Assignment 9

Numerical Methods: RK Method, Shooting Method and Boundary Value Problems

Sahil Raj
2301CS41

November 22, 2025

Contents

Problem 1: Numerical Solution of the Duffing Oscillator and Phase Space Plot using 4th Order Runge-Kutta

Problem Statement

The objective of this problem is to **numerically solve** the **Duffing oscillator** equation:

$$m\frac{d^2x}{dt^2} + b\frac{dx}{dt} + kx + lx^3 = F_0\cos(\omega t)$$

with parameters: $m = 1$, $k = -1$, $b = 0.2$, $l = 1$, $F_0 = 0.3$, $\omega = 1.2$. The goal is to compute the time evolution of $x(t)$ and $v(t) = \frac{dx}{dt}$, and plot the phase space diagram using the **fourth-order Runge-Kutta (RK-4) method**.

Methodology

Fourth-Order Runge-Kutta Method

The Duffing oscillator is a second-order ODE. We rewrite it as a system of two first-order ODEs:

$$\begin{aligned}\frac{dx}{dt} &= v \\ \frac{dv}{dt} &= \frac{1}{m}\big(-kx - lx^3 - bv + F_0\cos(\omega t)\big)\end{aligned}$$

The RK-4 method approximates the solution iteratively:

$$\begin{aligned}k_1 &= f(t_n, y_n)\,dt \\ k_2 &= f(t_n + \frac{dt}{2}, y_n + \frac{k_1}{2})\,dt \\ k_3 &= f(t_n + \frac{dt}{2}, y_n + \frac{k_2}{2})\,dt \\ k_4 &= f(t_n + dt, y_n + k_3)\,dt \\ y_{n+1} &= y_n + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}\end{aligned}$$

Here $y_n = [x_n, v_n]^T$ and f represents the system of derivatives.

Implementation Steps

1. Define the initial conditions: $x(0) = 1$, $v(0) = 0$.
2. Specify constants: m, k, l, b, F_0, ω.
3. Set time step $\Delta t = 0.1$ and time range $t \in [0, 64]$.
4. Iterate the RK-4 scheme to compute $x(t)$ and $v(t)$ for each time step.
5. Plot:
    - $x(t)$ vs t
    - $v(t)$ vs t
    - Phase space: $v(t)$ vs $x(t)$

Results

The numerical solution produces the following behavior:

- The displacement $x(t)$ shows a non-linear oscillatory pattern due to the cubic term.
- The velocity $v(t)$ follows a corresponding non-linear trajectory.
- The phase space diagram exhibits a **limit cycle**, typical of a Duffing oscillator under periodic forcing.

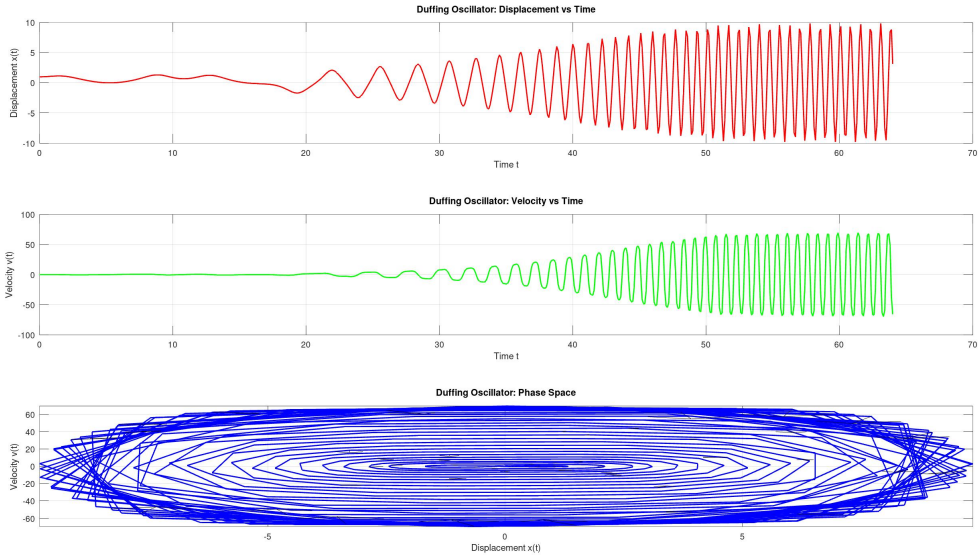

Figure 1.1: Time evolution of $x(t)$, $v(t)$, and phase space diagram v vs x for the Duffing oscillator.

