ENUME Project C

No. 8

Fengxi Zhao

317342

January 2024

Task 1:

I For the following experimental measurements (samples):

x_i	y_i
–5	–77.9639
–4	–39.5900
–3	–17.5814
–2	–5.1530
–1	0.7608
0	2.0270
1	1.2585
2	–0.5477
3	–2.2384
4	–5.1580
5	–8.1875

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.

Algorithm:

First, we create Gram's matrix based on the data and degree, then QR factorize it and use back substitution to solve it like in Project A. We plot the polynomial functions with different degrees until the error is small enough.

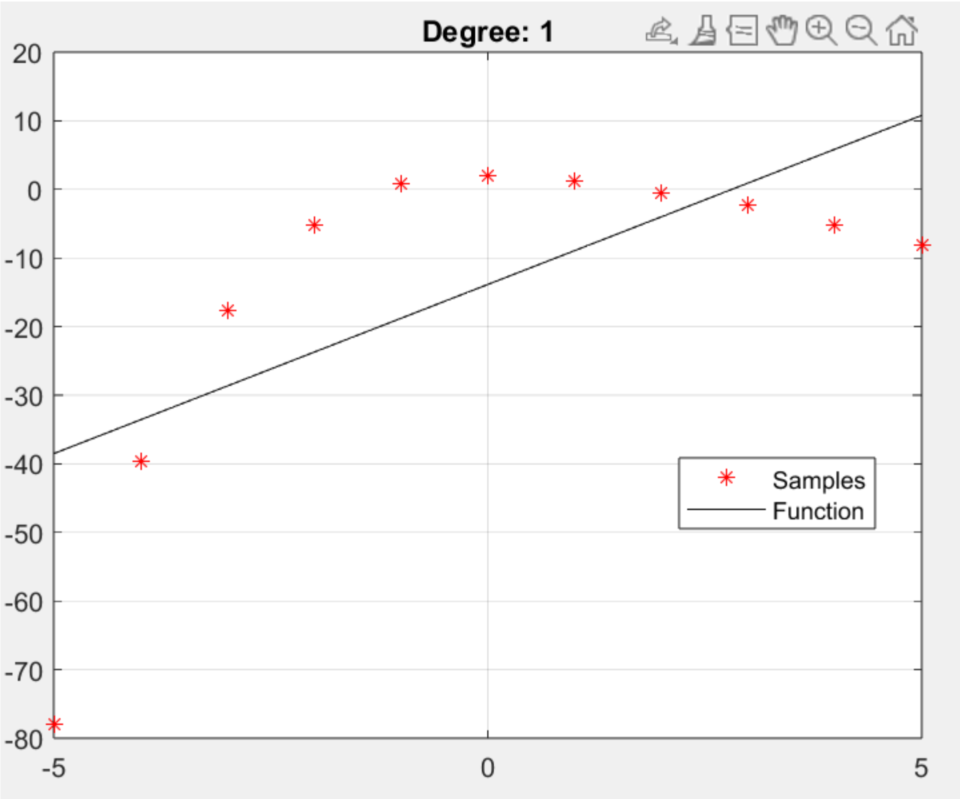
The polynomial is:

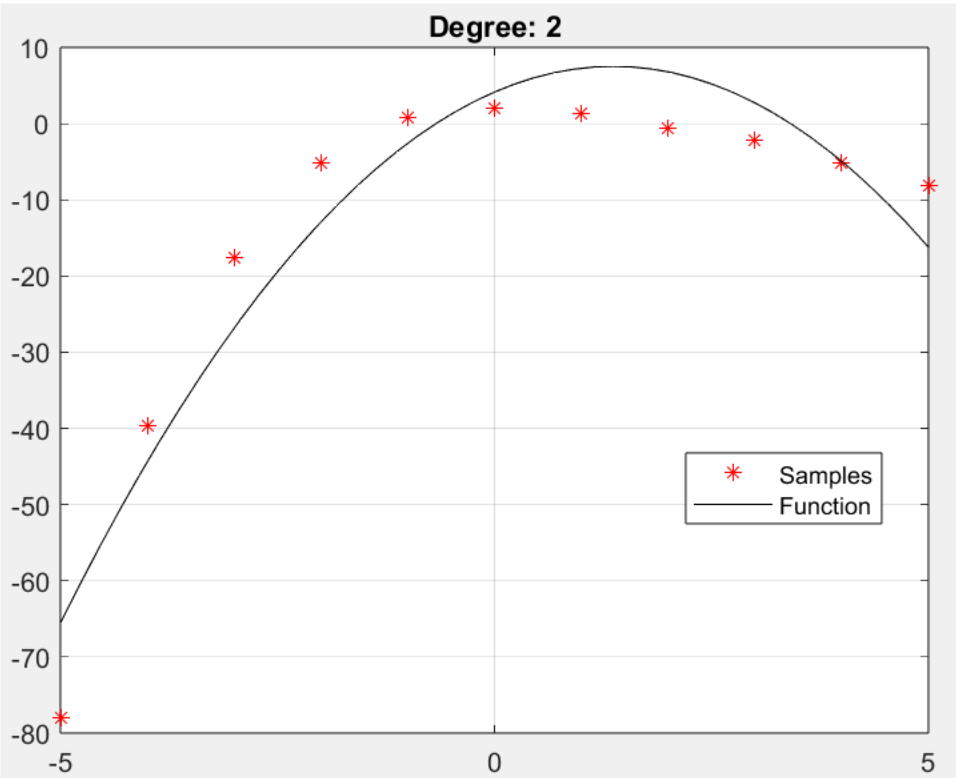
$$\forall F \in X_n \quad F(x) = \sum_{i=0}^{n} a_i \phi_i(x) . \qquad (4.5)$$

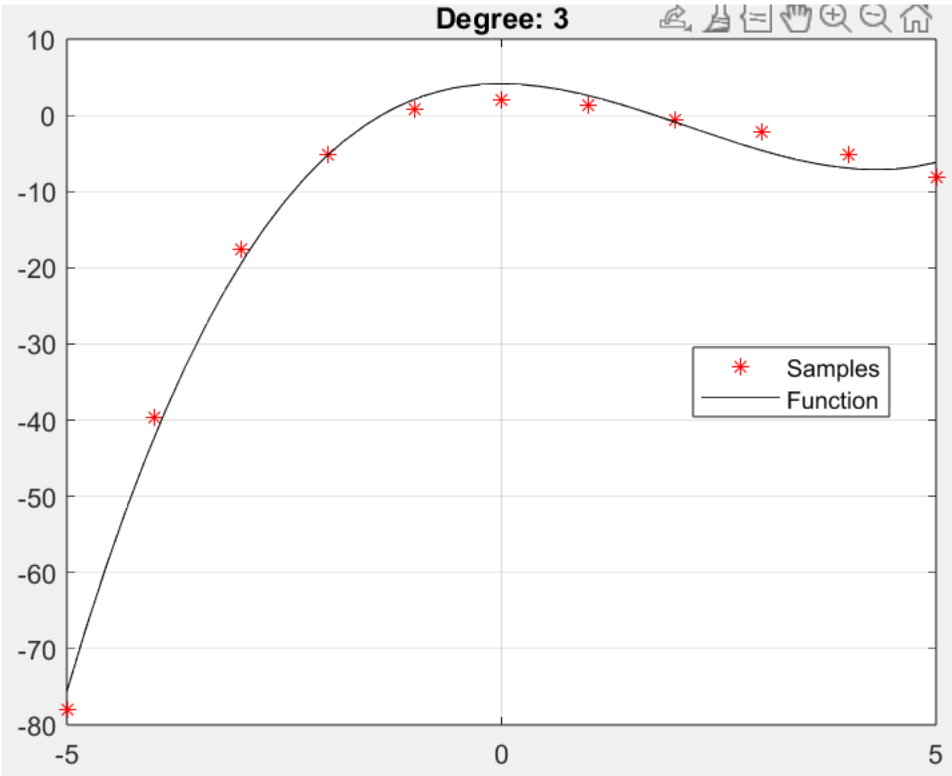
The equations to be solved:

