Concordia University
Engineering and Computer Science

Numerical Solution of Double Pendulum

ENCS 6021 – Engineering Analysis

Instructors: Dr. Alex De Visscher, Dr. Rolf Wuthrich

Submitted by
Jashwanth Reddy Earla – 40271577

I certify that this submission is my original work and meets the Faculty's Expectations of Originality

Date: 2024-11-22

Runge-Kutta Method

The RK4 method is a popular method for numerically solving differential equations due to its balance of computational cost and accuracy. It approximates the solution over each step size h by computing a weighted average of four intermediate estimates (K_1, K_2, K_3, K_4) of the slope of the function:

$$k_1 = h \cdot f(t_i, y_i),$$
$$k_2 = h \cdot f(t_i + \frac{h}{2}, y_i + \frac{k_1}{2}),$$
$$k_3 = h \cdot f(t_i + \frac{h}{2}, y_i + \frac{k_2}{2}),$$
$$k_4 = h \cdot f(t_i + h, y_i + k_3).$$

The next value of y is computed as:

$$y_{i+1} = y_i + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$$

Error Analysis and Error Estimation

- The Error Analysis and Estimation section aims to assess the accuracy of the RK4 solution across different step sizes. The process is as follows:
  A "true" solution is obtained using a very fine step size h_{true}=0.0001
- For each tested step size h, the RK4 method is applied, and the approximate values of y are compared to the values from the "true" solution.
- The absolute error for each variable is calculated as:
  Error = |y(tf) – y_true(tf)|
- Errors are plotted on a log-log scale to examine how they decrease with smaller step sizes.

The equations of motions derived earlier when simplified looks as follows:

$$\theta1'' = \frac{-g\ (2\, m1 + m2)\sin\theta1 - m2\ g\sin(\theta1 - 2\,\theta2\ - 2\sin(\theta1 - \theta2)\, m2\ (\theta2'^2\ L2\ +\ \theta1'^2\ L1\cos(\theta1 - \theta2))}{L1\ (2\, m1\ +\ m2\ -\ m2\cos(2\,\theta1 - 2\,\theta2))}$$

$$\theta2'' = \frac{2\sin(\theta1\ -\ \theta2)\ (\theta1'^2 L1\ (m1\ +\ m2)\ +\ g(m1\ +\ m2)\cos\theta1\ +\ \theta2'^2\ L2\, m2\cos(\theta1\ -\ \theta2))}{L2\ (2\, m1\ +\ m2\ -\ m2\cos(2\,\theta1 - 2\,\theta2))}$$

Numerical solution

We try to make the equations in such a way that it can be solved using the Runge-Kutta method. For that we can modify the equation by introducing the following variables.

Y1= θ1	**Y2= θ1′**	**Y3= θ2**	**Y4= θ2′**

Y1'=Y2 **Y3'=Y4**

$$\mathbf{Y2'} = \frac{\mathbf{-g\ (2\ m1 + m2)sinY1\ - m2\ g\ sin(Y1 - 2\ Y3\ - 2\ sin(Y1 - Y3)\ m2\ (Y4^2\ L2\ +\ Y2^2\ L1\ cos(Y1 - Y3))}}{\mathbf{L1\ (2\ m1\ +\ m2\ -\ m2\ cos(2\ Y1 - 2\ Y3))}}$$

$$\mathbf{Y4'} = \frac{\mathbf{2\ sin(Y1\ -\ Y3)\ (Y2^2 L1\ (m1\ +\ m2)\ +\ g(m1\ +\ m2)cos\ Y1\ +\ Y4^2\ L2\ m2\ cos(Y1\ -\ Y3))}}{\mathbf{L2\ (2\ m1\ +\ m2\ -\ m2\ cos(2\ Y1 - 2\ Y3))}}$$

Initial conditions

$\boldsymbol{\theta 1}$ = 0.5,0.05; $\boldsymbol{\theta 2}$ = 0.5,0.05; $\theta 1' = 0; \theta 2' = 0;$ And varying l2 from 0.1 to 0.7 and comparing the motion. m1 = 0.1; m2 = 0.1; L1 = 0.1; L2 = 0.1; g = 9.81

Explanation of the Code and Figures

1. **Define the ODEs**: The function myderiv(t, y,l2) defines the system's equations of motion based on the provided expressions.

