are:

$$t_j = \frac{j}{N}, \quad j = 0, 1, \ldots, N-1.$$

To get parametric values for your initial letter, follow the instructions given in: `/Supplementary/Generator/ReadMe`

Part 3: Fourier Coefficient Calculation Implement a function to compute the Fourier coefficients by approximating the continuous formula in **Equation 4 numerically, using any numerical integration method of your choice with a proper explanation of your selection.** The frequency composition of the letter curve is represented by the set of coefficients c_k for $k \in \{-M, -M+1, \ldots, M-1, M\}$.

Part 4: Curve Reconstruction From the computed Fourier coefficients, implement a function to reconstruct the letter outline using the **Synthesis Equation 5**: Evaluate this reconstruction at $N = 1000$ equally spaced points $t \in [0, 1)$ to obtain the reconstructed coordinates.

Part 5: Visualization The improvement in reconstruction with increasing mode number should be visualized by creating a figure with five subplots that compare reconstructions for:

$$M \in \{2, 8, 32, 64, 128\}.$$

Each subplot should display both the **original curve**, plotting, and the **reconstructed curve**. For reference see `/Supplementary/Outputs`.

Part 6: Animation Animate the reconstruction of the letter by showing the evolution of each frequency component with $n : 0 \to N$. A sample animation is given in `/Supplementary/Outputs`.

5. Damped, Driven Harmonic Oscillator [50]

The motion of many physical systems, from mechanical pendulums to electrical circuits, can be modeled by second-order ordinary differential equations. This question explores the numerical solution of a damped, driven harmonic oscillator, a classic problem in physics and engineering.

The equation of motion for this system, derived from Newton's second law, is:

$$\frac{d^2x}{dt^2} + 2\gamma\frac{dx}{dt} + \omega_0^2 x = A_0 \cos(\omega t)$$

where:

- $x(t)$ is the displacement from equilibrium.
- γ is the damping coefficient, representing energy dissipation.
- ω_0 is the natural angular frequency of the undamped system.
- $A_0 \cos(\omega t)$ represents the external driving force per unit mass.

Problem Parameters You are tasked with solving the motion of an oscillator with the following parameters and initial conditions:

- Natural angular frequency (in rad/s): $\omega_0 = 2\pi$
- Damping coefficient (in s^{-1}): $\gamma = 0.5$
- Driving acceleration amplitude (in m/s^2): $A_0 = 2.0$

- Driving angular frequency (in rad/s): $\omega = 2.0$
- Initial position (at $t = 0$): $x(0) = 1.0$ m
- Initial velocity (at $t = 0$): $\frac{dx}{dt}\big|_{t=0} = 0.0$ m/s

Part 1: Numerical Implementation :

- Use **Euler's method and the Fourth-Order Runge-Kutta method** and implement a function in Python/Matlab to solve the given second-order ODE over the time interval from $t = 0$ **to** $t = 30$ **seconds**.
- Write the equations/algorithm used in these methods in your report.
- Use a time step of $\Delta t = 0.01$ s. Your function should return the time and corresponding displacement arrays.
- Tabulate the time t, displacement $x(t)$ and velocity $\frac{dx}{dt}$ at each **500th** time step.

Part 2: Visualization of Solution : Generate plots of the oscillator's displacement $x(t)$ **versus time** t for both the methods used. The plots must be clearly labeled with a title, axis labels, and appropriate units. Ensure the plot clearly shows the evolution of the motion over the full time interval.

Part 3: Analysis of Motion : Examine your plot from Part 2.

- Identify and label the initial **transient behavior**, where the motion is influenced by the starting conditions.
- Identify and label the final **steady-state behavior**, where the oscillator settles into a regular pattern.
- Explain which term in the governing ODE is responsible for the decay of the transient motion.

Part 4 : Convergence analysis : Read this part carefully.

- Take different time step values, $\Delta t = 0.0025, 0.005, 0.01, 0.02, 0.04, 0.08$ s
- **Take the Runge-Kutta method of order 4, with time step,** $\Delta t = 0.00125s$ **as the true solution.**

- Taking the above as true solution, obtain the L2-error (root mean square error) for all the 6 time steps given above, for both **Euler's and RK 4 method**. **Ensure** you take the errors at time steps which are present in both your test method and the true solution.
- Plot the L2-errors vs time step, in a log-log scale. Plot a line of best fit for your data. State the slope of this line in your report and display its value on the graph.
- Analyse the order of convergence of both the methods, and report which one is better with proper justifications.