# ESC103F MATLAB Test Notes
# Code for ODEs, IEM, EM and Least Squares

Aditya Jain

November 25, 2025

## 1 Eulers Method

```
% compute the approximation to a function using
% infinitesmally small steps

function [t,x,y] = EMSolver(A, x0, y0, T, N)
% A = Matrix of coeffecients of the two diffy equations
% Ex. A = [1 0; -1 1]
% x0, y0 = initial values
% T = domain of function(s)
% N = step count

dt = T/N
t = 0 : dt : T % creating the steps

Z = NaN(2, length(t))
Z(:, 1) = [x0, y0]

for n = 2:length(t)
    Z(:, n) = Z(:, n-1) + dt * A * Z(:, n-1)
end

x = Z(1,:)
y = Z(2,:)

end
```

## 2 Improved EM

```
% compute the approximation to a function using
% infinitesmally small steps

function [t,x,y] = IEMSolver(A, x0, y0, T, N)
% A = Matrix of coeffecients of the two diffy equations
% Ex. A = [1 0; -1 1]
% x0, y0 = initial values
% T = domain of function/s
% N = step count

dt = T/N
t = 0 : dt : T % creating the steps

Z = NaN(2, length(t))
Z(:, 1) = [x0, y0]

for n = 2:length(t)
    ZL = A * Z(:, n-1)
    Z_pred = Z(: , n-1) + dt * ZL
    ZR = A * Z_pred
    Z(:, n) = Z(:, n-1) + dt * (ZR + ZL) * 0.5
end

x = Z(1,:)
y = Z(2,:)

end
```

## 3 Least Squares Implementation

```
x = x(:) % forces it to be a column vector
y = y(:) % forces it to be a column vector

X = [ones(size(x)) x]
% for quadratic regression : X = [ones(size(x)) x x.^2]
% for cubic regression : X = [ones(size(x)) x x.^2 x.^3]

X_star = X' * X
y_star = X' * y

linreg = X_star \ y_star % yields constant and slope

% plotting
x_plot = linspace(min(x), max(x), 200);
y_plot = linreg(1) + linreg(2) * x_plot;

plot(x , y , "*")
hold on
plot(x_plot, y_plot, "--")

y_pred = X * linreg % works for any polynomial
e = y - y_pred
terms = 1 / (length(x))
MSE_initial = sum(e.^2)
MSE_final = terms * MSE_initial
```

## Mathematical Derivation: Second-Order to First-Order ODE

**Original equations:**

$$m_1 x_1'' = -k_1 x_1 - k_2(x_1 - x_2) \tag{1}$$
$$m_2 x_2'' = -k_2(x_2 - x_1) \tag{2}$$

**Introduce velocities:** Let $v_1 = x_1'$ and $v_2 = x_2'$

**First-order system:**

$$x_1' = v_1 \tag{3}$$
$$v_1' = \frac{-(k_1 + k_2)}{m_1} x_1 + \frac{k_2}{m_1} x_2 \tag{4}$$
$$x_2' = v_2 \tag{5}$$
$$v_2' = \frac{k_2}{m_2} x_1 - \frac{k_2}{m_2} x_2 \tag{6}$$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{k_1+k_2}{m_1} & 0 & \frac{k_2}{m_1} & 0 \\ 0 & 0 & 0 & 1 \\ \frac{k_2}{m_2} & 0 & -\frac{k_2}{m_2} & 0 \end{bmatrix}$$

# 4 Solving 2nd order ODEs

```
function [t, Z] = IEMsolver2(A, Z_0, T, N)
% Improved Euler's Method solver for system of first-order ODEs
% Inputs:
%   A   - State matrix (4x4 for this problem)
%   Z_0 - Initial conditions (4x1 vector)
%   T   - Final time
%   N   - Number of time steps
% Outputs:
%   t - Time array
%   Z - Solution matrix (each row is a variable, each column is a time step)

dt = T / N;
t = 0 : dt : T;

% Initialize solution matrix
Z = NaN(length(Z_0), length(t));
Z(:, 1) = Z_0;

% IEM iteration
for n = 2:length(t)
    % Step 1: Euler prediction
    Z_temp = Z(:, n-1) + dt * A * Z(:, n-1);

    % Step 2: IEM correction (average of slopes)
    Z(:, n) = Z(:, n-1) + (dt/2) * (A*Z(:,n-1) + A*Z_temp);
end

end
```

## 5 Main code for 2nd Order ODEs

```
%% Solve using IEM with different step sizes

N_values = [50, 100, 200, 500];

figure
subplot(2,1,1)
hold on
subplot(2,1,2)
hold on

for i = 1:length(N_values)
    N = N_values(i);
    [t, Z] = IEMsolver2(A, Z_0, T, N);

    % Plot x1(t) - displacement of floor 1
    subplot(2,1,1)
    plot(t, Z(1,:), 'DisplayName', sprintf('N = %d', N))

    % Plot x2(t) - displacement of floor 2
    subplot(2,1,2)
    plot(t, Z(3,:), 'DisplayName', sprintf('N = %d', N))
end

subplot(2,1,1)
xlabel('Time (s)')
ylabel('x_1 (mm)')
title('Displacement of Floor 1')
legend
grid on

subplot(2,1,2)
xlabel('Time (s)')
ylabel('x_2 (mm)')
title('Displacement of Floor 2')
legend
grid on
```