2. **RK4 Step and Solution:** rk4_step computes one RK4 step by calculating intermediate slopes K_1, K_2, K_3, K_4 combining them to estimate the next state. rk4_solution applies rk4_step iteratively over the interval to compute a full solution from t0 to tf.

3. **Error Analysis:** The code calculates solutions for progressively smaller step sizes, comparing each to a high-accuracy solution at h_{true}=0.0001. Absolute errors are computed for each component and plotted on a log-log scale to confirm fourth-order accuracy, with errors decreasing proportionally to h^4**.**

4. **Parameter Sensitivity:** Solutions for different values of l2 (0.1, 0.3, 0.5, 0.7) are calculated to observe how this parameter affects system dynamics. Separate plots show the time evolution of each component for various l2 values, illustrating changes in oscillation behaviors.

5. **Generate Plots**:

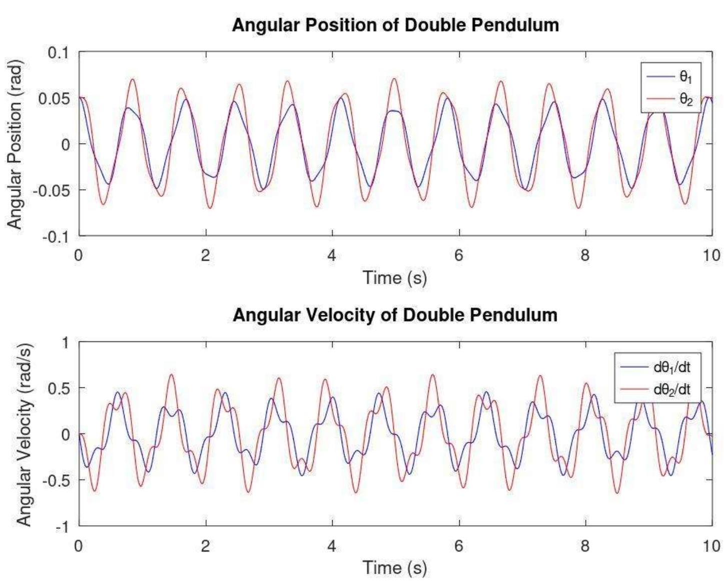

Figure 1Angular position and angular velocity

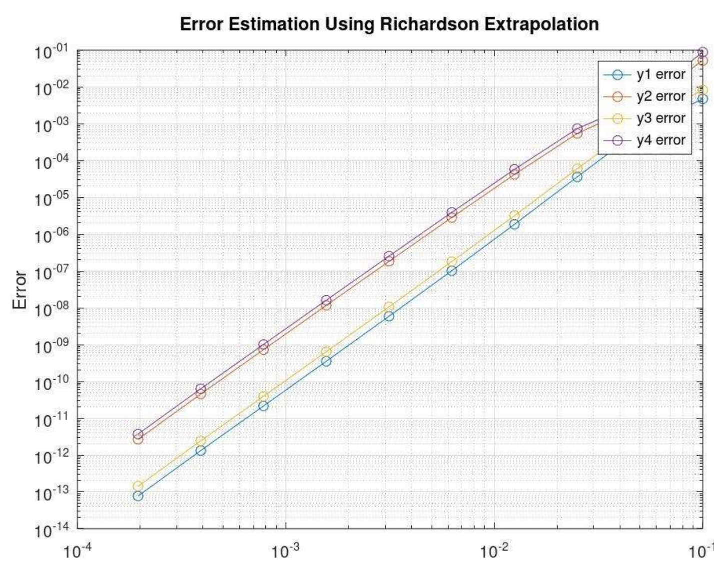

Figure 2 Error Estimates

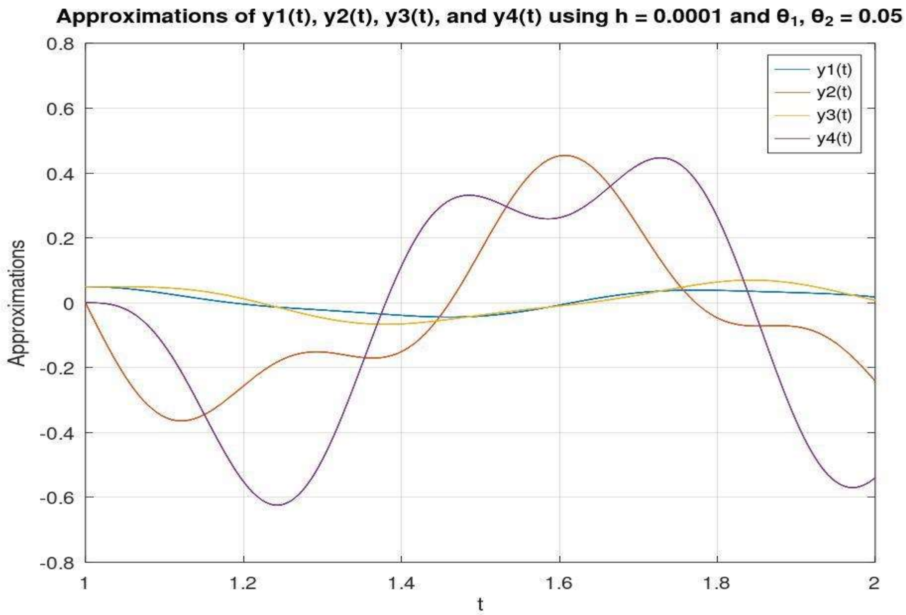