Conclusion

The 4th-order Runge-Kutta method accurately integrates the Duffing oscillator equation, capturing both the non-linear dynamics and the characteristic limit cycle in phase space. This method provides stable and precise results for moderately stiff non-linear ODEs with relatively large time steps.

Problem 2: Temperature Evolution in a Rod: Solving the 1D Heat Equation

Problem Statement

The goal of this problem is to **numerically solve the one-dimensional heat equation**

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

on a rod of length $L = 1$ meter with thermal diffusivity $\alpha = 0.01$. The domain is $x \in [0, 1]$, $t \geq 0$.

The boundary and initial conditions are:

$$u(0, t) = 0, \qquad u(1, t) = 0,$$

$$u(x, 0) = \sin(\pi x).$$

Methodology

Finite Difference Discretization

The heat equation is discretized using the **explicit forward-time, central-space (FTCS)** finite difference scheme:

$$u_i^{n+1} = c\, u_{i+1}^n + c\, u_{i-1}^n + (1 - 2c)\, u_i^n,$$

where:

$$c = \frac{\alpha\, \Delta t}{\Delta x^2}$$

and:

$$u_i^n \approx u(x_i, t_n).$$

For this simulation:

$$\Delta x = 0.05, \quad \Delta t = 0.1, \quad c = \frac{0.01 \cdot 0.1}{(0.05)^2} = 0.4.$$

The scheme is stable because $c \leq 0.5$.

Implementation Steps

1. Discretize the spatial domain using $x_i = 0 : 0.05 : 1$.
2. Discretize time using $t_n = 0 : 0.1 : 32$.
3. Initialize the solution using $u(x, 0) = \sin(\pi x)$.
4. Apply boundary conditions: $u(0, t) = 0$ and $u(1, t) = 0$.
5. Use the FTCS formula to compute the temperature for each time step.
6. Visualize the evolution using a surface plot.

Results

The numerical solution shows the diffusion of the initial heat profile over time:

- The sinusoidal temperature profile smooths out as time progresses.
- Heat dissipates toward the boundaries due to the fixed boundary temperatures.
- The temperature approaches zero everywhere for large t, consistent with physical expectation.

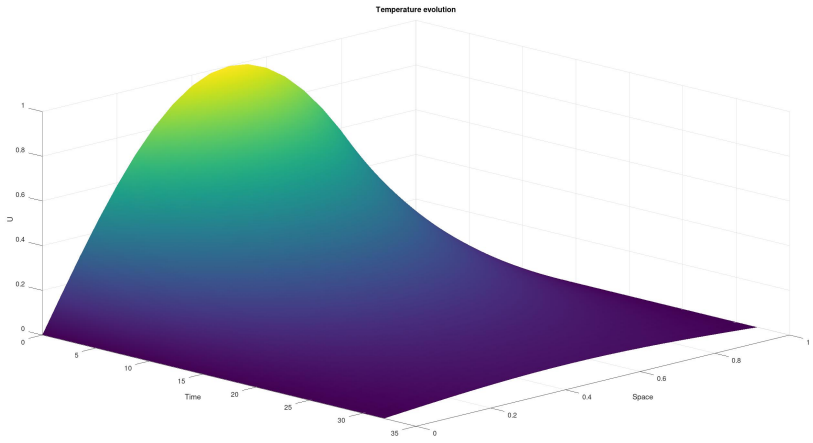

Figure 2.1: Temperature evolution $u(x, t)$ for the 1D heat equation using the FTCS method.