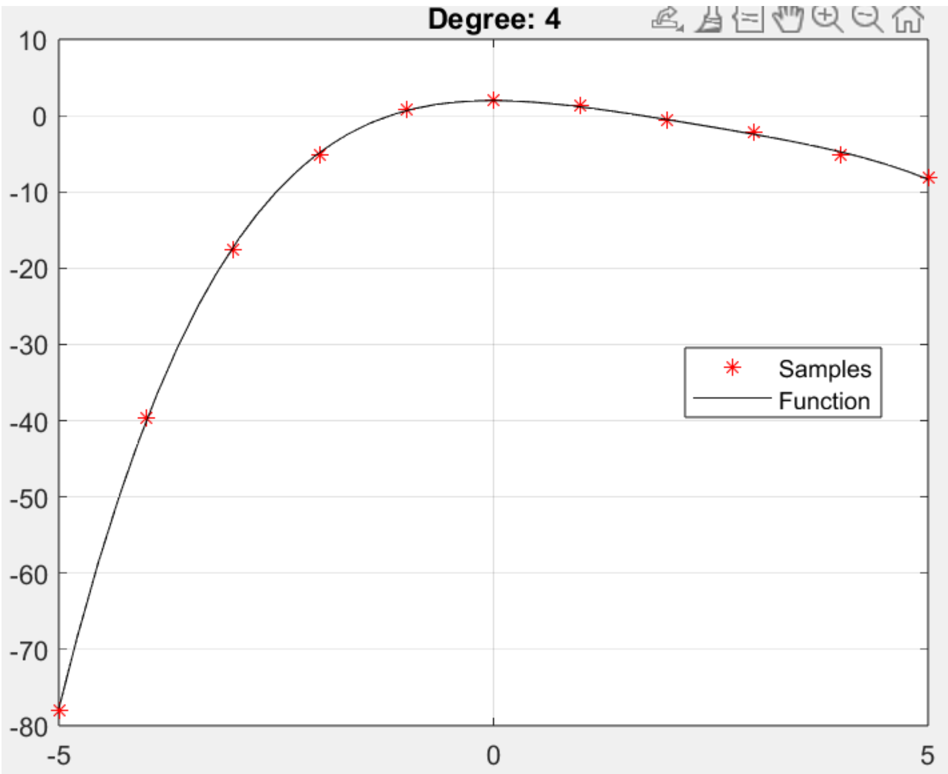
$$\begin{bmatrix} \langle \phi_0, \phi_0 \rangle & \langle \phi_1, \phi_0 \rangle & \cdots & \langle \phi_n, \phi_0 \rangle \\ \langle \phi_0, \phi_1 \rangle & \langle \phi_1, \phi_1 \rangle & \cdots & \langle \phi_n, \phi_1 \rangle \\ \vdots & \vdots & \vdots & \vdots \\ \langle \phi_0, \phi_n \rangle & \langle \phi_1, \phi_n \rangle & \cdots & \langle \phi_n, \phi_n \rangle \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \langle \phi_0, f \rangle \\ \langle \phi_1, f \rangle \\ \vdots \\ \langle \phi_n, f \rangle \end{bmatrix} . \qquad (4.8)$$

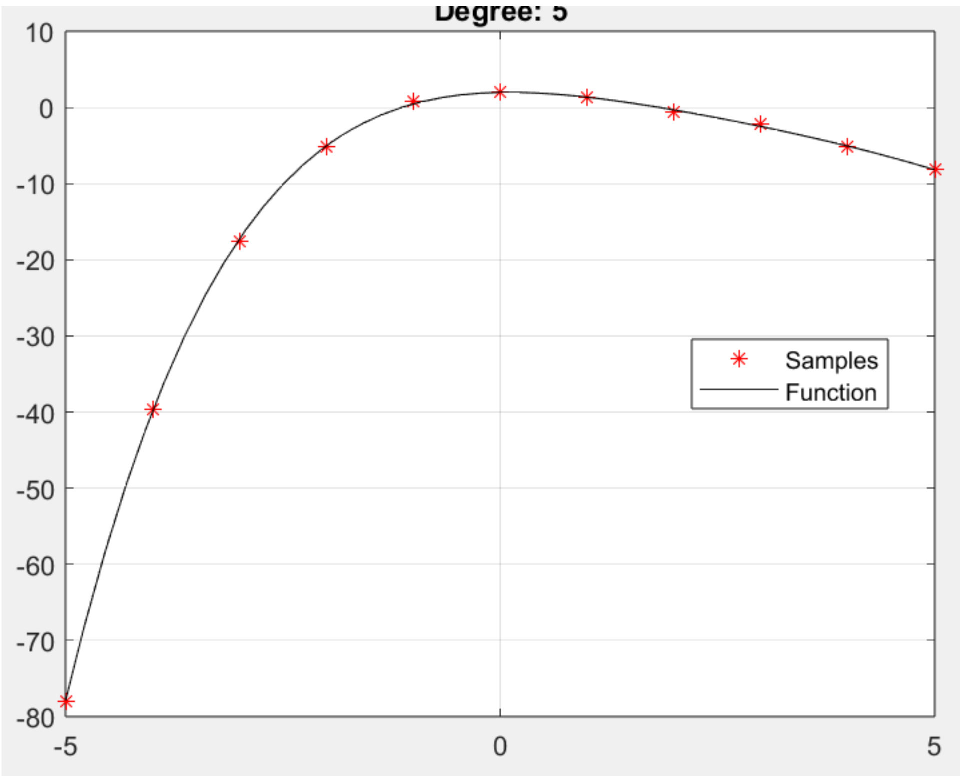
Result:

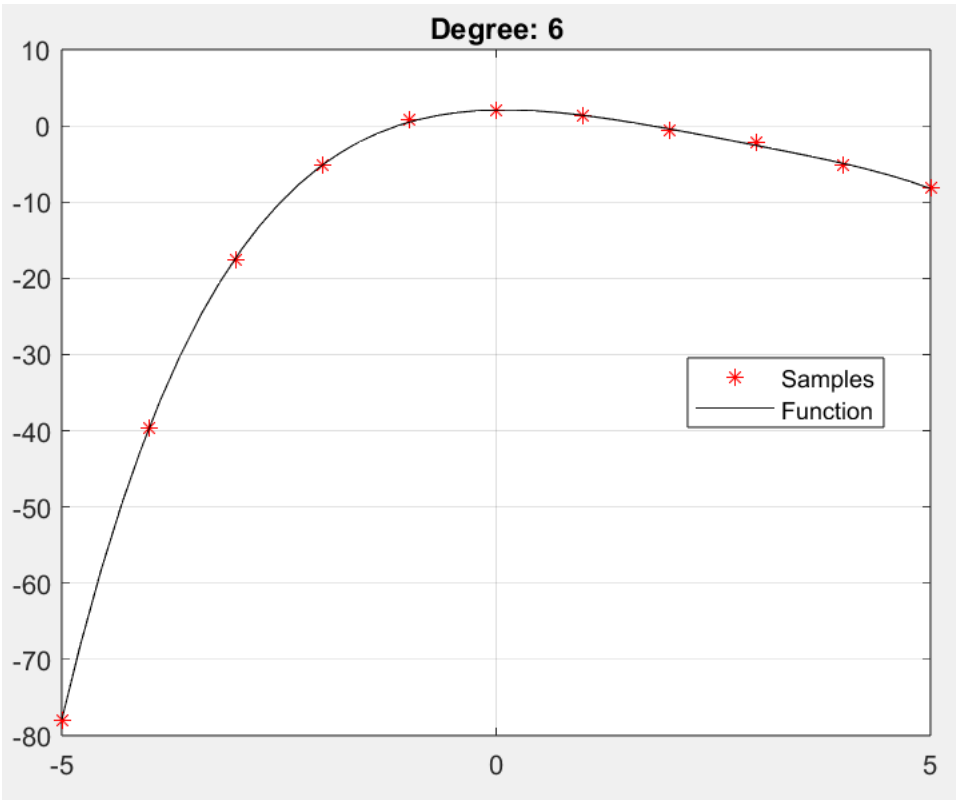

Degree: 1
20
10
0
-10
-20
-30
-40
-50
-60
-70
-80
-5
0
5
Samples
Function

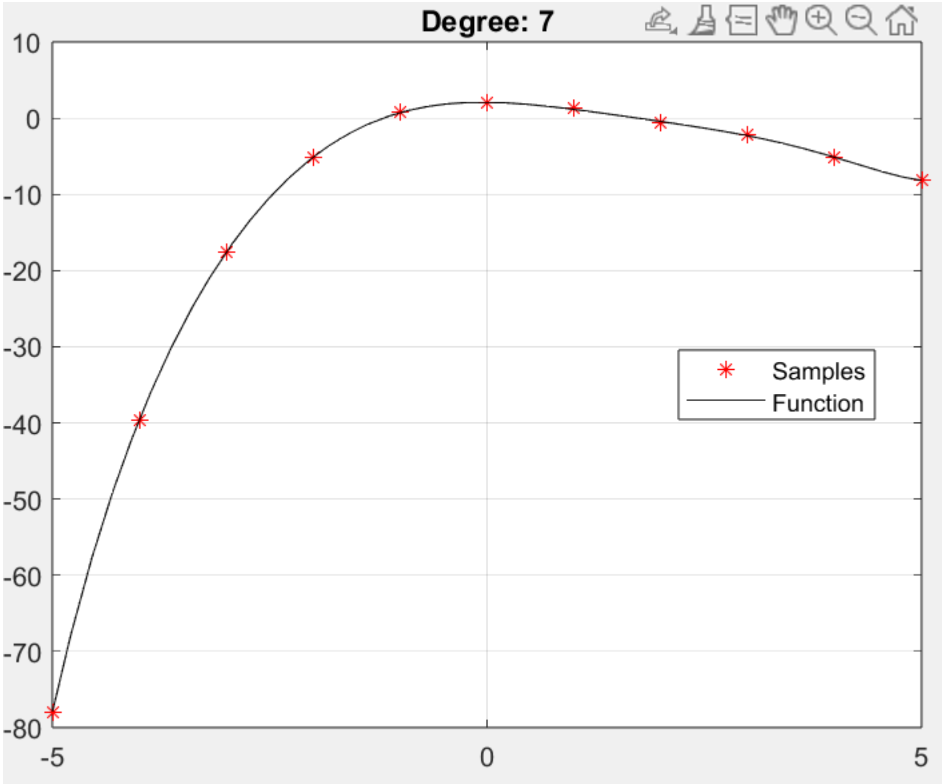

Degree: 2
10
0
-10
-20
-30
-40
-50
-60
-70
-80
-5
0
5
Samples
Function

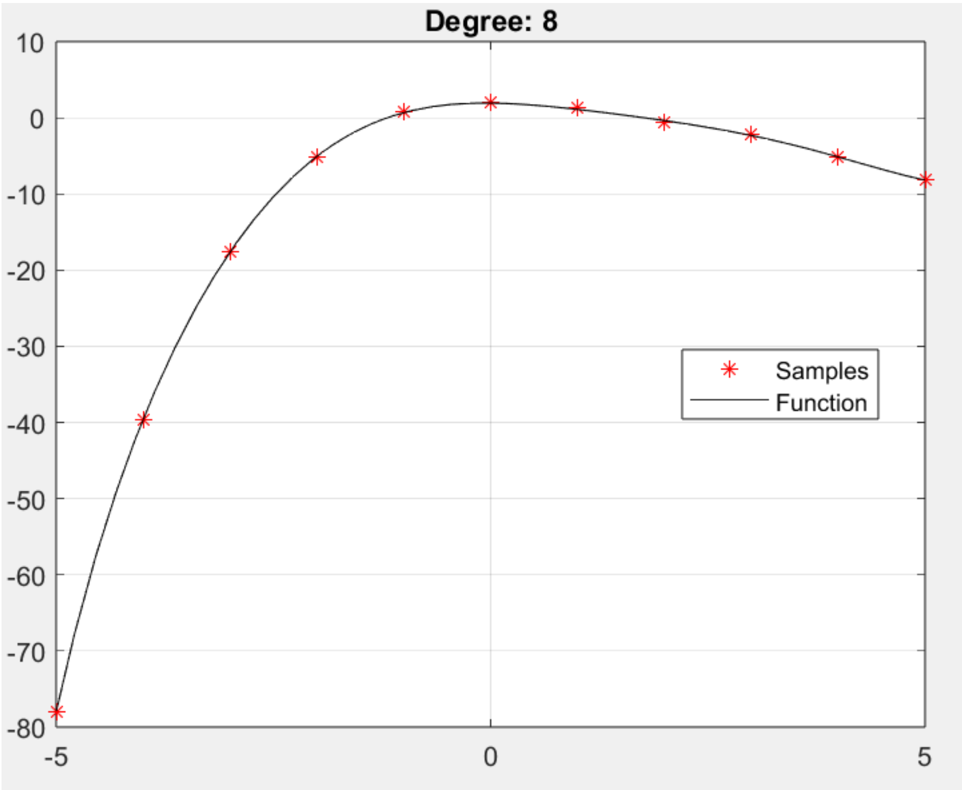

Degree: 3
10
0
-10
-20
-30
-40
-50
-60
-70
-80
-5
0
5
Samples
Function


Degree: 4
10
0
-10
-20
-30
-40
-50
-60
-70
-80
-5
0
5
Samples
Function


Degree: 5
Samples
Function


Degree: 6
Samples
Function


Degree: 7
10
0
-10
-20
-30
-40
-50
-60
-70
-80
-5
0
5
Samples
Function


Degree: 8
10
0
-10
-20
-30
-40
-50
-60
-70
-80
-5
0
5
Samples
Function

Comparison of errors and condition numbers:

Degree	Error	Condition number
1	0.23947	10
2	0.00039808	408.7796
3	6.9653e-05	8558.4366
4	4.7878e-07	317981.4472
5	7.7004e-09	7467495.6595
6	4.7716e-11	283155896.0339
7	2.3022e-12	7646220977.7267
8	0	330546434323.1818

Conclusion:

The increment of degree gives us more accurate approximation, as we can see from the graphs and errors. The condition number increases quite fast due to the size of the matrix. In this case 4^{th} degree or 5^{th} degree is acceptable on the graph.

