Warsaw University of Technology

Numerical Methods

ENUME

Project C No.10

Report

Submitted by:

Gaurav Chauhan

309602

Problem I-Least-Square Approximation

Problem Description

For the following experimental measurements (samples):

xi	**yi**
-5	–4.6643
-4	–5.5445
-3	–4.0378
-2	–2.0868
-1	–1.0393
0	0.6916
1	–0.0237
2	0.4107
3	–3.6761
4	–9.8466
5	–18.8868

Determine a polynomial function y=f(x) that best fits the experimental data by using the least-squares approximation (test polynomials of various degrees). Present the graphs of obtained functions along with the experimental data. To solve the least-squares problem use the system of normal equations with QR factorization of a matrix A. For each solution calculate the error defined as the Euclidean norm of the vector of residuum and the condition number of the Gram's matrix. Compare the results in terms of solutions' errors.

Method description

Least-square approximation fit a model to data such that the squared residuals is minimum. Equation 1 is a mathematical description of a model function for least-squares approximation.

$$\forall F \in X_n, \ \ F(x) = \sum_{i=0}^{n} a_i \emptyset_i(x) \tag{1}$$

A residual is the difference between the model function and data samples. Sum of squared residuals is then calculated according to Equation 2.

$$S = \sum_{i=0}^{n} r_i^2 = \sum_{i=0}^{n} [f(x_i) - a_i \emptyset_i(x)]^2 \tag{2}$$

By using calculus of derivatives, we find the minimum of sum of squared residuals at the point where the derivative of sum of squared residuals equals zero as described by Equation 3.

$$\frac{dS}{da_j} = -2 \sum_{i=0}^{n} r_i \frac{dr_i}{da_j} = 0, j = 0, \dots, n$$

$$\frac{dS}{da_k} = -2 \sum_{j=0}^{N} \left[f(x_j) - \sum_{i=0}^{n} a_i \emptyset_i(x_j) \right] . \emptyset_k(x_j) = 0, k = 0, \dots, n \tag{3}$$

Equation 3 is used to develop as a set of normal equations as described in Equation 4.

$$a_0 \sum_{j=0}^{N} \emptyset_0(x_j) . \emptyset_n(x_j) + a_1 \sum_{j=0}^{N} \emptyset_1(x_j) . \emptyset_n(x_j) + \cdots + a_n \sum_{j=0}^{N} \emptyset_n(x_j) . \emptyset_n(x_j) = \sum_{j=0}^{N} f(x_j) . \emptyset_n(x_j) \tag{4}$$

From equation 4, a solution of the set of equations is described by Equation 5

$$P(a) = \left(||y - \boldsymbol{A}a||_2 \right)^2 \tag{5}$$

Where

$$\boldsymbol{A} = \begin{bmatrix} \emptyset_0(x_0) & \emptyset_1(x_0) & \dots & \emptyset_n(x_0) \\ \emptyset_0(x_1) & \emptyset_1(x_0) & \dots & \emptyset_n(x_1) \\ \emptyset_0(2) & \emptyset_1(x_0) & \dots & \emptyset_n(x_2) \\ \dots & \dots & \dots & \dots \\ \emptyset_0(x_N) & \emptyset_1(x_N) & \dots & \emptyset_n(x_N) \end{bmatrix}, \qquad a = \begin{bmatrix} a_0 \\ a_1 \\ \dots \\ a_n \end{bmatrix}, \qquad y = \begin{bmatrix} y_0 \\ y_1 \\ \dots \\ y_n \end{bmatrix}$$

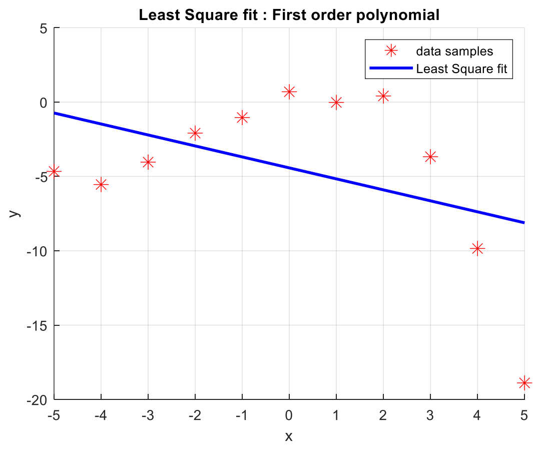
QR factorization on matrix A is used to solve for the unknown parameters according to Equation 6.

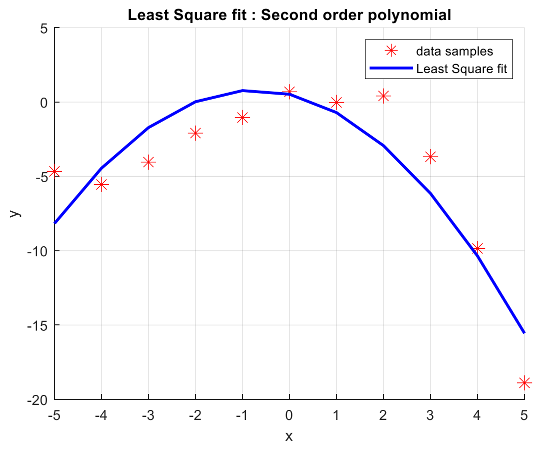
$$a = R^{-1}(Q^T y) \tag{6}$$

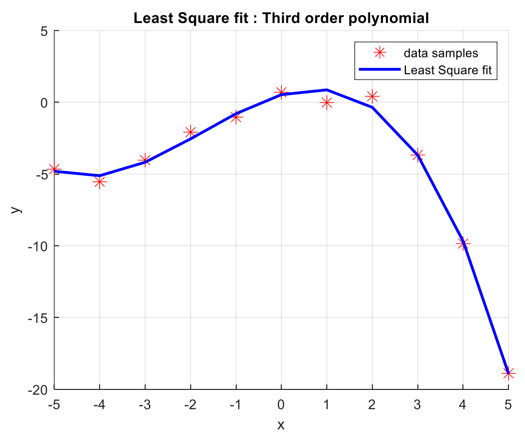
The data point provided contains 11 samples. This allows to fit a 10th degree polynomial or less for the unknown parameters to be deterministic. A Matlab program is used to implement the solution method described. The program code is attached in the appendix section.

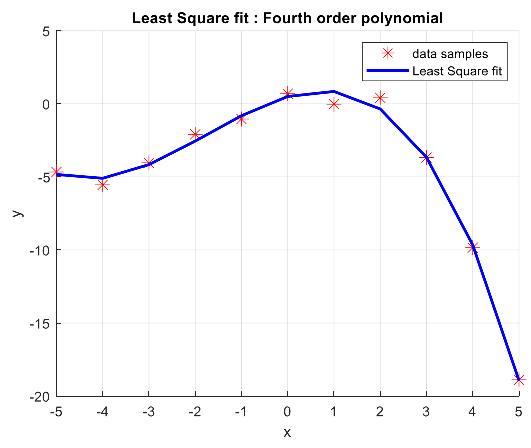
Results

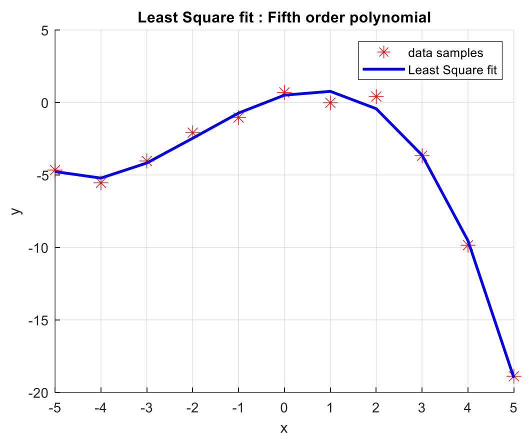
There are 10 polynomials starting form first order to 10th order which were fitted to the data as shown in the following figures.