Conclusion

The explicit finite-difference method successfully models the temperature evolution in a rod governed by the 1D heat equation. The solution respects the imposed boundary conditions and demonstrates the expected decay of the initial sinusoidal temperature profile as heat diffuses through the rod.

Problem 3: Numerical Solution of the 2D Poisson Equation on a Square Domain

Problem Statement

The purpose of this problem is to **numerically solve** the two-dimensional Poisson equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -2\pi^2 \sin(\pi x) \sin(\pi y)$$

on the square domain:

$$0 \leq x \leq 1, \qquad 0 \leq y \leq 1$$

with boundary condition:

$$u(x, y) = 0 \quad \text{for all boundaries.}$$

The exact solution to this PDE is known:

$$u(x, y) = \sin(\pi x) \sin(\pi y).$$

This provides a good reference to verify the numerical method.

Methodology

Finite Difference Discretization

The domain is discretized using a uniform grid of size:

$$N = 50, \qquad h = \frac{1}{N+1}.$$

Using the second-order central difference approximation, the Poisson equation becomes:

$$u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j} = -h^2 F_{i,j}$$

where

$$F(x, y) = -2\pi^2 \sin(\pi x) \sin(\pi y).$$

Rearranging gives the classic **Jacobi iteration** formula:

$$u_{i,j}^{(k+1)} = \frac{1}{4} \left(u_{i+1,j}^{(k)} + u_{i-1,j}^{(k)} + u_{i,j+1}^{(k)} + u_{i,j-1}^{(k)} - h^2 F_{i,j} \right).$$

Implementation Steps

1. Construct a $(N+2) \times (N+2)$ grid including boundaries.
2. Initialize $u = 0$ everywhere (consistent with boundary conditions).
3. Compute the forcing term $F(x, y)$.
4. Iterate using Jacobi updates until:
$$\|U^{(k+1)} - U^{(k)}\|_\infty < 10^{-6}$$
5. Visualize the final solution using a surface plot.

Convergence is checked through the maximum absolute difference between successive iterates.

Results

The Jacobi method converged within the specified tolerance. The resulting numerical solution reproduces the expected shape of the analytic solution $u(x, y) = \sin(\pi x)\sin(\pi y)$.

- The solution exhibits a smooth peak at the center $(0.5, 0.5)$.
- Boundary values correctly remain at zero.
- The surface plot clearly shows the symmetric sinusoidal structure.

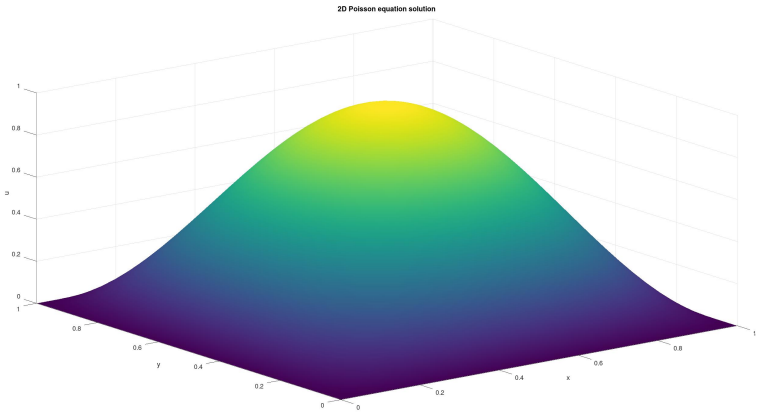

Figure 3.1: Numerical solution of the 2D Poisson equation using the Jacobi iteration method.

Conclusion

The Jacobi finite-difference method successfully solves the 2D Poisson equation on a square domain. The numerical solution matches the theoretical structure of the exact solution and demonstrates correct behavior under zero boundary conditions. Although Jacobi iteration converges slowly, it provides a simple and reliable method for validating PDE solvers.