Task 2:

II A motion of a point is given by equations:

$$dx_1/dt = x_2 + x_1 (0.5 - x_1^2 - x_2^2),$$
$$dx_2/dt = -x_1 + x_2 (0.5 - x_1^2 - x_2^2).$$

Determine the trajectory of the motion on the interval [0, 15] for the following initial conditions: $x_1(0) = 9$, $x_2(0) = 8$. Evaluate the solution using:

a) Runge-Kutta method of 4^{th} order (RK4) and Adams PC (P_5EC_5E) – each method a few times, with different constant step-sizes 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 4^{th} 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.

Algorithm:

To solve ODEs, there are single-step methods and multi-step methods.

In a single-step method, we check what happens in a point and around it, but forget about it in next iterations.

In a multi-step method, we also care about previous iterations.

Both of them are difference methods, which means there are discrete points.

The distance between them is called step size.

The Runge-Kutta method of order 4:

$$y_{n+1} = y_n + \frac{1}{6}h\left(k_1 + 2k_2 + 2k_3 + k_4\right), \tag{7.20a}$$
$$k_1 = f\left(x_n, y_n\right), \tag{7.20b}$$
$$k_2 = f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1), \tag{7.20c}$$
$$k_3 = f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2), \tag{7.20d}$$
$$k_4 = f\left(x_n + h, y_n + hk_3\right). \tag{7.20e}$$

Multistep method:

$$y_n = \sum_{j=1}^{k} \alpha_j \cdot y_{n-j} + h \sum_{j=0}^{k} \beta_j \cdot f\left(x_{n-j}, y_{n-j}\right),$$
$$y_0 = y(x_0) = y_a,\ x_n = x_0 + nh,\ x \in [a = x_0, b]. \tag{7.39}$$

For the Adams methods the PkECkE algorithm has the following form:

$$\begin{aligned}
&\text{P:} \quad && y_n^{[0]} = y_{n-1} + h \sum_{j=1}^{k} \beta_j f_{n-j}, \\
&\text{E:} \quad && f_n^{[0]} = f(x_n, y_n^{[0]}), \\
&\text{C:} \quad && y_n = y_{n-1} + h \sum_{j=1}^{k} \beta_j^* f_{n-j} + h\beta_0^* f_n^{[0]}, \\
&\text{E:} \quad && f_n = f\left(x_n, y_n\right).
\end{aligned}$$

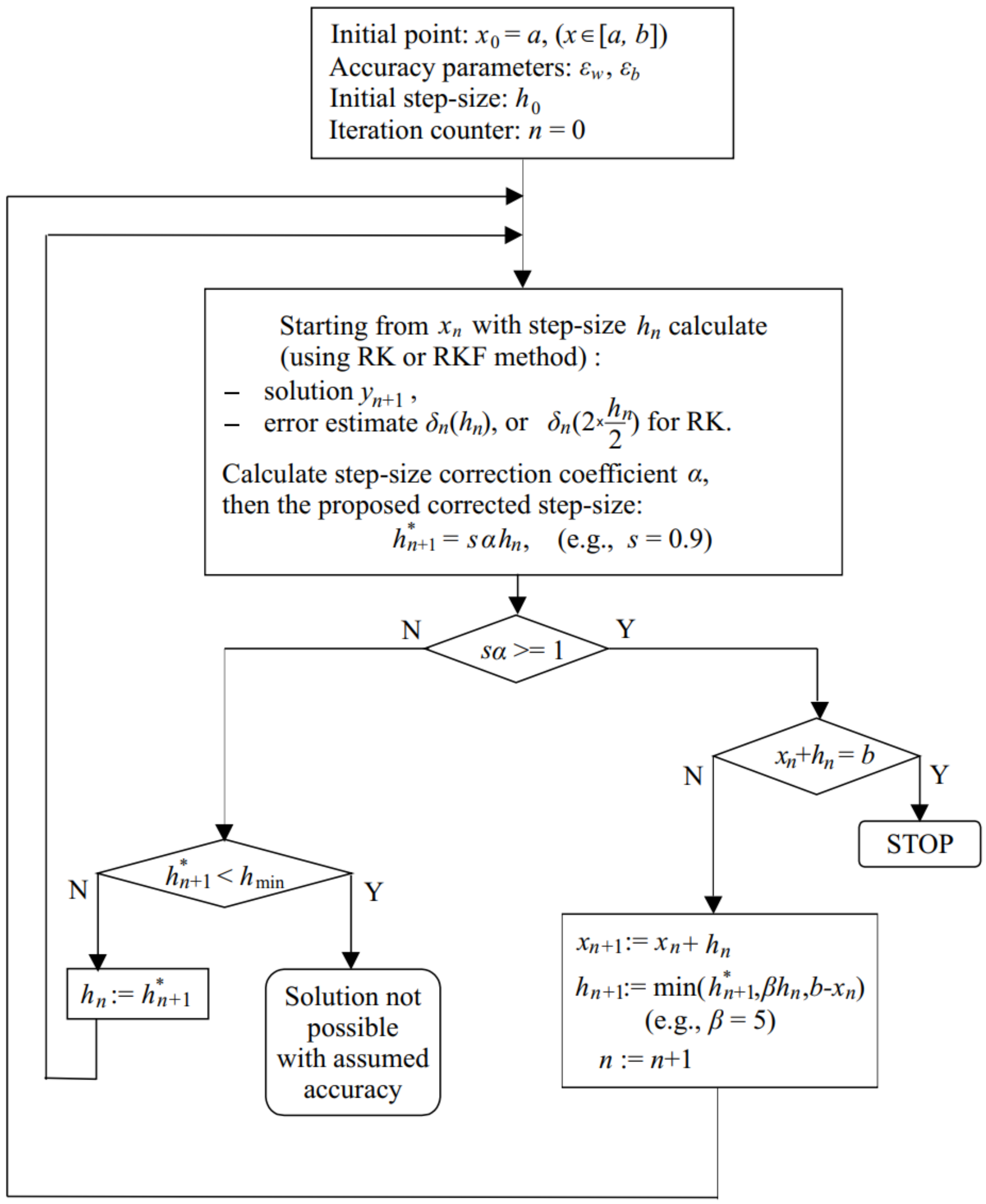
(P – prediction)

(E – evaluation)

(C – correction)