Figure 3 Approximations graph for y1, y2, y3, y4 for step size 0.0001

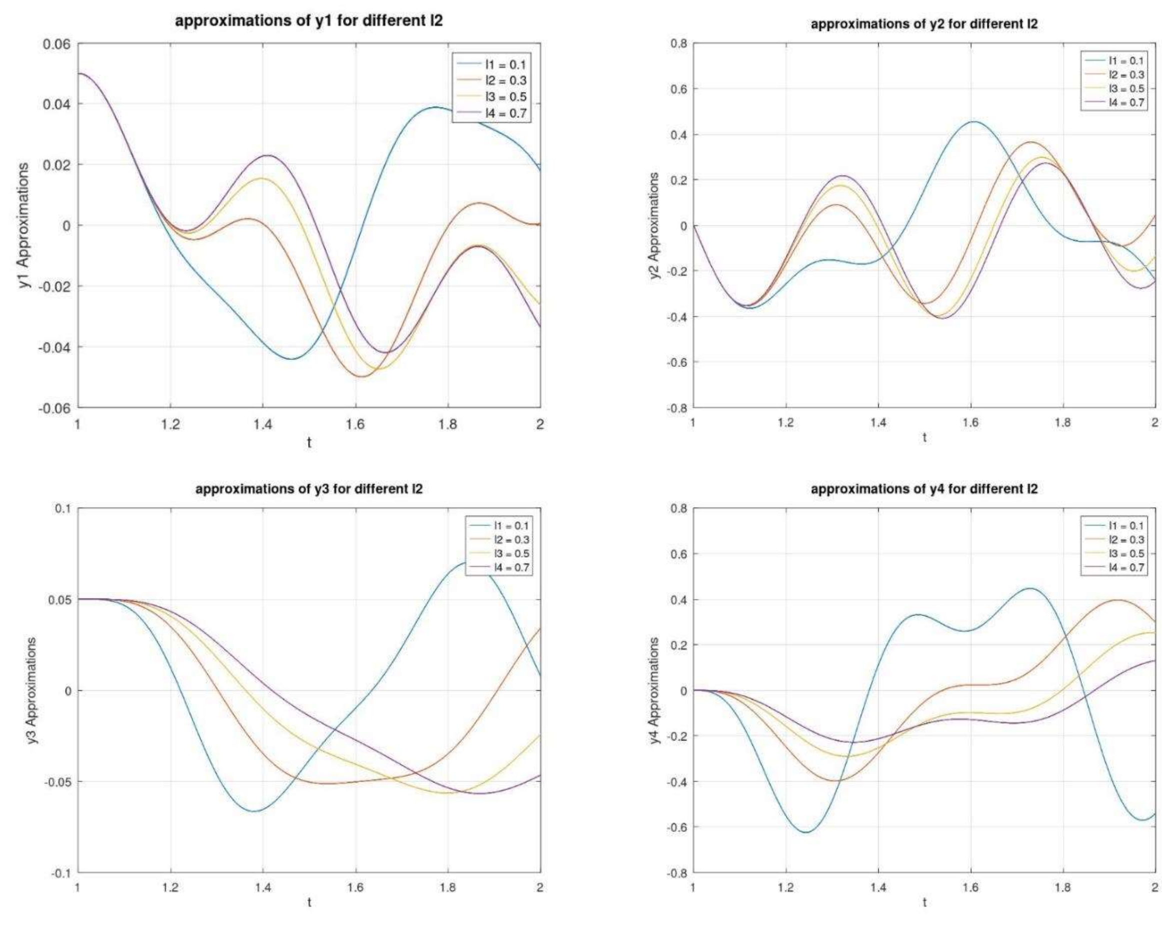

Figure 4 variation in y1, y2, y3, y4 for changing l2

Appendix

```
%initial conditions setup
t0 = 1;
tf = 2;
y0 = [0.05; 0; 0.05; 0];
l2 = 0.1

%single step size solution
h = 0.1;
[t, y] = rk4_solution(t0, tf, y0, h, l2);
fprintf('Approximation at h=0.1 and t=2: y1(2) = %.6f, y2(2) = %.6f\n, y3(2) = %.6f, y4(2) = %.6f\n', y(1, end), y(2, end), y(3, end), y(4, end));

%eroor analysis
steps = [0.1, 0.05, 0.025, 0.0125, 0.00625, 0.003125, 0.0015625, 0.00078125, 0.000390625, 0.0001953125];
errors_y1 = zeros(1, length(steps));
errors_y2 = zeros(1, length(steps));
errors_y3 = zeros(1, length(steps));
errors_y4 = zeros(1, length(steps));

%Approximate true solution(h = 0.001)
h_true = 0.0001;
[t_true, y_true] = rk4_solution(t0, tf, y0, h_true, l2);
y1_true = y_true(1, end);
y2_true = y_true(2, end);
y3_true = y_true(3, end);
y4_true = y_true(4, end);

for j = 1:length(steps)
    h = steps(j);
    [~, y] = rk4_solution(t0, tf, y0, h, l2);
    errors_y1(j) = abs(y(1, end) - y1_true);
    errors_y2(j) = abs(y(2, end) - y2_true);