Problem 4: Time-Dependent Schrödinger Equation for a Harmonic Oscillator with Oscillating Potential

Problem Statement

The goal of this problem is to numerically solve the time-dependent Schrödinger equation (TDSE)

$$i\frac{\partial \psi}{\partial t} = -\frac{1}{2}\frac{\partial^2 \psi}{\partial x^2} + V(x,t)\,\psi(x,t),$$

for a particle placed in a time-dependent "breathing" harmonic potential

$$V(x,t) = \frac{1}{2}\,k(t)\,x^2, \qquad k(t) = \cos^2\!\left(\frac{2\pi t}{T}\right).$$

The initial wavefunction is chosen as the ground-state of a harmonic oscillator:

$$\psi(x,0) = \left(\frac{1}{\pi}\right)^{1/4} e^{-x^2/2}.$$

The objective is to evolve this wavefunction forward in time and study how the probability density $|\psi(x,t)|^2$ behaves as the trap strength oscillates.

Methodology

Discretization

The spatial domain is taken as

$$x \in [-5, 5], \qquad N_x = 200,$$

and the spacing is

$$\Delta x = \frac{x_{\max} - x_{\min}}{N_x - 1}.$$

The simulation runs for

$$\Delta t = 0.0005, \qquad N_t = 2000$$

time steps.

Numerical Update

The second derivative is approximated using the usual central finite-difference expression,

$$\frac{\partial^2 \psi}{\partial x^2} \approx \frac{\psi_{i+1} - 2\psi_i + \psi_{i-1}}{\Delta x^2}.$$

At each time step, the potential is updated according to the breathing function

$$k(t) = \cos^2\left(\frac{2\pi t}{T}\right),$$

and the wavefunction is advanced using the simple explicit update formula implemented in the code. Boundary conditions

$$\psi(-5, t) = \psi(5, t) = 0$$

are applied at every step.

The quantity $|\psi(x, t)|^2$ is stored for visualization, and a heatmap of its logarithm is plotted.

Results

The expected behaviour was a clear breathing pattern, where the wavefunction periodically expands and contracts as the trap oscillates. However, during the simulation the following was observed:

- The values of $|\psi|^2$ steadily increase with time.
- The gradient in the heatmap becomes stronger as time progresses.
- The breathing pattern is not clearly visible in the output.

A sample heatmap produced by the simulation is shown below.

Current Issue and Instructor Feedback Request

At the moment, the simulation does not show the breathing motion that was expected from the physical setup. Instead, the magnitude seems to grow over time, and I am not sure where the issue lies in the numerical procedure.

Since I am still learning these methods, I am unsure whether the problem comes from the time step, the discretization, or something in the update formula.

I would like to request guidance from the instructor on how to correct this and obtain the intended breathing behaviour.

Conclusion

A numerical simulation of a particle in a breathing harmonic potential was performed, but the expected oscillatory behaviour was not clearly observed. The results show growth in the wavefunction amplitude, and further clarification is needed regarding how to stabilize or correct the procedure.

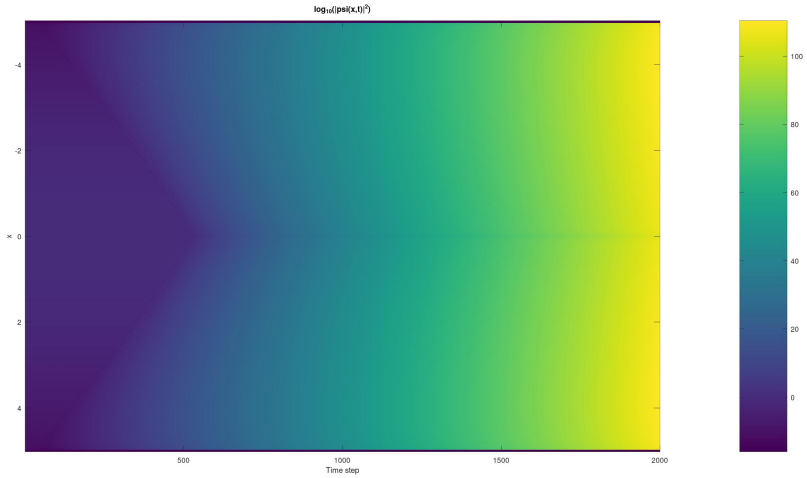

Figure 4.1: Heatmap of $\log_{10}(|\psi(x,t)|^2)$ obtained from the simulation.