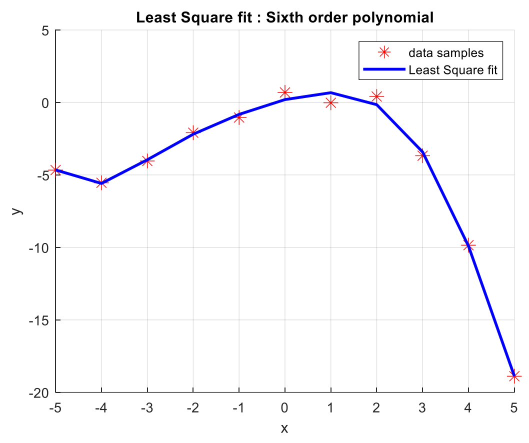


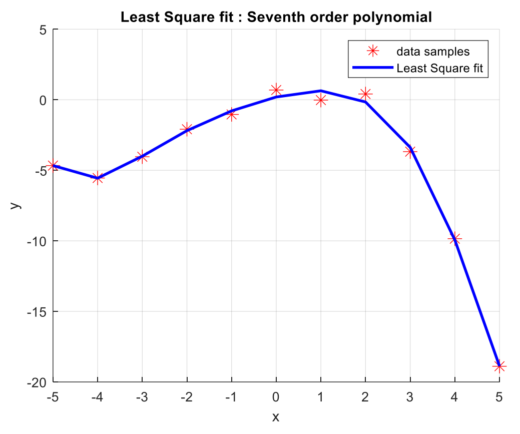


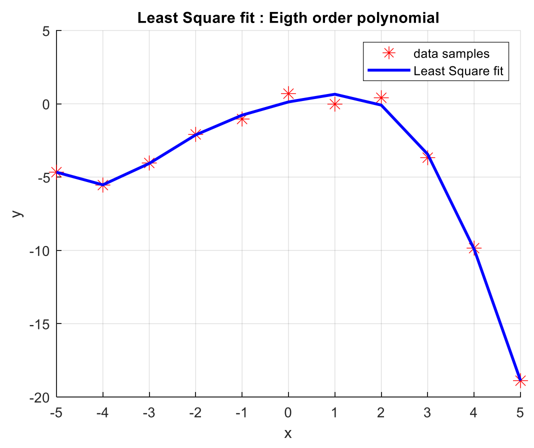

Least Square fit : Third order polynomial
data samples
Least Square fit
x
y

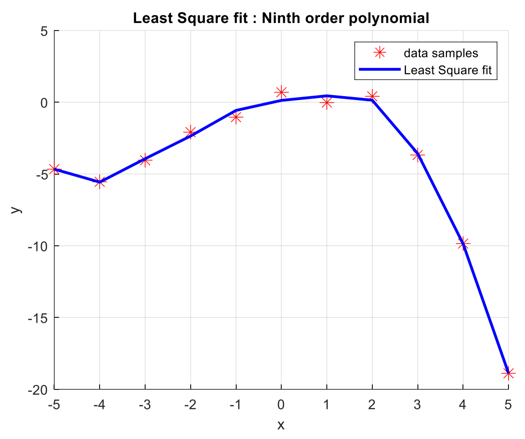

Least Square fit : Fourth order polynomial
data samples
Least Square fit
x
y

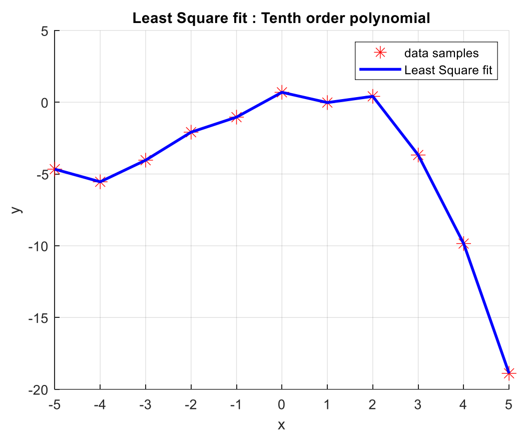

Least Square fit : Fifth order polynomial
data samples
Least Square fit
x
y


Least Square fit : Sixth order polynomial
data samples
Least Square fit
x
y


Least Square fit : Seventh order polynomial
data samples
Least Square fit
x
y


Least Square fit : Eigth order polynomial
data samples
Least Square fit
x
y



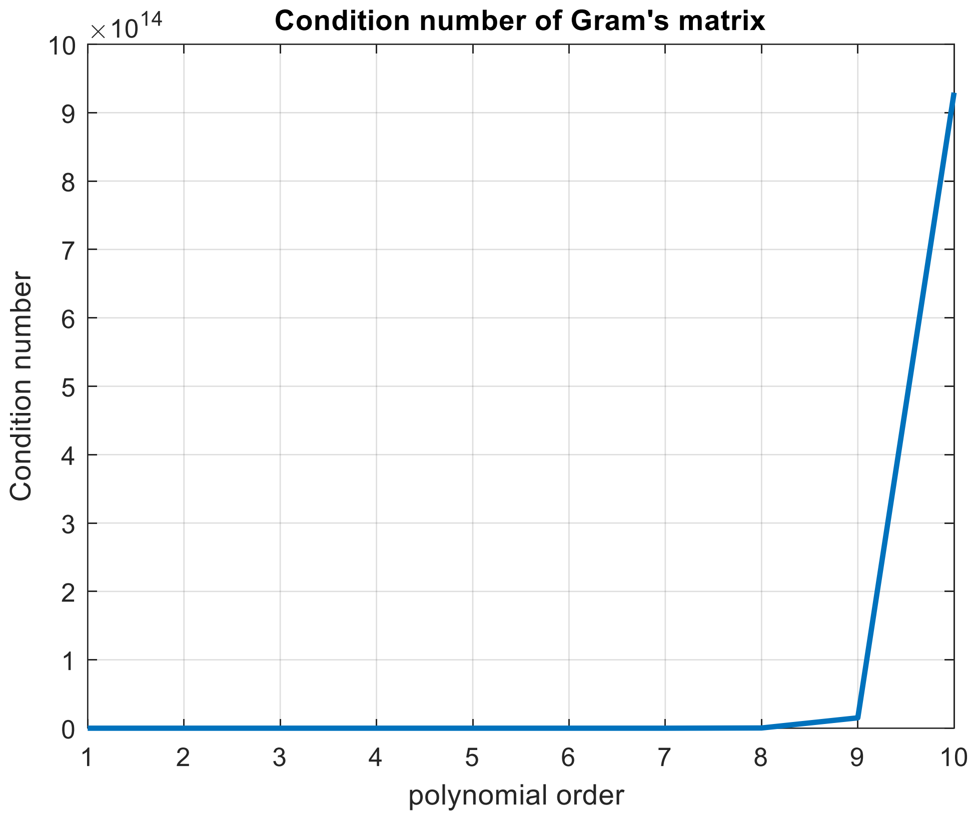


As the polynomial order increases, the accuracy of the approximation increases. The polynomial degree 10 provides the least Euclidean Norm of residuum of A matrix provided in table 1.

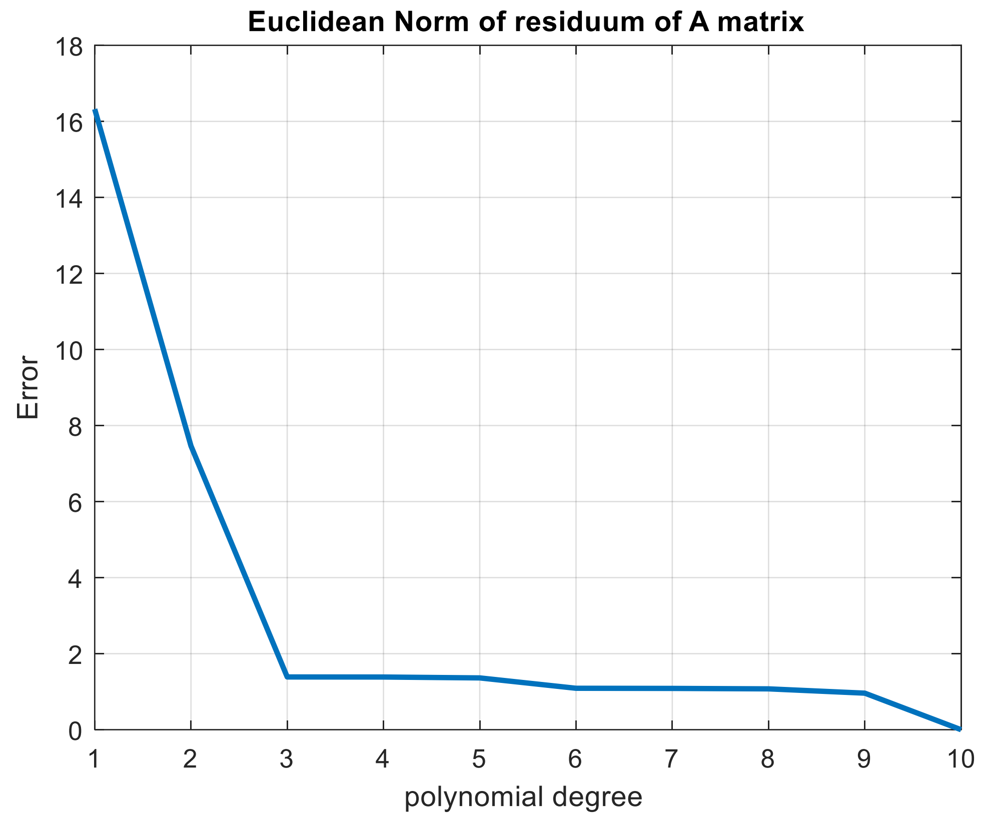
Table 1: Summary of the Euclidean Norm of residuum of A matrix and Gram's Matrix

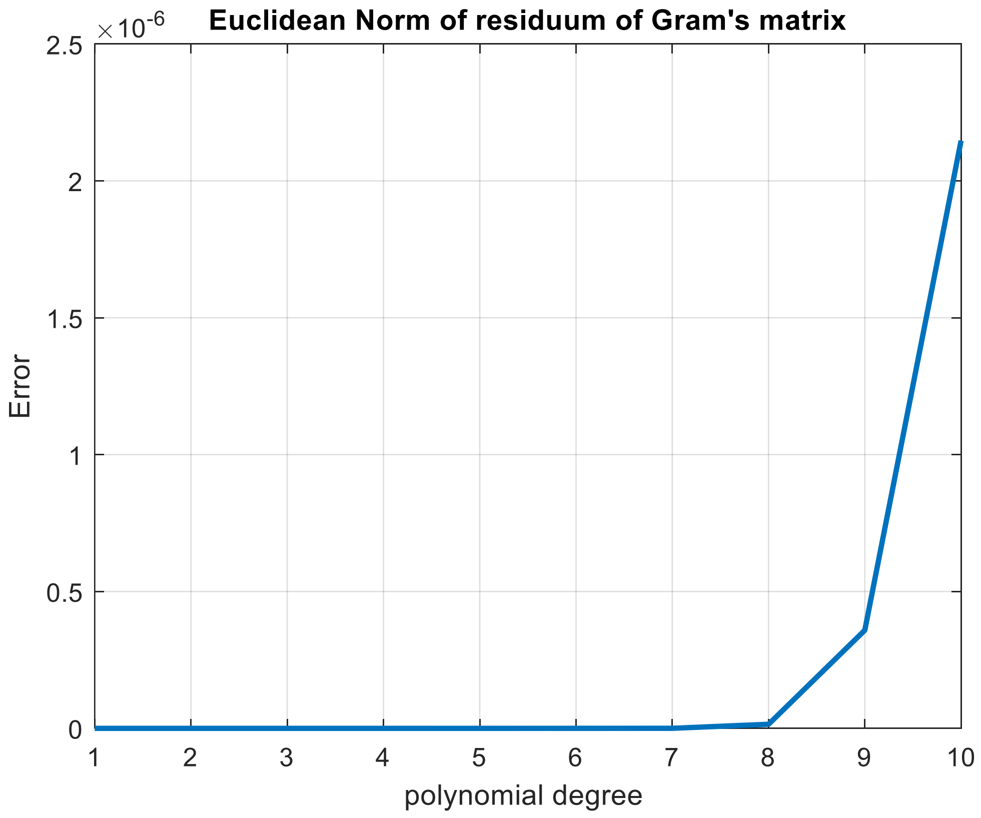
Polynomial Degree	**Euclidean Norm of residuum of A matrix**	**Condition number of Gram's matrix**	**Euclidean Norm of residuum of Gram's Matrix**
1	16.323	10	5.6843E-14
2	7.4658	408.78	4.5834E-13
3	1.387	8558.4	2.2793E-13
4	1.3855	3.1798e+05	6.9288E-13
5	1.3621	7.4675e+06	1.4554E-11
6	1.0898	2.8316e+08	1.1738E-10
7	1.086	7.6462e+09	2.3340E-10
8	1.0741	3.3055e+11	1.4980E-08
9	0.96118	1.5167e+13	3.5845E-07
10	5.6185e-13	9.293e+14	2.1470E-06

There is a rapid increase of condition number of Gram's matrix with increases degree of the polynomial. A condition number indicates sensitivity to round off errors in the calculations using Gram’s matrix. This is the reason why the solution of the system of equations are solved through QR factorization. The changes of condition number of Gram’s matrix with varying polynomial degree is shown in the following plot.



The Euclidean Norm of residuum **A** matrix and graph matrix are shown in the following two plots. The error reduces with increasing polynomials for A matrix while it increases for a Gram’s matrix. The Gram’s matrix results to increasing errors as the size of matrix increases. The A matrix solved is good conditioned and have reducing errors as the order of polynomials increases.