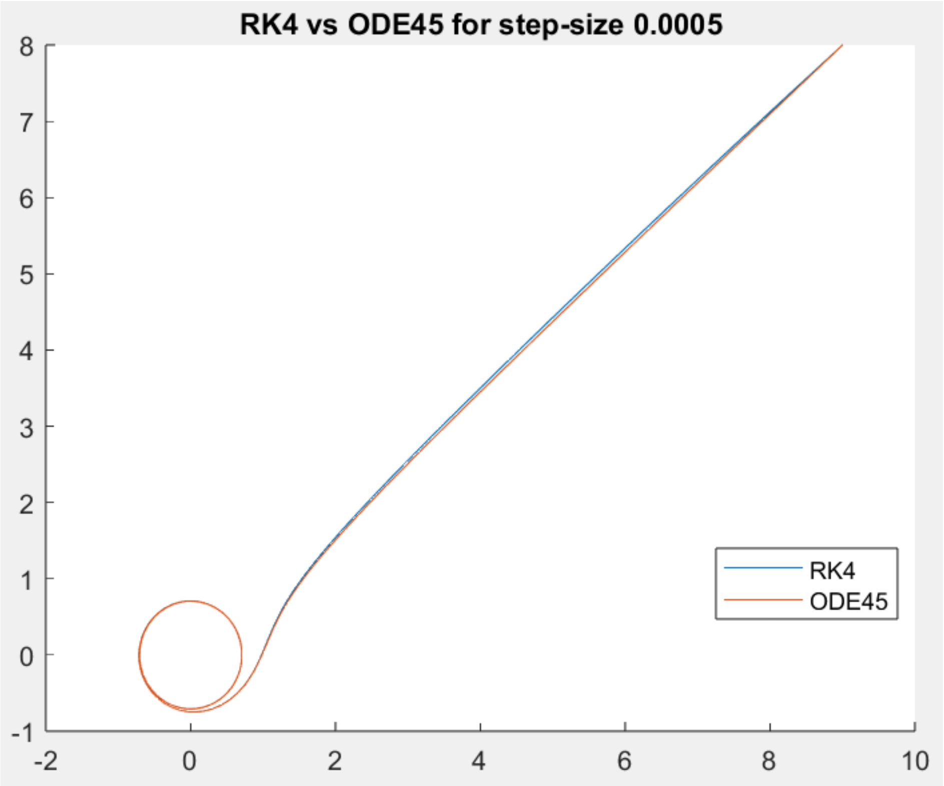
(E – evaluation)

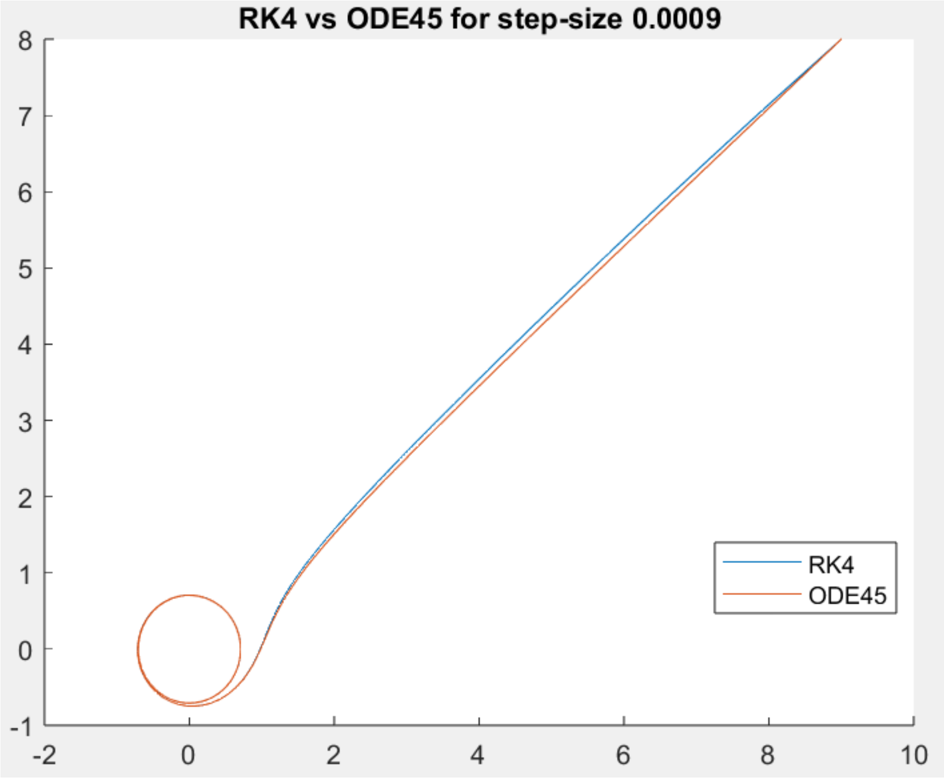
For Part B, the flow chart from book is:

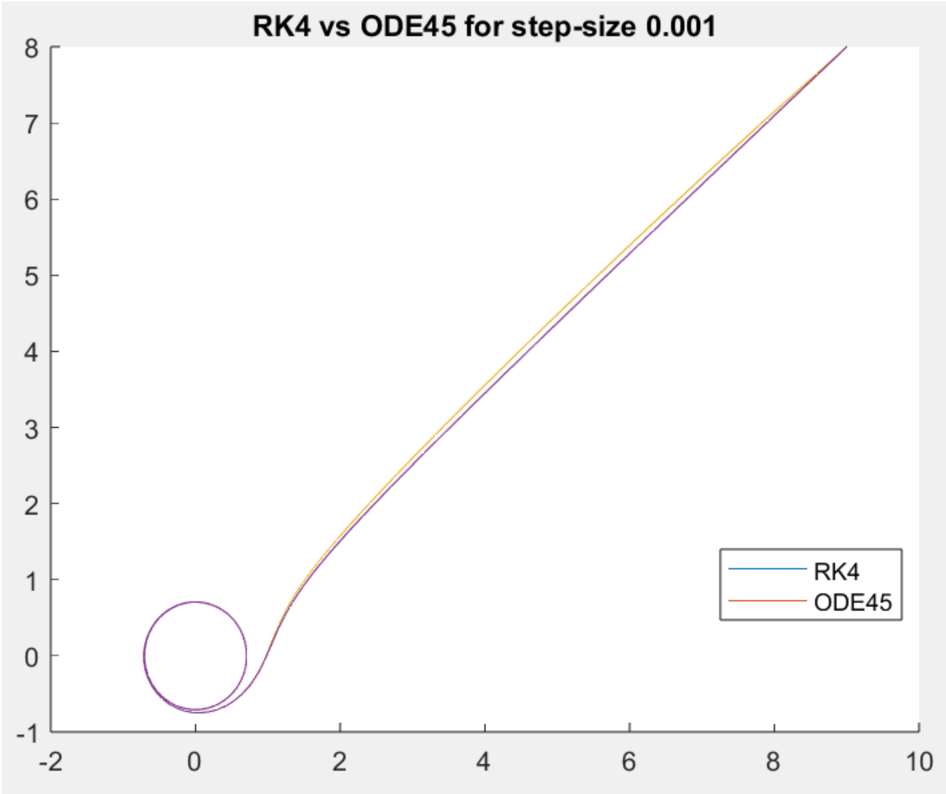


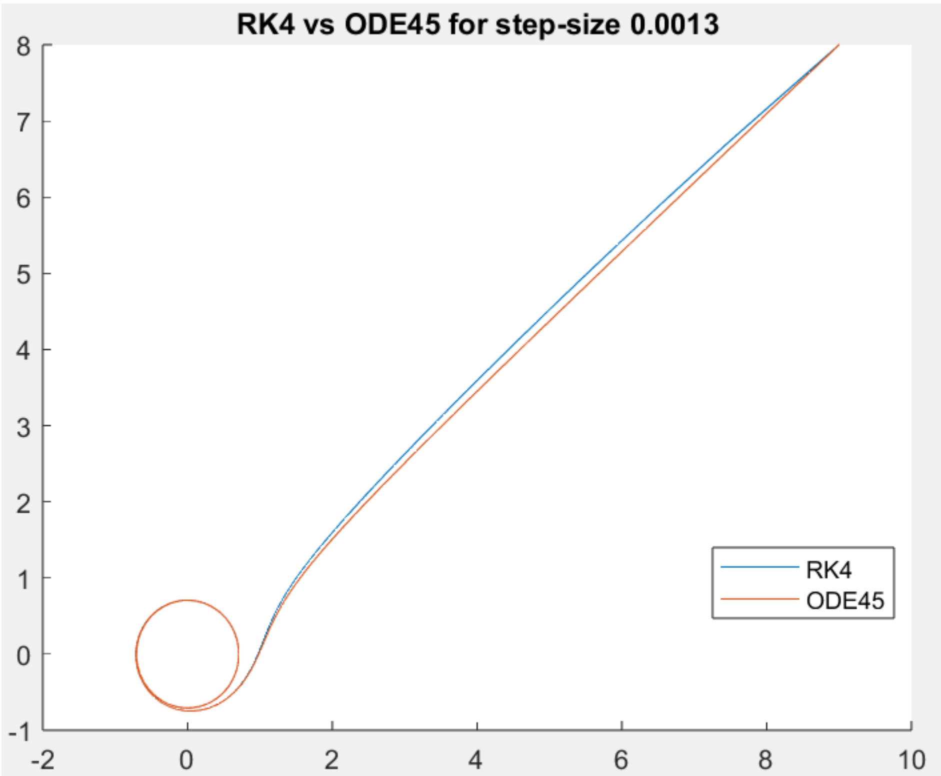
Result:

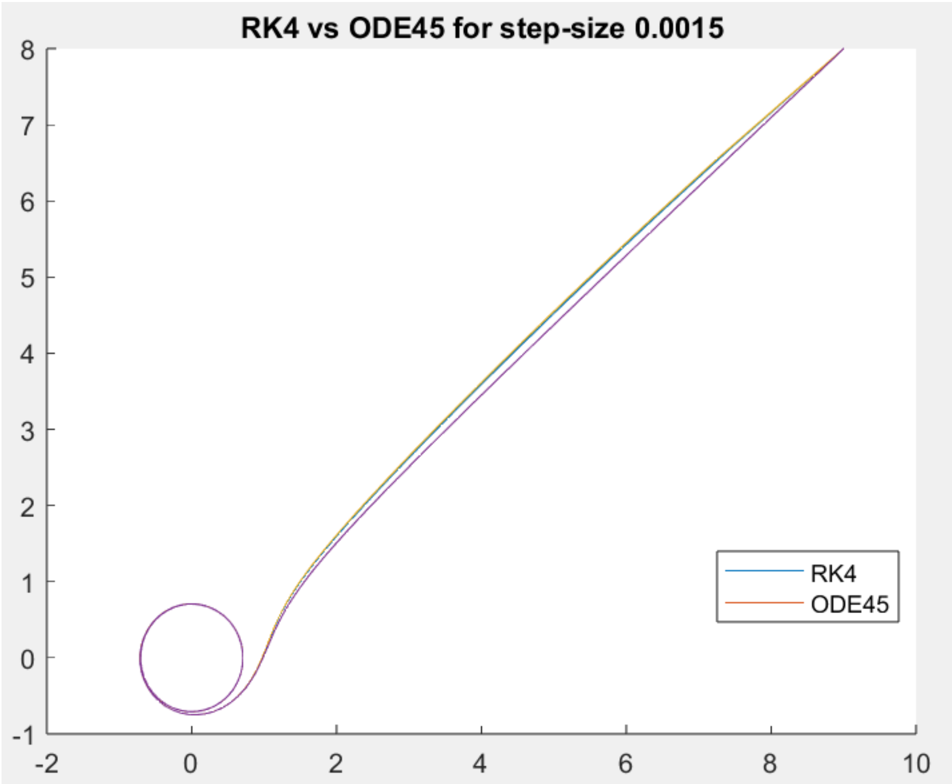
PART A: RK4 with different step-sizes:

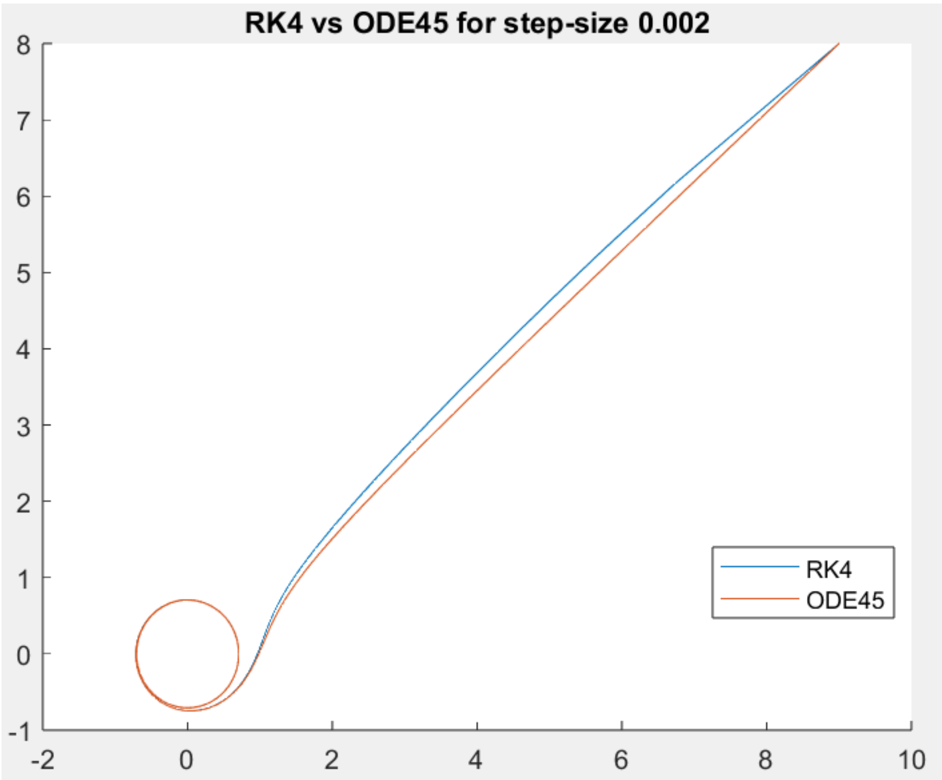


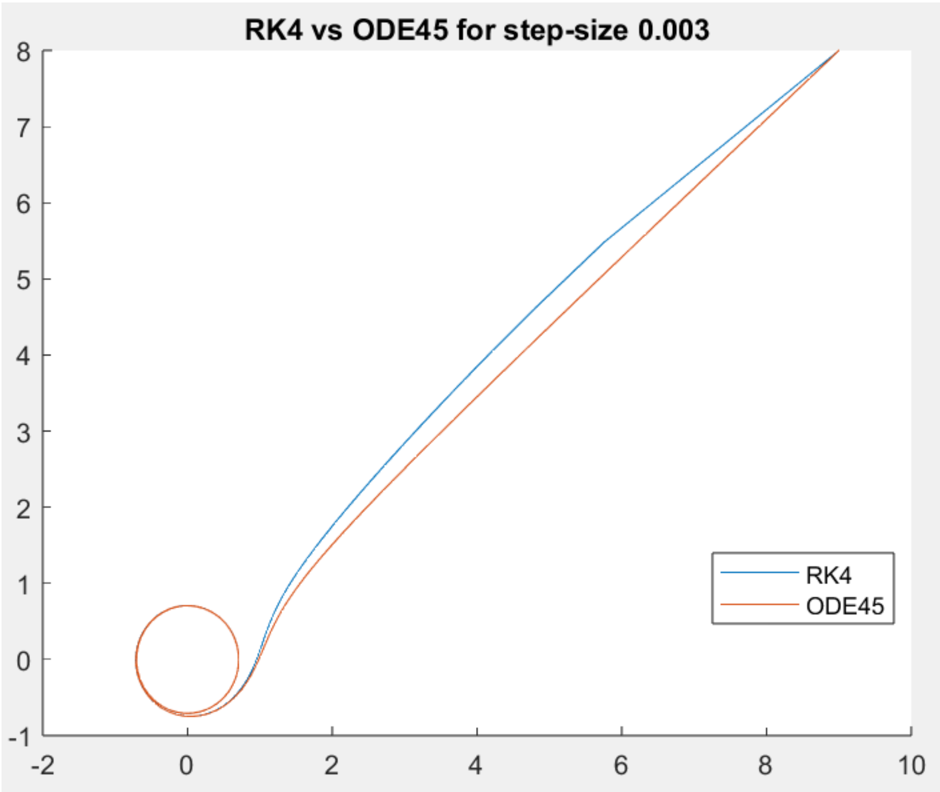


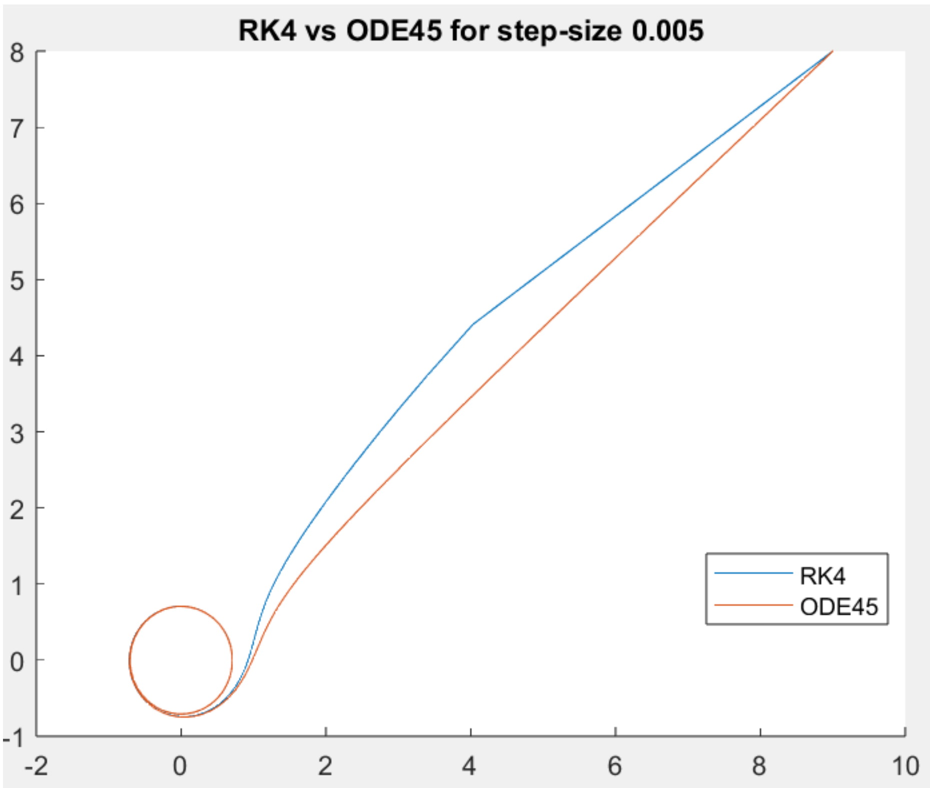

RK4 vs ODE45 for step-size 0.001
8
7
6
5
4
3
2
1
0
-1
-2
0
2
4
6
8
10
RK4
ODE45

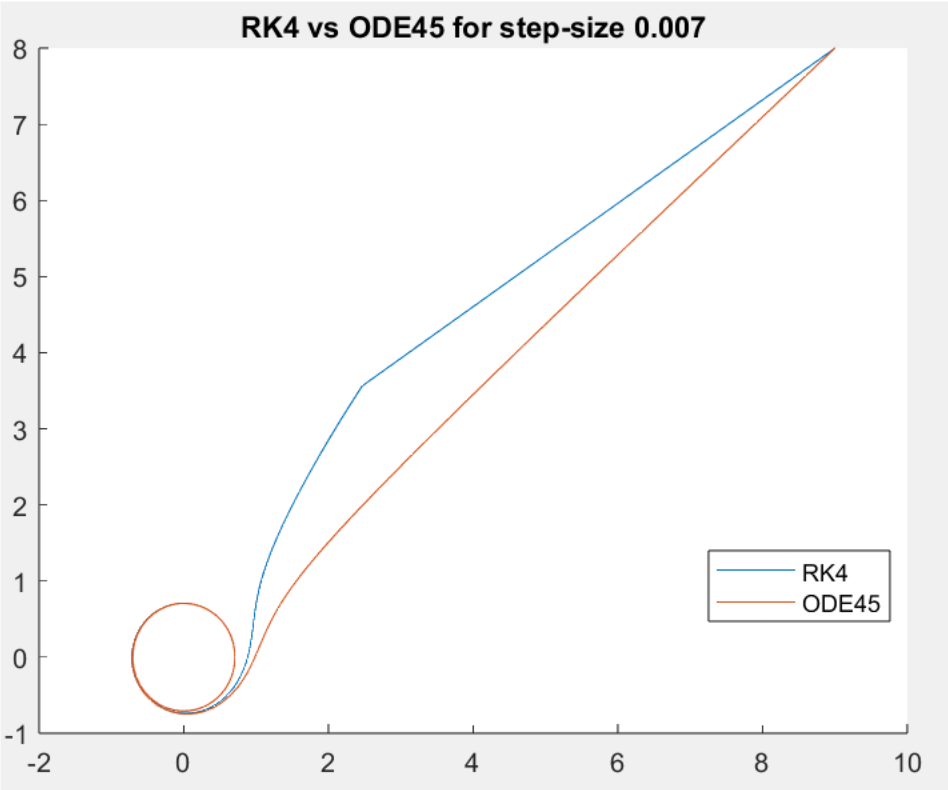

RK4 vs ODE45 for step-size 0.0013
8
7
6
5
4
3
2
1
0
-1
-2
0
2
4
6
8
10
RK4
ODE45

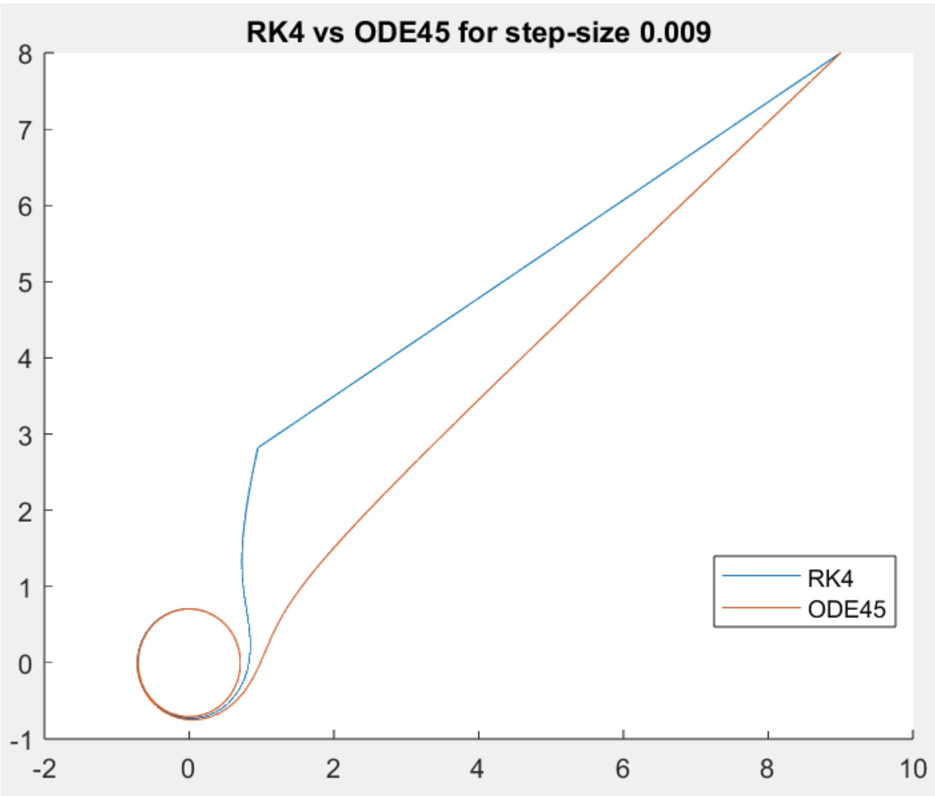

RK4 vs ODE45 for step-size 0.0015
RK4
ODE45
8
7
6
5
4
3
2
1
0
-1
-2
0
2
4
6
8
10