Appendix A: Octave Code for Problem 1

```
%% Program to solve for the Duffing Oscillator Equation using RK-4
    Method
%
% Author: Sahil Raj
% Assignment 9 Problem 1

clc; clear all;

% CONSTANTS
k = -1;
b = 0.2;
l = 1;
F0 = 0.3;
w = 1.2;
m = 1.0;

% COEFFICIENTS
c1 = -k/m;
c2 = -l/m;
c3 = b/m;
c4 = F0/m;

% FUNCTION FOR SLOPE OF V(t)
F = @(x, v, t) c1*x + c2*x*x*x + c3*v + c4*cos(w*t);

x0 = 1.0;
v0 = 0.0;

% INITIAL VECTOR
Y0 = [
  x0;
  v0;
];

dt = 0.1;
tmin = 0.0;
tmax = 64.0;

ts = tmin:dt:tmax;
N = length(ts);

% SOLUTION ARRAY
Ys = zeros(N, 2);

% PLACE THE INITIAL CONDITION
Ys(1, :) = Y0;

```

```
for i = 2:N
  Y = Ys(i-1, :);
  x = Y(1);
  v = Y(2);
  t = tmin + (i-1) * dt;

  % slopes for x and v
  k1x = dt * v;
  k1v = dt * F(x, v, t);
  k2x = dt * (v + 0.5*k1v);
  k2v = dt * F(x + 0.5*k1x, v + 0.5*k1v, t + 0.5*dt);
  k3x = dt * (v + 0.5*k2v);
  k3v = dt * F(x + 0.5*k2x, v + 0.5*k2v, t + 0.5*dt);
  k4x = dt * (v + k3v);
  k4v = dt * F(x + k3x, v + k3v, t + dt);
  x_next = x + (k1x + 2*k2x + 2*k3x + k4x)/6;
  v_next = v + (k1v + 2*k2v + 2*k3v + k4v)/6;
  Ys(i,:) = [x_next, v_next];
endfor

% Extract results
Xs = Ys(:, 1);
Vs = Ys(:, 2);

% Create figure with 3 subplots
figure('Name','Duffing Oscillator Results','NumberTitle','off');

% Plot x(t)
subplot(3,1,1);
plot(ts, Xs, 'r', 'LineWidth', 1.5);
grid on;
xlabel('Time t');
ylabel('Displacement x(t)');
title('Duffing Oscillator: Displacement vs Time');

% Plot v(t)
subplot(3,1,2);
plot(ts, Vs, 'g', 'LineWidth', 1.5);
grid on;
xlabel('Time t');
ylabel('Velocity v(t)');
title('Duffing Oscillator: Velocity vs Time');

% Plot phase space (v vs x)
subplot(3,1,3);
plot(Xs, Vs, 'b', 'LineWidth', 1.5);
grid on;
xlabel('Displacement x(t)');
ylabel('Velocity v(t)');
title('Duffing Oscillator: Phase Space');
axis tight;

% Optional: add arrow markers along the phase trajectory to indicate
    direction
hold on;
quiver(Xs(1:10:end-1), Vs(1:10:end-1), ...
       diff(Xs(1:10:end)), diff(Vs(1:10:end)), 0.5, 'k');
hold off;
```

```
104
105 % Improve spacing between subplots
106 sgtitle('Duffing␣Oscillator␣Dynamics␣(RK4␣Method)');
```