Euclidean Norm of residuum of A matrix
Error
polynomial degree
0
2
4
6
8
10
12
14
16
18
1
2
3
4
5
6
7
8
9
10


Euclidean Norm of residuum of Gram's matrix
$\times 10^{-6}$
Error
polynomial degree
0
0.5
1
1.5
2
2.5
1
2
3
4
5
6
7
8
9
10

Problem II-Solving A system of ODEs using Runge-Kutta method and Adams PC

Problem description

A motion of a point is given by equations:

$$\begin{aligned}\frac{dx_1}{dt} &= x_2 + x_1(0.5 - x_1^2 - x_2^2)\\ \frac{dx_1}{dt} &= -x_1 + x_2(0.5 - x_1^2 - x_2^2)\end{aligned} \tag{7}$$

Determine the trajectory of the motion on the interval [0,20] for the following initial conditions: x1(0) = 0.002, x2(0) = 0.02. Evaluate the solution using:

(a) Runge-Kutta method of 4th order (RK4) and Adams PC (P5CE5E)-each method a few times, with different constant step-size until an optimal constant step size is found, i.e when its decrease does not influence the solution significantly but its increase does

(b) Runge-Kutta method of 4th order (RK4) with a variable step size automatically adjusted by the algorithm, making error estimation according to the step-doubling rule.

Compare the results with the ones obtained using an ode SOLVER e.g ode45.

Method description

Runge-Kutta 4th order method

A system of ODE can be solved numerically using various methods. One of the most common method is Runge-Kutta method 4th order method. Runge-Kutta method can be modelled for different orders such as second order, third order, and fourth order and higher. We can describe a general mth stage of a Runge-Kutta method by the following Equation.

$$y(t_{n+1}) = y_{n+1} = y_n + h\sum_{i=1}^{m} c_i k_i \tag{8}$$

Where

$$k_1 = f(t_n, y_n),$$

$$k_2 = f(t_n + \alpha_2 h, y_n + h\beta_{21}k_1),$$

$$k_3 = f(t_n + \alpha_3 h, y_n + h\beta_{31}k_1 + h\beta_{32}k_2),$$

$$\vdots$$

$$k_m = f\left(t_n + \alpha_m h, y_n + h\sum_{j=1}^{m-1} \beta_{mj}k_j\right).$$

The parameters $\alpha_m, \beta_{mj} and\ k_j$ are chosen to provide the highest convergence order for the given number of stages. Runge-Kutta 4th order (RK4) method has four stages with convergence of order 4.

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \qquad (9)$$

Where

$$k_1 = f(t_n, y_n),$$

$$k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right),$$

$$k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right),$$

$$k_4 = f(t_n + h, y_n + hk_3).$$

Equation 9 describes a constant step size algorithm for the Runge-Kutta fourth order.

Variable step size Runge-Kutta method (RK4)

The accuracy of Runge-Kutta method (RK4) can be improved by adjusting the step size in each iteration. The adaptive control of step size calculates solution for the current step size h and at half step size. Based on the two solutions at different step sizes, the improved solution based on y1 and y2 as given by the following equations:

$$Y_{n+1} = y_1 + Ch^{p+1} + \mathcal{O}(h^{p+2}) \tag{10a}$$

$$Y_{n+1} = y_2 + 2C\left(\frac{h}{2}\right)^{p+1} + \mathcal{O}(h^{p+2}) \tag{10b}$$

The difference between 10a and 10b gives the following equation.

$$|y_1 - y_2| = CH^{p+1}\left(1 - \frac{1}{2^p}\right)$$

A solution for C is given by the following expression.

$$C = \frac{|y_1 - y_2|}{(1 - 2^{-p})h^{p+1}}$$

Equation 10b can be re-written as follows in Equation 11.

$$Y_{n+1} = y_2 + \delta + \mathcal{O}(h^{p+2}) \tag{11}$$

Where

$$\delta = \frac{|y_1 - y_2|}{2^p - 1}$$

Relative error and absolute error are combined into an error term $\mathcal{E}$ which is estimated using the following Equation.

$$\mathcal{E} = |y_2| * \mathcal{E}_r + \mathcal{E}_a \tag{12}$$

Where

$\mathcal{E}_r = relative\ error,\qquad \mathcal{E}_a = absolute\ error$

The step size is adjusted such that the truncation error is less than the tolerance. This is defined in Equation 13,

$$h_{new} = \beta h \alpha \tag{13}$$

Where

α = step size correction coefficient

β = a safety factor.

The $\mathcal{E}$ is used for step-size control in the calculation of correction coefficient as shown in Equation 14.

$$\alpha = \max\left(\frac{\varepsilon_i}{|\delta_n(h)_i|}\right)^{\frac{1}{p+1}} \tag{14}$$

$$\beta = 0.9 \tag{15}$$

For the next iteration, the used step-size is the minimum of (hnew, tf-tn, 2hn). The term tf-tn is used to ensure the last iteration matches the chosen end-time in the defined interval. The figure that follows shows a flow diagram of the adaptive step size in Runge-Kutta method.

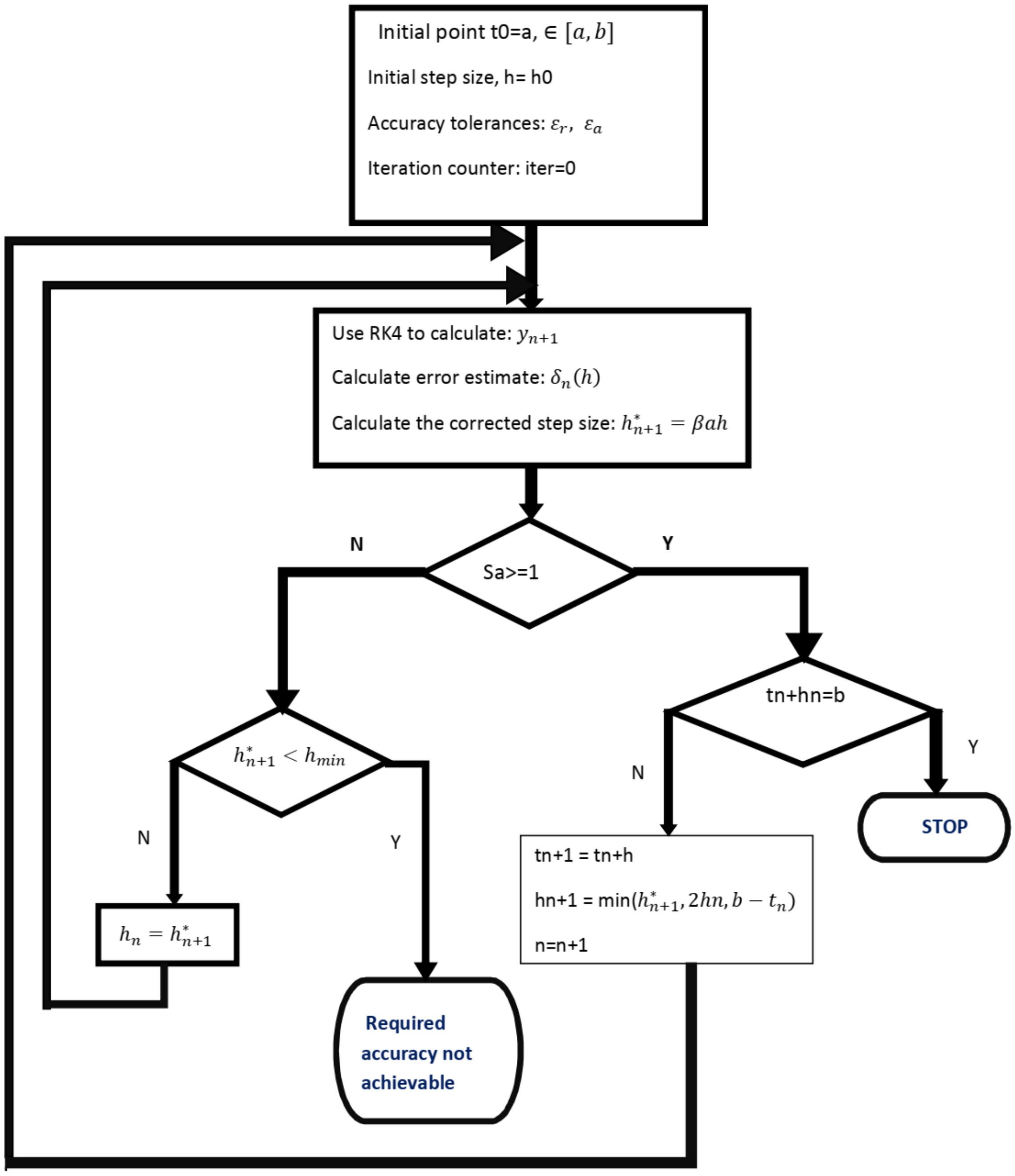

Initial point t0=a, ∈ $[a,b]$
Initial step size, h= h0
Accuracy tolerances: $\varepsilon_r,\ \varepsilon_a$
Iteration counter: iter=0
Use RK4 to calculate: y_{n+1}
Calculate error estimate: $\delta_n(h)$
Calculate the corrected step size: $h^*_{n+1} = \beta a h$
N
Y
Sa>=1
tn+hn=b
$h^*_{n+1} < h_{min}$
N
Y
N
Y
$h_n = h^*_{n+1}$
tn+1 = tn+h
hn+1 = min($h^*_{n+1}, 2hn, b - t_n$)
n=n+1
STOP
Required accuracy not achievable

Fig1: **Flow diagram of the adaptive step size control by use of the step doubling method.**

Adams P5EC5E

Adams method is based on first making a prediction followed by a correction step. The prediction and correction steps can be repeated several times defined at the start of the steps.The predictor equation is an explicit method from Adams–Bashforth method while the corrector equation is an implicit function Adams–Moulton method.

The general Adams–Bashforth method is given Equation 16.

$$y_n = y_{n-1} + h \sum_{j=1}^{k} \beta_j f(x_{n-j}, y_{n-j}) \tag{16}$$

Table 2 provides the parameters of Adam-Bashforth Equations.

Table 2: The parameters of Adam-Bashforth method

k	β_1	β_2	β_3	β_4	β_5	β_6	β_7
1	1						
2	$\frac{3}{2}$	$-\frac{1}{2}$					
3	$\frac{23}{12}$	$-\frac{16}{12}$	$\frac{5}{12}$				
4	$\frac{55}{24}$	$-\frac{59}{24}$	$\frac{37}{24}$	$-\frac{9}{24}$			
5	$\frac{1901}{720}$	$-\frac{2774}{720}$	$\frac{2616}{720}$	$-\frac{1274}{720}$	$\frac{251}{720}$		
6	$\frac{4277}{1440}$	$-\frac{7923}{1440}$	$\frac{9982}{1440}$	$-\frac{7298}{1440}$	$\frac{2877}{1440}$	$-\frac{475}{1440}$	
7	$\frac{198721}{60480}$	$-\frac{447288}{60480}$	$\frac{705549}{60480}$	$-\frac{688256}{60480}$	$\frac{407139}{60480}$	$-\frac{134472}{60480}$	$\frac{19087}{60480}$

The general Adam-Moulton method is given Equation 17.

$$y_n = y_{n-1} + h\sum_{j=1}^{k} \beta_j^* f(x_{n-j}, y_{n-j}) \tag{17}$$

Table 3 provides the parameters of Adam-Moulton Equations.