    errors_y3(j) = abs(y(3, end) - y3_true);
    errors_y4(j) = abs(y(4, end) - y4_true);
end

% Plot errors on a log-log scale
figure;
loglog(steps, errors_y1, '-o', 'DisplayName', 'y1 error');
hold on;
loglog(steps, errors_y2, '-o', 'DisplayName', 'y2 error');
loglog(steps, errors_y3, '-o', 'DisplayName', 'y3 error');
loglog(steps, errors_y4, '-o', 'DisplayName', 'y4 error');
xlabel('Step size (h)');
ylabel('Error');
title('Error Estimation');
legend;
grid on;

%Solution for h = 0.0001
h = 0.0001;
[t, y] = rk4_solution(t0, tf, y0, h, l2);
fprintf('True solution at h = 0.0001 and t=2: y1(2) = %.6f, y2(2) = %.6f\n, y3(2) = %.6f, y4(2) = %.6f\n', y(1, end), y(2, end), y(3, end), y(4, end));


% Plot y1(t) and y2(t)
figure;
plot(t, y(1, :), 'DisplayName', 'y1(t)');
hold on;
plot(t, y(2, :), 'DisplayName', 'y2(t)');
plot(t, y(3, :), 'DisplayName', 'y3(t)');
plot(t, y(4, :), 'DisplayName', 'y4(t)');
xlabel('t');
ylabel('Approximations');
title('Approximations of y1(t), y2(t), y3(t), and y4(t) using h = 0.0001 and \theta_1, \theta_2 = 0.05');
legend;
```

```
grid on;
```