Appendix B: Octave Code for Problem 2

```
%% Program to solve the 1D heat flow equation
%
% Author: Sahil Raj
% Assignment 9 Problem 2

clc; clear all;

alph = 0.01;

dx   = 0.05;
xmin = 0.0;
xmax = 1.0; % LENGTH OF THE ROD

dt   = 0.1;
tmin = 0.0;
tmax = 32.0;

c = alph * dt / power(dx, 2);

Xs = xmin:dx:xmax;
Ts = tmin:dt:tmax;

Nx = length(Xs);
Nt = length(Ts);

U  = zeros(Nx, Nt);

% INITIAL AND BOUNDARY CONDITIONS
U(:, 1) = sin(pi*Xs);
U(1, :) = 0;
U(Nx, :) = 0;


for j = 2:Nt
  for i = 2:Nx-1
    U(i, j) = c*U(i+1, j-1) + c*U(i-1, j-1) + (1-2*c)*U(i, j-1);
  endfor
endfor

% VISUALIZATION
surf(Ts, Xs, U)
view(45,30)
shading interp
xlabel('Time'); ylabel('Space'); zlabel('U');
title('Temperature evolution');
```

Appendix C: Octave Code for Problem 3

```
%% Program to solve the 2D Poisson Equation
%
% Author: Sahil Raj
% Assignment 9 Problem 3

clc; clear all;

N = 50;
h = 1/(N+1);
x = linspace(0,1,N+2);
y = linspace(0,1,N+2);

[X,Y] = meshgrid(x,y);

U = zeros(N+2,N+2);
F = -2*pi^2*sin(pi*X).*sin(pi*Y);

tol = 1e-6;
maxIter = 10000;

for iter = 1:maxIter
    U_old = U;
    for i = 2:N+1
        for j = 2:N+1
            U(i,j) = 0.25*( U_old(i+1,j) + U_old(i-1,j) + U_old(i,j+1)
                + U_old(i,j-1) - h^2 * F(i,j) );
        end
    end

    % compute error
    err = max(max(abs(U-U_old)));
    fprintf('Iteration %d, Error = %.6e\n', iter, err);

    % check convergence
    if max(max(abs(U-U_old))) < tol
        fprintf('Converged at iteration %d\n', iter);
        break;
    end
end

surf(X,Y,U)
shading interp
xlabel('x'); ylabel('y'); zlabel('u')
title('2D Poisson equation solution')
```

Appendix D: Octave Code for Problem 4

```
%% Progrm to solve for particle in a breathing potential
%
% Author: Sahil Raj
% Assignment 9 Problem 4

clc; clear all; close all;

% PARAMTERS
Nx = 200;              % Number of spatial points
xmax = 5; xmin = -5;  % Spatial domain
dx = (xmax - xmin)/(Nx-1);
x = linspace(xmin, xmax, Nx)';

T = 1;                 % Oscillation period of the potential
dt = 0.0005;           % Time step (must be small for stability)
Nt = 2000;             % Number of time steps

%% Initial wavefunction: ground state of harmonic oscillator (k=1)
psi = (1/pi)^(1/4) * exp(-0.5*x.^2);

%% Preallocate for visualization (optional)
psi_record = zeros(Nx, Nt);

%% Time evolution
for n = 1:Nt
    t = n*dt;

    % Time-dependent spring constant
    k_t = cos(2*pi*t/T)^2;

    % Potential
    V = 0.5 * k_t * x.^2;

    % Second derivative (Laplacian)
    d2psi = zeros(size(psi));
    d2psi(2:end-1) = (psi(3:end) - 2*psi(2:end-1) + psi(1:end-2)) /
        dx^2;

    % TDSE update (explicit Euler-like)
    psi_new = psi - 1i*dt*(-0.5*d2psi + V.*psi);

    % Boundary conditions
    psi_new(1) = 0;
    psi_new(end) = 0;

    % Update wavefunction
    psi = psi_new;
```

```
47
48      % Store for visualization
49      psi_record(:, n) = psi;
50  end
51
52  %% Visualization: |psi(x,t)|^2 as a heatmap
53  imagesc(1:Nt, x, log10(abs(psi_record).^2 + 1e-16)); % add small
        number to avoid log(0)
54  xlabel('Time step'); ylabel('x'); title('log_{10}(|psi(x,t)|^2)');
55  colorbar;
```