Table 3: parameters of Adam-Moulton method

k	β_1^*	β_1	β_2	β_3	β_4	β_5	β_6	β_7
1⁺	1							
1	$\frac{1}{2}$	$-\frac{1}{2}$						
2	$\frac{5}{12}$	$\frac{8}{12}$	$-\frac{1}{12}$					
3	$\frac{9}{24}$	$\frac{19}{24}$	$-\frac{5}{24}$	$\frac{1}{24}$				
4	$\frac{251}{720}$	$\frac{646}{720}$	$-\frac{264}{720}$	$\frac{106}{720}$	$-\frac{19}{720}$			
5	$\frac{475}{1440}$	$\frac{1427}{1440}$	$-\frac{798}{1440}$	$\frac{482}{1440}$	$-\frac{173}{1440}$	$\frac{27}{1440}$		
6	$\frac{19087}{60480}$	$\frac{65112}{60480}$	$-\frac{46461}{60480}$	$\frac{37505}{60480}$	$-\frac{20211}{60480}$	$\frac{6312}{60480}$	$-\frac{863}{60480}$	
7	$\frac{36799}{120960}$	$\frac{139849}{120960}$	$-\frac{121797}{120960}$	$\frac{123133}{120960}$	$-\frac{88547}{120960}$	$\frac{41499}{120960}$	$-\frac{11351}{120960}$	$\frac{1375}{120960}$

The 5 step Adams PC method is correctly named as the Adam PC (P_5EC_5E) because of the 5 steps at prediction and 5 steps at correction. Equation 18 and 19 defines the predictor and corrector expressions.

Prediction:

$$y_{n+5} = y_{n+4} + h\left(\frac{1901}{720}f(t_{n+4}, y_{n+4}) - \frac{2774}{720}f(t_{n+3}, y_{n+3}) + \frac{2616}{720}f(t_{n+2}, y_{n+2}) - \frac{1274}{720}f(t_{n+1}, y_{n+1}) + \frac{251}{720}f(t_n, y_n)\right) \tag{18}$$

Correction:

$$y_{n+5} = y_{n+4} + h\left(\frac{475}{1440}f(t_{n+5}, y_{n+5}) + \frac{1427}{1440}f(t_{n+4}, y_{n+4}) - \frac{798}{1440}f(t_{n+3}, y_{n+3}) + \frac{482}{1440}f(t_{n+2}, y_{n+2}) - \frac{173}{1440}f(t_{n+1}, y_{n+1}) + \frac{27}{1440}f(t_n, y_n)\right) \qquad (19)$$

Since the methods requires at least 5 points, it requires other methods to generate the first 5 points. In the Adam PC (P_5EC_5E) we use Runge-Kutta 4th order method to generate the first five points.

Results

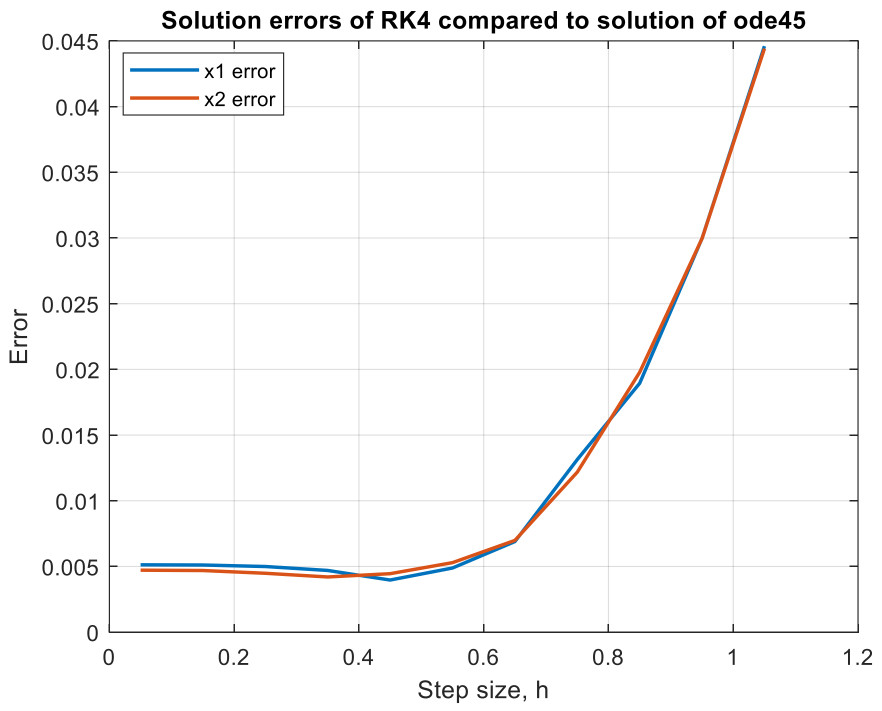
Task Ia-RK4

While using constant step size with Runge-Kutta method, we find that the solution error increases with increasing step size. The results are summarized in Table 4.

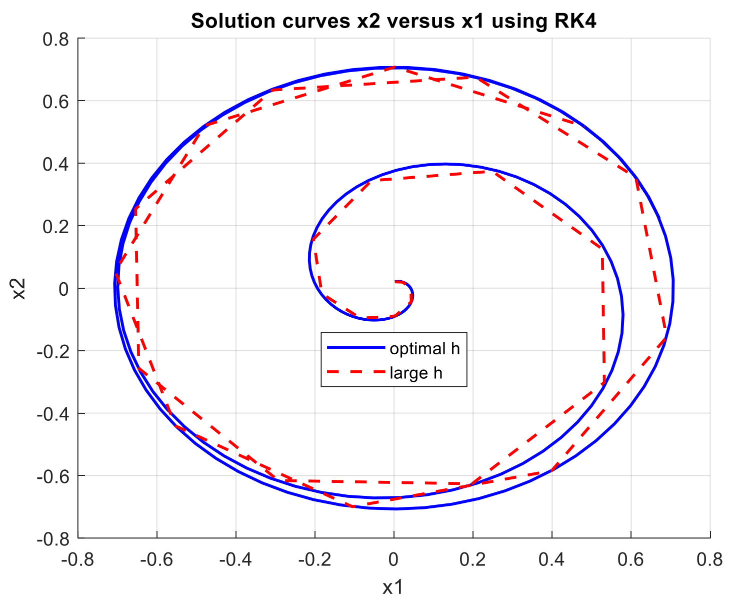
Table 4: Summary of the errors of RK4 with respect to the step size

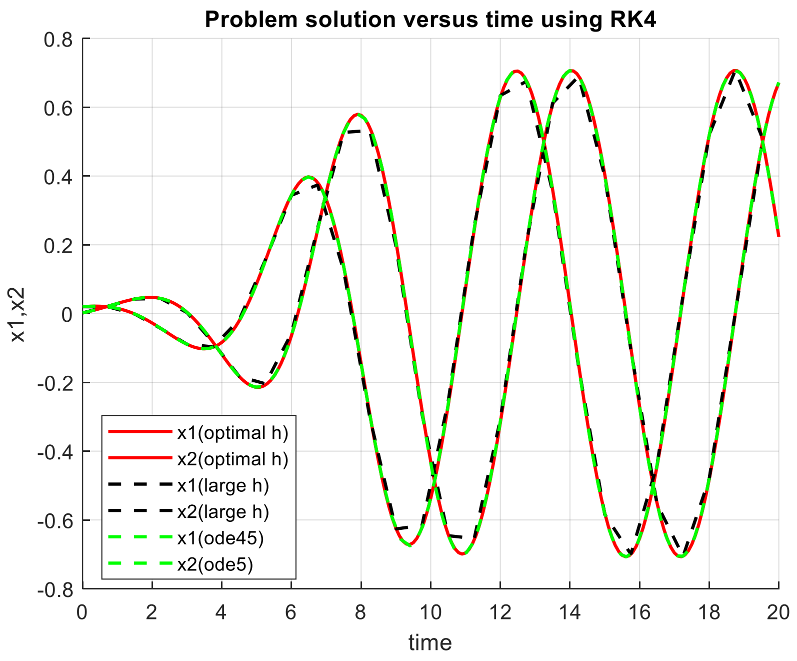
Step size for RK4	Error x1	Error x2
0.05	0.005126	0.004717
0.15	0.005111	0.004691
0.25	0.004997	0.004482
0.35	0.004697	0.004201
0.45	0.003968	0.004451
0.55	0.004884	0.005292
0.65	0.006895	0.006983
0.75	0.013139	0.01218
0.85	0.018943	0.01978
0.95	0.029972	0.029965
1.05	0.044602	0.044414

The same trend found in table 4 is also shown the figure plotting the solution error versus time. An optimal step size of 0.1 is chosen to calculate the final solution using Runge-Kutta method of fourth order.

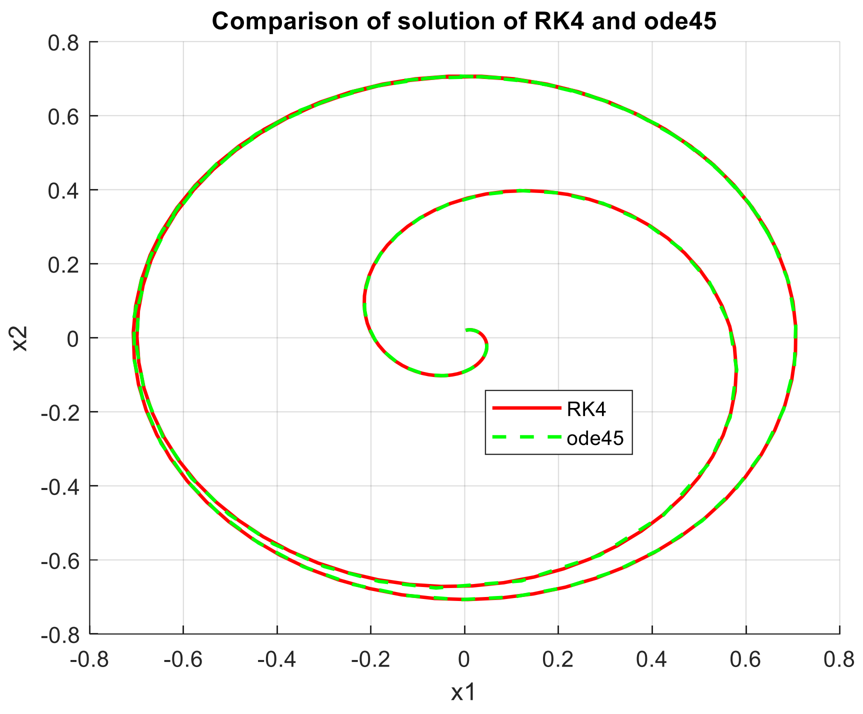


The solution curve using the optimal step size generates a smooth curve while the solution using a large step size generates low accuracy curve. This behavior is also observed when the solution is plotted as function of time.