Appendix

```
%initial conditions setup
t0 = 1;
tf = 2;
y0 = [0.05; 0; 0.05; 0];

% Single step size for solution
h = 0.0001;
n = ceil((tf - t0) / h);
l2_values = [0.1; 0.3; 0.5; 0.7];

% Initialize arrays to store final values of y1, y2, y3, and y4 at t = 2 for each l2
y1_final = zeros(length(l2_values), n + 1);
y2_final = zeros(length(l2_values), n + 1);
y3_final = zeros(length(l2_values), n + 1);
y4_final = zeros(length(l2_values), n + 1);

% Loop over each l2 value

[t, y_l1] = rk4_solution(t0, tf, y0, h, l2_values(1));  % Solve the system for this l2
[t, y_l2] = rk4_solution(t0, tf, y0, h, l2_values(2));
[t, y_l3] = rk4_solution(t0, tf, y0, h, l2_values(3));
[t, y_l4] = rk4_solution(t0, tf, y0, h, l2_values(4));

%plot y1(t) for different l2
figure;
plot(t, y_l1(1,:), 'DisplayName', 'l1 = 0.1');
hold on;
plot(t, y_l2(1,:), 'DisplayName', 'l2 = 0.3');
plot(t, y_l3(1,:), 'DisplayName', 'l3 = 0.5');
plot(t, y_l4(1,:), 'DisplayName', 'l4 = 0.7');
xlabel('t');
ylabel('y1 Approximations');
title('approximations of y1 for different l2');
legend;
grid on;

%plot y2(t) for different l2
figure;
plot(t, y_l1(2,:), 'DisplayName', 'l1 = 0.1');
hold on;
plot(t, y_l2(2,:), 'DisplayName', 'l2 = 0.3');
plot(t, y_l3(2,:), 'DisplayName', 'l3 = 0.5');
plot(t, y_l4(2,:), 'DisplayName', 'l4 = 0.7');
xlabel('t');
ylabel('y2 Approximations');
title('approximations of y2 for different l2');
legend;
grid on;

%plot y3(t) for different l2
figure;
plot(t, y_l1(3,:), 'DisplayName', 'l1 = 0.1');
hold on;
plot(t, y_l2(3,:), 'DisplayName', 'l2 = 0.3');
plot(t, y_l3(3,:), 'DisplayName', 'l3 = 0.5');
plot(t, y_l4(3,:), 'DisplayName', 'l4 = 0.7');
xlabel('t');
ylabel('y3 Approximations');
title('approximations of y3 for different l2');
legend;
grid on;

%plot y4(t) for different l2
figure;
plot(t, y_l1(4,:), 'DisplayName', 'l1 = 0.1');
hold on;
plot(t, y_l2(4,:), 'DisplayName', 'l2 = 0.3');
plot(t, y_l3(4,:), 'DisplayName', 'l3 = 0.5');
plot(t, y_l4(4,:), 'DisplayName', 'l4 = 0.7');
```

```
xlabel('t');
ylabel('y4 Approximations');
title('approximations of y4 for different l2');
legend;
grid on;
```

Appendix

```
% Function to compute RK4 solution for a given step size
function [t, y] = rk4_solution(t0, tf, y0, h, l2)
    n = ceil((tf - t0) / h);
    t = linspace(t0, tf, n + 1);
    y = zeros(4, n + 1);
    y(:, 1) = y0;
    for i = 1:n
        y(:, i + 1) = rk4_step(t(i), y(:, i), h, l2);
    end
end
```

Appendix

```
% Function to compute RK4 step
function y_next = rk4_step(t_i, y_i, h, l2)
      f = @(t,x) myderiv(t,x,l2);
    k1 = h * f(t_i, y_i);
    k2 = h * f(t_i + h/2, y_i + k1/2);
    k3 = h * f(t_i + h/2, y_i + k2/2);
    k4 = h * f(t_i + h, y_i + k3);
    y_next = y_i + (k1 + 2*k2 + 2*k3 + k4) / 6;
end
```

```
function z = myderiv(t,y,l2)
  %constants
    m1 = 0.1;
    m2 = 0.1;
    L1 = 0.1;
    g = 9.81;

    L2 = l2;
    z(1) = y(2);
    z(2) = (-g * (2 * m1 + m2) * sin(y(1)) - m2 * g * sin(y(1) - 2 * y(3)) ...
             - 2 * sin(y(1) - y(3)) * m2 * (y(4)^2 * L2 + y(2)^2 * L1 * cos(y(1) - y(3)))) ...
             / (L1 * (2 * m1 + m2 - m2 * cos(2 * y(1) - 2 * y(3))));
    z(3) = y(4);
    z(4) = (2 * sin(y(1) - y(3)) * (y(2)^2 * L1 * (m1 + m2) + g * (m1 + m2) * cos(y(1)) ...
             + y(4)^2 * L2 * m2 * cos(y(1) - y(3)))) / (L2 * (2 * m1 + m2 - m2 * cos(2 * y(1) - 2 * y(3))));
    z = [z(1); z(2); z(3); z(4)];
end
```