A comparison of the solution curve obtained using Runge-Kutta method is plotted on the same axis with the solution obtained from ode45 solver. The two curves overlap indicating two have similar solution with comparable accuracies.



Task Ia-Adams PC (P_5EC_5E)

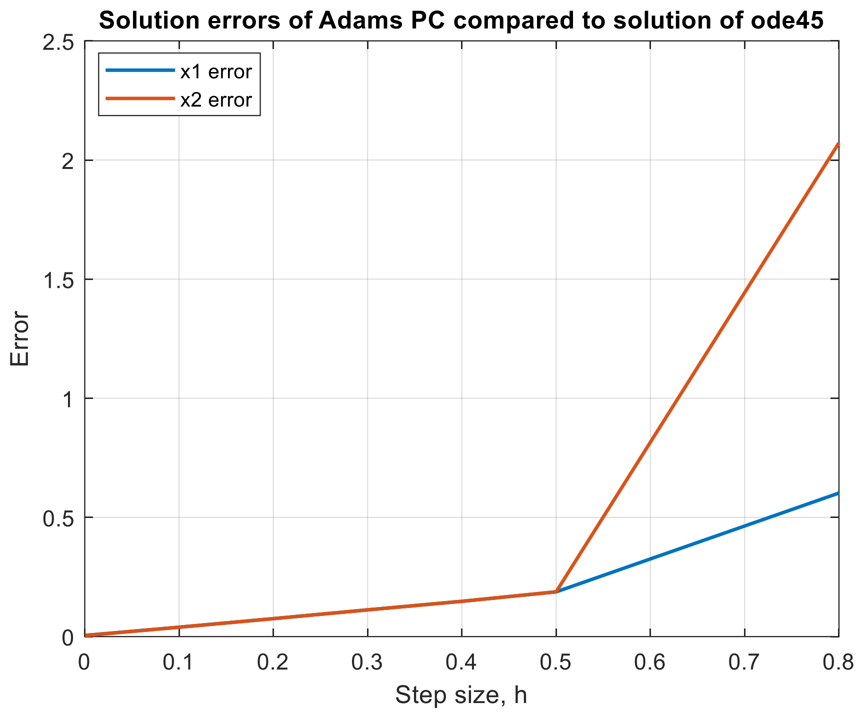
In the Adams PC method, the errors are calculated considering the solution of ode45 solver as the true value. The optional step size chosen is 0.001 below which the error change is negligible. Table 5 shows how the error increases with increasing step size used in the Adams PC method.

Table 5: Summary of the errors of Adams PC with respect to the step size

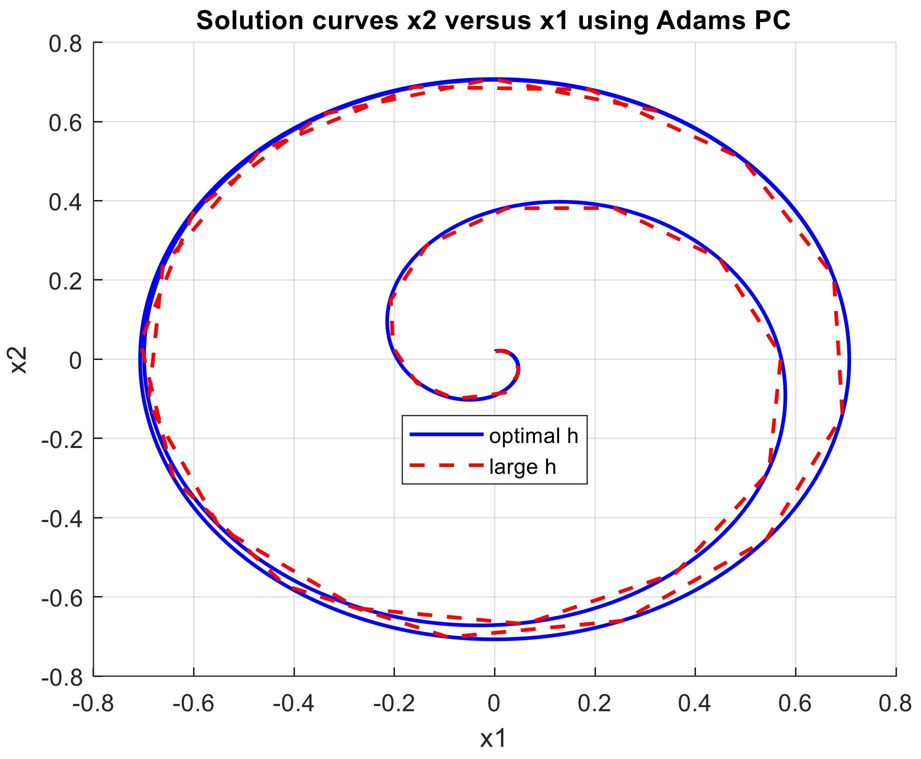
Step size for Adams PC	Error x1	Error x2
0.8	0.60231	2.072
0.5	0.18759	0.18794
0.4	0.14802	0.14799
0.3	0.1121	0.11197
0.2	0.075233	0.075146
0.1	0.039295	0.039278

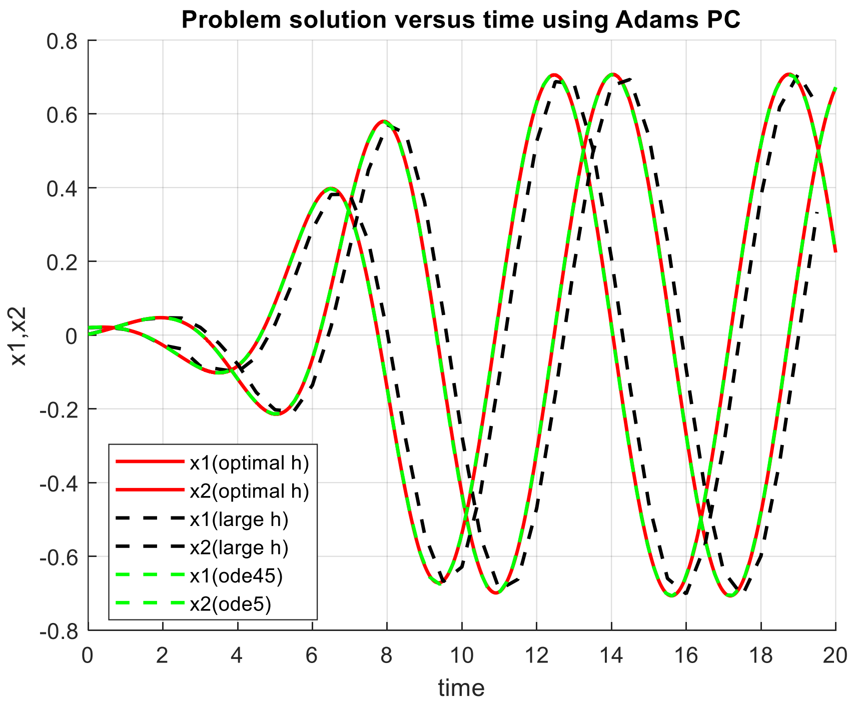
0.01	0.008292	0.007587
0.005	0.006682	0.005856
0.001	0.00543	0.004747
0.0005	0.005277	0.00474

The data in Table 5 is further supported by plotting the error vs step size. The plot also shows the error increases as the step size increases.



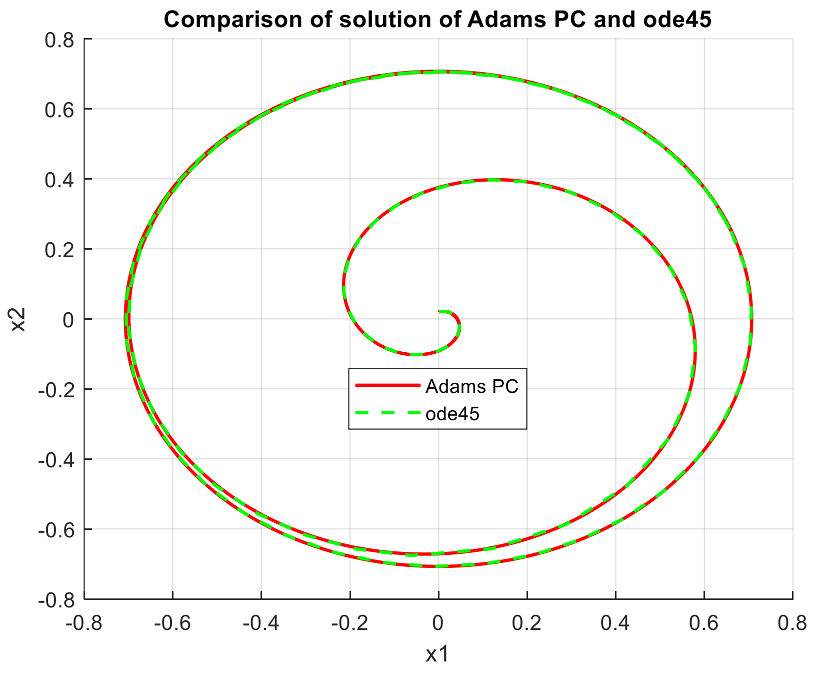
A plot of the solution curve x2 versus x1 is very smooth when optional step size chosen is used. Low accuracy solution is achieved when a large step size is used in the method.





Similarly, the solution plot versus time shows the high accuracy is obtained when optimal step size used compared to a large step size use. The solution of ode45 solver overlaps the solution of

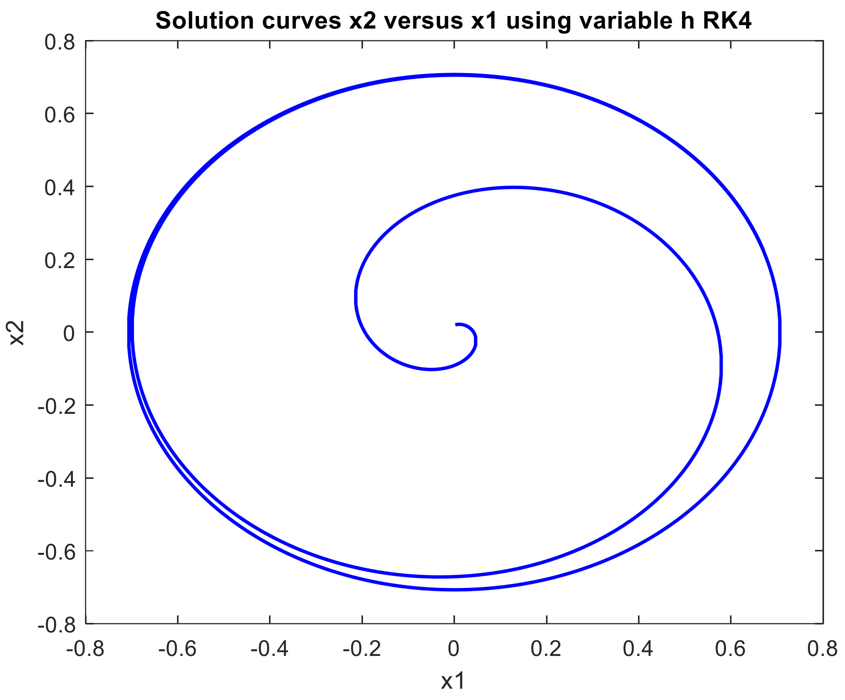
Adams PC with optimal step size. Also the same is observed with the plot of the x2 versus x2 when plotted on the axis with the solution of ode45 solver.

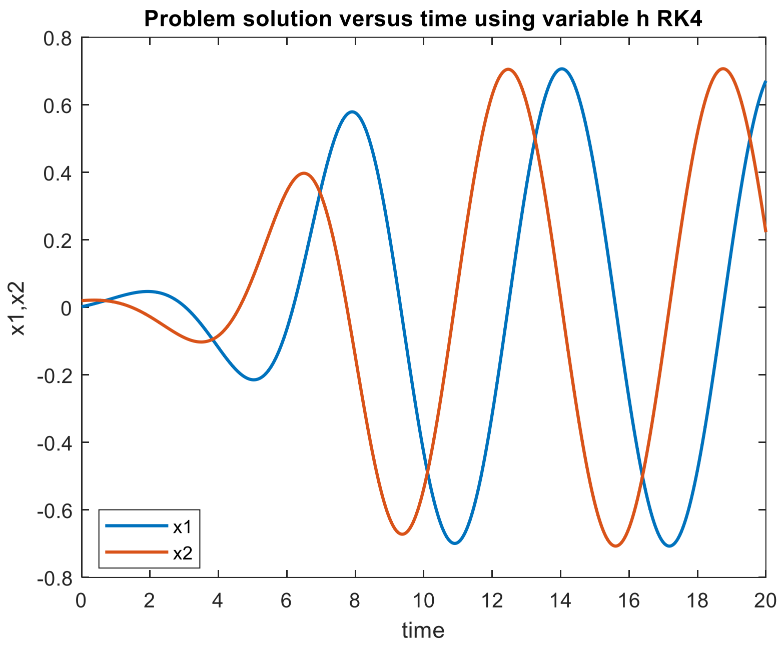


RK4 with automatic variable step size

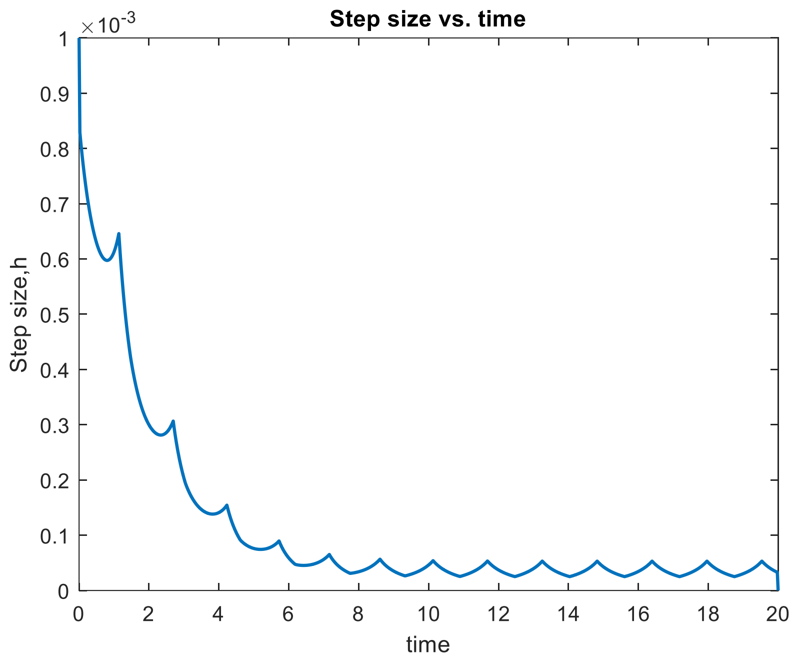
The adaptive methods takes a long time to run a total of 42899173 iterations in about 1.7473e+03 seconds. The adaptive control used error tolerances of 10^{-6}, safety factor of 0.9 and minimum step size of 10^{-6}. The chosen minimum step size and error tolerances are chosen such that, lower values does not result to any significant improvement in solution accuracy.

The solution curves produced are smooth with a higher accuracy. This result is comparable to ode45 solutions from the previous solutions. A plot of the solution as a function of time also shows the method produces the correct curve compared to ode45 solution and the solution of the constant step size methods using the optimal step size.

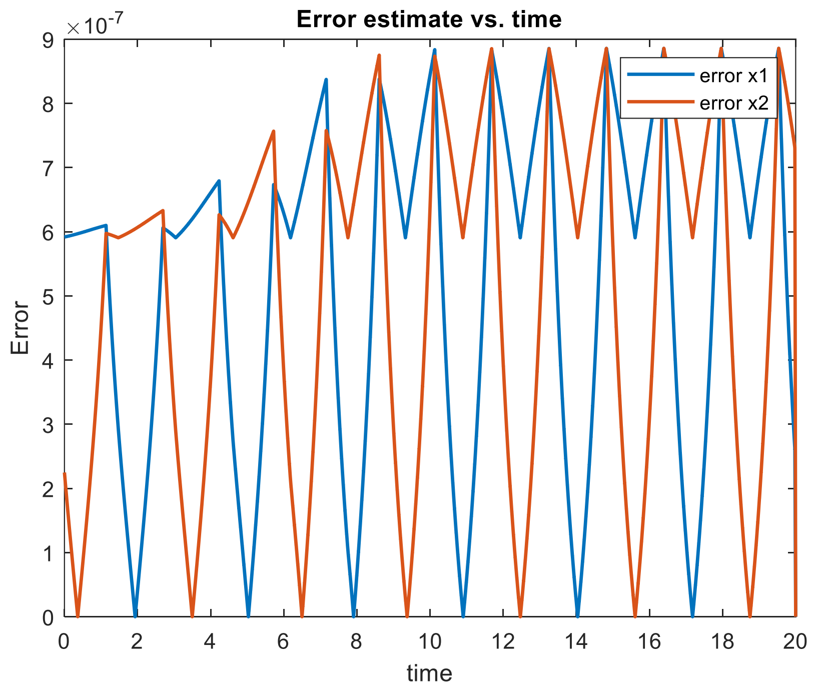




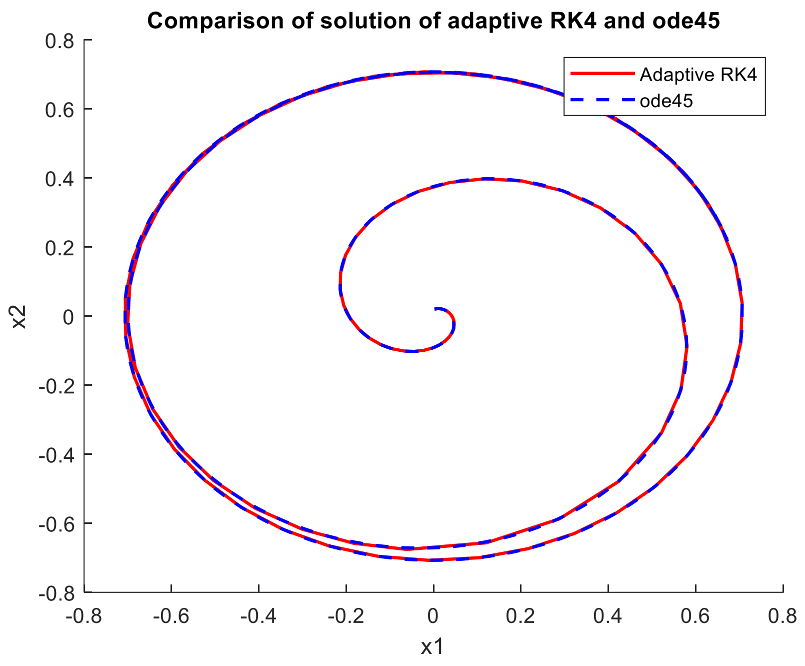
The algorithm starts with a relatively large step size which reduces over time. The decrease of step size is not linear with time, but instead it increases and decreases within short time intervals. After a much longer time, the step size oscillations between two levels. At this point, there is no further improvement of solution accuracy that can be achieved.

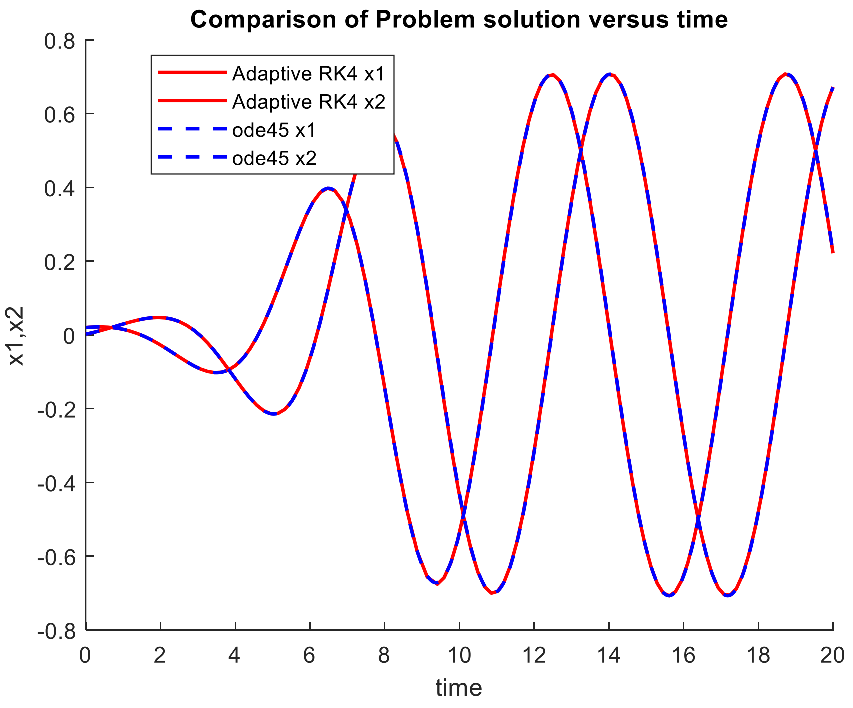


The oscillations of the solution errors occurs because of the oscillations of the step sizes that happens in the adjacent iterations. The errors are tames to remain within a predefined tolerance.



A comparison of the adaptive method to ode45 solution shows that there is negligible difference between the solutions as shown in the plots of the solution x2 versus x1 and solution versus time.





Appendix

Problem-I CODE

Task_I.m

```
% Problem I:
% We are required to determine a polynomial function y=f(x) that best fits
% the experimental data by using the least-squares approximation by testing
% polynomials of various degrees

% Clear and prepare the Matlab workspace
clc;clear all;close all
% The following are the experimental measurements (samples):
x = (-5:5);
y = [-4.6643,-5.5445,-4.0378,-2.0868,-1.0393,0.6916,-0.0237,0.4107,...
    -3.6761,-9.8466,-18.8868];



%% Least Square approximation
% To solve the least-squares problem use the system of normal equations
% with QR factorization of a matrix A.For each solution calculate the error
% defined as the Euclidean norm of the vector of residuum and the condition
% number of the Gram's matrix. Compare the results in terms of solutions’
% errors

N = length(x)-1; % Highest polynomial degree depends of number of samples
y_fit = zeros(length(x),N); % Initialize variable to store fitted values
CNum = zeros(N,1); % Initialze a varible to store condition numbers
% Euclidean norm of the vector of residuum with A matrix
ENorm_A = zeros(N,1);
% Euclidean norm of the vector of residuum with Gram's Matrix

ENorm_G = zeros(N,1);
for degree_n=1:N
    % Generate a system of normal equations for polynomial order j
    [A,b]= SystemOfEquations(x,y,degree_n);
    % solve the least-squares problem use the system of normal equations
    % with QR factorization of a matrix A
    LS_Sol = QR_solution(A,b);% Least Square (LS) Solutions
    y_fit(:,degree_n) = polyval(fliplr(LS_Sol(:)'),x(:));

    % Euclidean norm of the vector of residuum with Gram's Matrix
    GM = A'*A;                              % Gram's matrix
    CNum(degree_n) = cond(GM);              % Cond No.of Gram's matrix
    residuum_gm = GM*LS_Sol(:) - (A'*b);    % Residuum
    ENorm_G(degree_n) = norm(residuum_gm,2);% Euclidean norm

    % Euclidean norm of the vector of residuum with A matrix
    residuum_A = A*LS_Sol(:) - b;           % Residuum
    ENorm_A(degree_n) = norm(residuum_A,2); % Euclidean norm
    clear coeff A
end
```

```
% Display the results in a table format
PolynomialDegree=[1:10]';
Results = table(PolynomialDegree,ENorm_A,CNum, ENorm_G,...
    'VariableNames',...
    {'Polynomial Degree', 'Euclidean Norm of residuum of A matrix',...
    'Condition number of Gram''s matrix', ...
    'Euclidean Norm of residuum of Gram''s Matrix'})

%% Plot the results
% Plot the solutions for of the polynomials of various degrees)
p_order ={'First','Second','Third','Fourth','Fifth','Sixth','Seventh',...
    'Eigth','Ninth','Tenth'};
for j=1:N
    figure(); % Generate a new figure window for each polynomial degree
    hold on % Use hold to place different plots on the same axis
    plot(x,y,'r*','markersize',10); % Plot the data samples
    plot(x,y_fit(:,j),'-b','linewidth',2)% Plot the fitted data
    xlabel('x');ylabel('y')% Add axis lables
    grid on; % Add grid lines to the plot
    % Add a title heading to the plot
    TitleHead = ['Least Square fit : ',p_order{j},' order polynomial'];
    title(TitleHead)
    % Add legend for each plotted curve
    legend('data samples','Least Square fit')
    hold off % Allos the the plotted curves to shown in the figure window
end

% Plot the condition number associated with the matrix of each polynomial
% order
figure()                                    % Create a new figure window
plot(1:N,CNum,'linewidth',2)                % Plot the condition number
xlabel('polynomial order ')                 % Add x-axis label
ylabel('Condition number')                  % Add y-axis label
grid on                                     % Add grid lines to the plot
title('Condition number of Gram''s matrix') % Add title of graph

% Plot Euclidean Norm of residuum of A matrix
figure()
plot(1:N,ENorm_A,'linewidth',2)                 % Add the plot
xlabel('polynomial degree ')                    % Add x-axis label
ylabel('Error')                                 % Add y-axis label
grid on                                         % Add grid lines on plot
title('Euclidean Norm of residuum of A matrix') % Add title of graph

% Plot the Error: Euclidean Norm of residuum of Gram's matrix
figure()                                        % Create a new figure window
plot(1:N,ENorm_G,'linewidth',2)                 % Add the plot
xlabel('polynomial degree ')                    % Add x-axis label
ylabel('Error')                                 % Add y-axis label
grid on                                         % Add grid lines on plot
title('Euclidean Norm of residuum of Gram''s matrix')% Add title

%% Functions
function [A,b]=SystemOfEquations(x,y,Poly_degree)
x = x(:); % data samples
b = y(:); % function values at data sample points
```

```
% Initialize the coefficients matrix of normal equations
A = [ones(size(x))];
for i=1:Poly_degree
    A(:,i+1)=x.^i;
end
end
function coeff=QR_solution(A,b)
b = b(:);          % Ensure it a a column vector
[~,n] = size(A); % Get the number of columns of matrix A
[Q,R,P] = qr(A); % Apply QR factorization
d = Q'*b;          % Compute matrix-vector product

% Apply back substitution
x = zeros(n,1); % Initialize vector x
for i = n:-1:1
    x(i) = d(i);
    for j = i+1:n
        x(i) = x(i) - R(i,j)*x(j);
    end
    x(i) = x(i)/R(i,i);
end
coeff = P*x; % Re-order the coefficients using permutation matrix from QR
end
```

Problem-II CODE

Problem_IIa_RK4.m

```
% Problem II:
% Determine the trajectory of the motion on the interval [0, 20] for the
% following initial conditions: x1(0) = 0.002, x2(0) = 0.02. Evaluate the
% solution using: a) Runge-Kutta method of 4th order (RK4)
clc;close all;clear all
% Define the Equations

f = @(t,x) [x(2)+x(1)*(0.5-x(1)^2-x(2)^2);
    -x(1)+x(2)*(0.5-x(1)^2-x(2)^2)];
% Define the initial conditions
t0 = 0;               % Start Time
tf = 20;              % End time
interval = [t0,tf]; % Time interval
x0 =[0.002,0.02];    % x0 = [x1(0), x2(0)]

%% Solution
% Evaluate the solution using different constant step-sizes
ode45_sol = ode45(f,interval,x0);
dh = 0.05:0.1:1.1;
sol_error = zeros(length(dh),2);
for i=1:length(dh)
    [ rk4_t, rk4_sol ] = RK4_h_constant(f, t0, tf, x0, dh(i));
    ode45_sol_x = deval(ode45_sol,rk4_t);
    abs_error = abs(rk4_sol-ode45_sol_x');
    sol_error(i,1)=max(abs_error(:,1));
    sol_error(i,2)=max(abs_error(:,2));
end
% Compare errors for each step size.
table(dh(:),sol_error(:,1),sol_error(:,2),...
    'variablenames',{'h','del_x1','del_x2'})
% Plot the error trend with increasing step size, h
figure()                                % Create a new figure window
plot(dh,sol_error,'linewidth',1.5)      % Add the plot on the figure
grid on                                 % Add grid to the plot
xlabel('Step size, h');ylabel('Error')  % Add axis labels
legend('x1 error','x2 error','location','best')% Add a legend
title('Solution errors of RK4 compared to solution of ode45')  % Add title

%% Plot Solution curves x2 versus x1
[t45,ode45_sol_x] = ode45(f,interval,x0);
[ Trk4_optimal, Xrk4_optimal ] = RK4_h_constant(f, t0, tf, x0, 0.1);
[ Trk4_large, Xrk4_large ] = RK4_h_constant(f, t0, tf, x0, 0.75);
figure() % Create a new figure window
hold on  % holds the current plot and all axis properties
plot(Xrk4_optimal(:,1),Xrk4_optimal(:,2),'-b','linewidth',1.5)
plot(Xrk4_large(:,1),Xrk4_large(:,2),'--r','linewidth',1.5)
title('Solution curves x2 versus x1 using RK4') % Add title
xlabel('x1');ylabel('x2')                       % Add axis labels
legend('optimal h','large h','location','best') % Add legend
grid on                                         % Add grid lines
hold off
```

```
%% Plot Problem solution versus time
figure() % Create a new figure window
hold on  % holds the current plot and all axis properties
plot(Trk4_optimal,Xrk4_optimal,'-r','linewidth',1.5)
plot(Trk4_large, Xrk4_large,'--k','linewidth',1.5)
plot(t45, ode45_sol_x,'--g','linewidth',1.5)
title('Problem solution versus time using RK4') % Add title
xlabel('time');ylabel('x1,x2')                 % Add axis labels
legend('x1(optimal h)','x2(optimal h)','x1(large h)','x2(large h)',...
    'x1(ode45)','x2(ode5)','location','best')   % Add legend
grid on                                         % Add grid lines
hold off
%% Compare the results with the ones obtained using ode45.
figure() % Create a new figure window
hold on  % holds the current plot and all axis properties
plot(Xrk4_optimal(:,1),Xrk4_optimal(:,2),'-r','linewidth',1.5)
plot(ode45_sol_x(:,1),ode45_sol_x(:,2),'--g','linewidth',1.5)
title('Comparison of solution of RK4 and ode45')    % Add title
xlabel('x1');ylabel('x2')                           % Add axis labels
legend('RK4','ode45','location','best')             % Add legend
grid on                                             % Add grid lines
hold off
```

Runge-Kutta function with constant step size

```
function [ T, Y ] = RK4_h_constant( f, t0, tf, y0, h)
T = (t0 : h : tf)'; % Time vector
Y = zeros(length(T),length(y0)); % Initialize solution vector
Y(1,1) = y0(1); % Append initial conditions to the solution vector
Y(1,2) = y0(2);% Append initial conditions to the solution vector
% Implementing the Runge-Kutta method of 4th order (RK4) algorithm
for i = 2 : length(T)
    k1 = f(T(i-1),Y(i-1,:))';
    k2 = f(T(i-1)+0.5*h,Y(i-1,:)+0.5*h*k1)';
    k3 = f(T(i-1)+0.5*h,Y(i-1,:)+0.5*h*k2)';
    k4 = f(T(i-1)+h,Y(i-1,:)+h*k3)';
    Y(i,:) = Y(i-1,:) + (h/6) * (k1 + 2*k2 + 2*k3 + k4);
end
end
```

Problem_IIa_Adams script

```
% Problem II:
% Determine the trajectory of the motion on the interval [0, 20] for the
% following initial conditions: x1(0) = 0.002, x2(0) = 0.02. Evaluate the
% solution using: a) Adams PC (P5EC5E)
clc;close all;clear all
% Define the Equations
f = @(t,x) [x(2)+x(1)*(0.5-x(1)^2-x(2)^2);
```

```
    -x(1)+x(2)*(0.5-x(1)^2-x(2)^2)];
% Define the initial conditions
t0 = 0;             % Start Time
tf = 20;            % End time
interval = [t0,tf]; % Time interval
x0 =[0.002,0.02];   % x0 = [x1(0), x2(0)]
%% Solution
% Evaluate the solution using different constant step-sizes
ode45_sol = ode45(f,interval,x0);
dh = [0.8,0.5,0.4,0.3,0.2,0.1,0.01,0.005,0.001,0.0005];
sol_error = zeros(length(dh),2);
for i=1:length(dh)
    [ T_adams, X_Adams ] = Adams_P5EC5E(f, interval,x0, dh(i));
    ode45_sol_x = deval(ode45_sol,T_adams);
    abs_error = abs(X_Adams-ode45_sol_x');
    sol_error(i,1)=max(abs_error(:,1));
    sol_error(i,2)=max(abs_error(:,2));
end
% Compare errors for each step size.
table(dh(:),sol_error(:,1),sol_error(:,2),...
    'variablenames',{'h','del_x1','del_x2'})
% Plot the error trend with increasing step size, h
figure()                                % Create a new figure window
plot(dh,sol_error,'linewidth',1.5)      % Add the plot on the figure
grid on                                 % Add grid to the plot
xlabel('Step size, h');ylabel('Error')  % Add axis labels
legend('x1 error','x2 error','location','best')% Add a legend
title('Solution errors of Adams PC compared to solution of ode45')% title
%% Plot Solution curves x2 versus x1
[t45,ode45_sol_x] = ode45(f,interval,x0);
[ Tadams_optimal, Xadams_optimal ] = Adams_P5EC5E(f, interval,x0, 0.001);
[ Tadams_large, Xadams_large ] = Adams_P5EC5E(f, interval,x0, 0.5);
figure() % Create a new figure window
hold on  % holds the current plot and all axis properties
plot(Xadams_optimal(:,1),Xadams_optimal(:,2),'-b','linewidth',1.5)
plot(Xadams_large(:,1),Xadams_large(:,2),'--r','linewidth',1.5)
title('Solution curves x2 versus x1 using Adams PC') % Add title
xlabel('x1');ylabel('x2')                       % Add axis labels
legend('optimal h','large h','location','best') % Add legend
grid on                                         % Add grid lines
hold off
%% Plot Problem solution versus time
figure() % Create a new figure window
hold on  % holds the current plot and all axis properties
plot(Tadams_optimal,Xadams_optimal,'-r','linewidth',1.5)
plot(Tadams_large, Xadams_large,'--k','linewidth',1.5)
plot(t45, ode45_sol_x,'--g','linewidth',1.5)
title('Problem solution versus time using Adams PC') % Add title
xlabel('time');ylabel('x1,x2')                  % Add axis labels
legend('x1(optimal h)','x2(optimal h)','x1(large h)','x2(large h)',...
    'x1(ode45)','x2(ode5)','location','best')   % Add legend
grid on                                         % Add grid lines
hold off
%% Compare the results with the ones obtained using ode45.
figure() % Create a new figure window
hold on  % holds the current plot and all axis properties
plot(Xadams_optimal(:,1),Xadams_optimal(:,2),'-r','linewidth',1.5)
plot(ode45_sol_x(:,1),ode45_sol_x(:,2),'--g','linewidth',1.5)
title('Comparison of solution of Adams PC and ode45')    % Add title
xlabel('x1');ylabel('x2')                                % Add axis labels
legend('Adams PC','ode45','location','best')             % Add legend
grid on                                                  % Add grid lines
hold off
```

Adams_P5EC5E function

```
function [T,Y] = Adams_P5EC5E(dyfun, tspan, y0, h)
k = 5; % Use 5 steps in the preictor and collector
% Runge-Kutta method is used to Initialize the first 5 points
ti = tspan(1):h:tspan(2);
[~,yy] = RK4_h_constant(dyfun,ti(1),ti(5),y0,h);
Y(1:5, :)= yy;
T(1:5) = ti(1:5);

% Predictor function
yn0 = @(y, f, h) (y + h/720*(1901*f(end,:)-2774*f(end-1,:) + ...
    2616*f(end-2,:) - 1274*f(end-3,:) + 251*f(end-4,:)));

% Corrector function
yn = @(y, f, h, fn0) (y + h/1440*(1427*f(end,:) - 798*f(end-1,:) + ...
    482*f(end-2,:) - 173*f(end-3,:) + 27*f(end-4,:)) + h*475/1440*fn0);

f = zeros(length(Y), 2); % Initialize f same size as X with zeros
N = (tspan(2) - tspan(1)) / h; % Total number of steps
% Adams PC (P5EC5E)
for i = k+1:N
    T(i) = T(i-1) + h;
    % P-prediction
    P = yn0(Y(i-1, :), f(1:i-1, :), h);

    % E-Evaluation
    f(i,:) = dyfun(T(i),P)';
    % C-Correction
    C = yn(Y(i-1, :), f(1:i-1, :), h, f(i, :));

    % E-Evaluation
    f(i,:) = dyfun(T(i),C)';

    Y(i, :) = C;
end
T = T(:);
end
```

Problem_IIb_RK4_.m

```
% Problem II: Determine the trajectory of the motion on the interval [0,
% 20] for the following initial conditions: x1(0) = 0.002, x2(0) = 0.02.
% Evaluate the solution using Runge-Kutta method of 4th order (RK4) with a
% variable step size automatically adjusted by the algorithm, making error
% estimation according to the step-doubling rule.
clc;close all;clear all
interval = [0 20];      % Time interval
x0 =[0.002,0.02];       % x0 = [x1(0), x2(0)]

% Define the Equations
f = @(t,x) [x(2)+x(1)*(0.5-x(1)^2-x(2)^2);
    -x(1)+x(2)*(0.5-x(1)^2-x(2)^2)];

% Record execution time
tStart = tic;
[X, T,h, err, iter] = RK4_variable_h(f,interval, x0);
tEnd = toc(tStart);
display(iter,'Number of iterations')
display(tEnd ,'Time of execution')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Call ode45 solver
[t45,x45] = ode45(f,interval,x0);

%% Plot Problem solution versus time--
figure()% Create a new figure window
plot(X(:,1),X(:,2),'-b','linewidth',1.5)
title('Solution curves x2 versus x1 using variable h RK4')
xlabel('x1');ylabel('x2') % Add axis labels
%% Plot problem solution vs time
figure()    % Create a new figure window
plot(T,X,'linewidth',1.5)    % Plot problem solution vs time
title('Problem solution versus time using variable h RK4')
xlabel('time');ylabel('x1,x2')      % Add axis labels
legend('x1','x2','location','best') % Add legend

%% Plot step size vs time
figure()% Create a new figure window
plot(T,h,'linewidth',1.5)% Plot step size vs time
title('Step size vs. time')% Add title
xlabel('time');ylabel('Step size,h') % Add axis labels

%% Plot error estimate vs time
figure()% Create a new figure window
plot(T(1:end-1),err,'linewidth',1.5)% Plot step size vs time
title('Error estimate vs. time') % Add title
xlabel('time');ylabel('Error')  % Add axis labels
legend('error x1','error x2')   % Add legend

%% Compare adaptive RK4 and ode45
figure()% Create a new figure window
hold on
plot(x45(:,1),x45(:,2),'-r','linewidth',1.5)
plot(X(:,1),X(:,2),'--b','linewidth',1.5)
title('Comparison of solution of adaptive RK4 and ode45')
```

```
xlabel('x1');ylabel('x2') % Add axis labels
legend('Adaptive RK4','ode45') % Add legend
hold off

%% Compare Adaptive RK4 and ode45
figure()% Create a new figure window
hold on
plot(t45, x45,'-r','linewidth',1.5)
plot(T,X,'--b','linewidth',1.5)
title('Comparison of Problem solution versus time')
xlabel('time');ylabel('x1,x2') % Add axis labels
legend('Adaptive RK4 x1','Adaptive RK4 x2','ode45 x1','ode45 x2',...
    'location','best') % Add legends
hold off
```

Adaptive RK4 function

```
function [X,t, h, X_err, counter] = RK4_variable_h(f,interval, y0)
beta = 0.9;             % Safety factor
abs_err = 1e-6;         % Absolute error tolerance
rel_err = 1e-6;         % Relative error tolerance
hmin = 1e-6;            % Set the Minimum step size
cf = 4;                 % Converge order of RK4 method
dh = 0.001;             % Set the initial step size
X(1, :) = y0;           % Record the initial solution
X_err(1,:) = [0, 0];    % Record starting error as zero
t = interval(1);        % Set starting time of the solution
j = 1;                  % usinh symbol  as steps counter
counter = 0;            % Starting counting iteration from 1 (default=0)
h(1) = dh;              % Record first step size

% Apply step-doubling rule to vary the step sizes
while t(j)~= interval(2)
    [ ~, YY ] = RK4_h_constant( f, t(j), t(j)+dh, X(j, :), dh);%Step-size =
h
    x1 = YY(end,:);
    dhh = dh/2; %Step-size = h/2
    [ ~, YY ] = RK4_h_constant( f, t(j), t(j)+dhh, X(j, :), dhh);
    x2 = YY(end,:);

    delta = abs((x2 - x1))./(2.^cf - 1);
    Error_estimate = abs(x2)*rel_err + abs_err; % Error estimate
    X_err(j, :) = delta; % Record Error
    da = (min(Error_estimate./delta)).^(1/(cf+1)); % Correct step size

    B = beta*da;% Calculate Safety factor
    htest = B*dh; % Calculate new step size
    if beta*da>= 1
        t(j+1) = t(j) + dh/2;    % Update time
        X(j+1, :) = x2;          % Update solution
        dh = min([htest, 2*dh, interval(2)- t(j+1)]); % Step size to be used
        j = j + 1; % Increament step counter
        h(j) = dh; % Record the step size
    else
        if htest < hmin
            disp('Error!:Minimum stepp size reached!.');
            break;
        else
            dh = htest;
        end
    end
    counter = counter + 1; % Update iteration counter
end
end
```