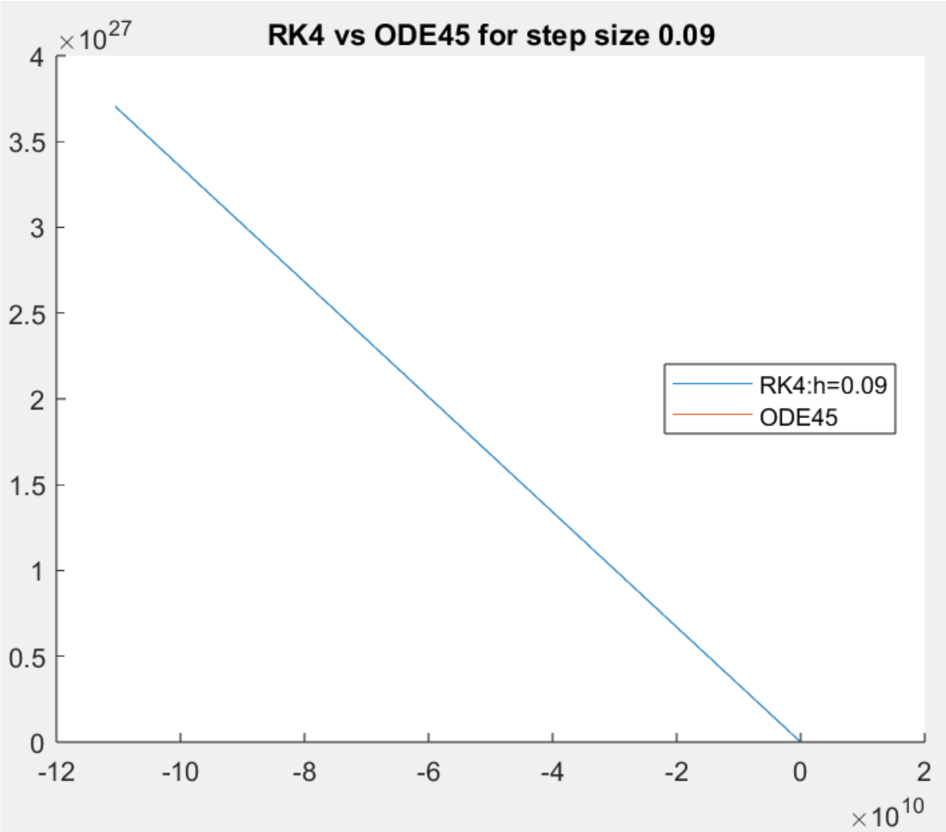

RK4 vs ODE45 for step-size 0.002
RK4
ODE45
8
7
6
5
4
3
2
1
0
-1
-2
0
2
4
6
8
10


RK4 vs ODE45 for step-size 0.003
RK4
ODE45
8
7
6
5
4
3
2
1
0
-1
-2
0
2
4
6
8
10


RK4 vs ODE45 for step-size 0.005
RK4
ODE45
8
7
6
5
4
3
2
1
0
-1
-2
0
2
4
6
8
10

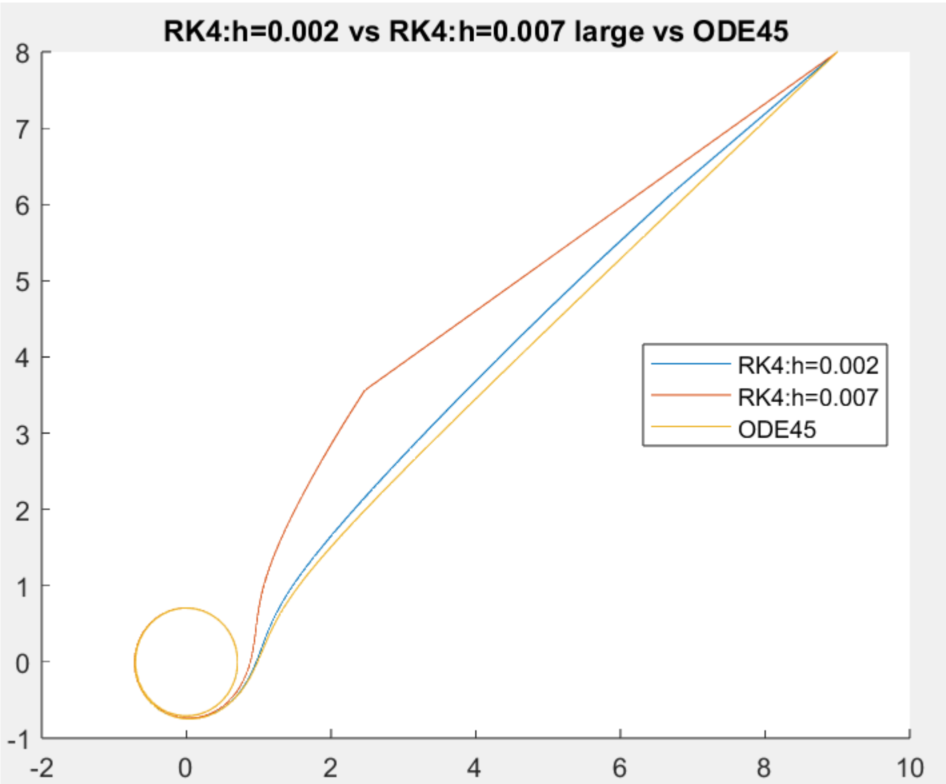

RK4 vs ODE45 for step-size 0.007
8
7
6
5
4
3
2
1
0
-1
-2
0
2
4
6
8
10
RK4
ODE45


RK4 vs ODE45 for step-size 0.009
8
7
6
5
4
3
2
1
0
-1
-2
0
2
4
6
8
10
RK4
ODE45

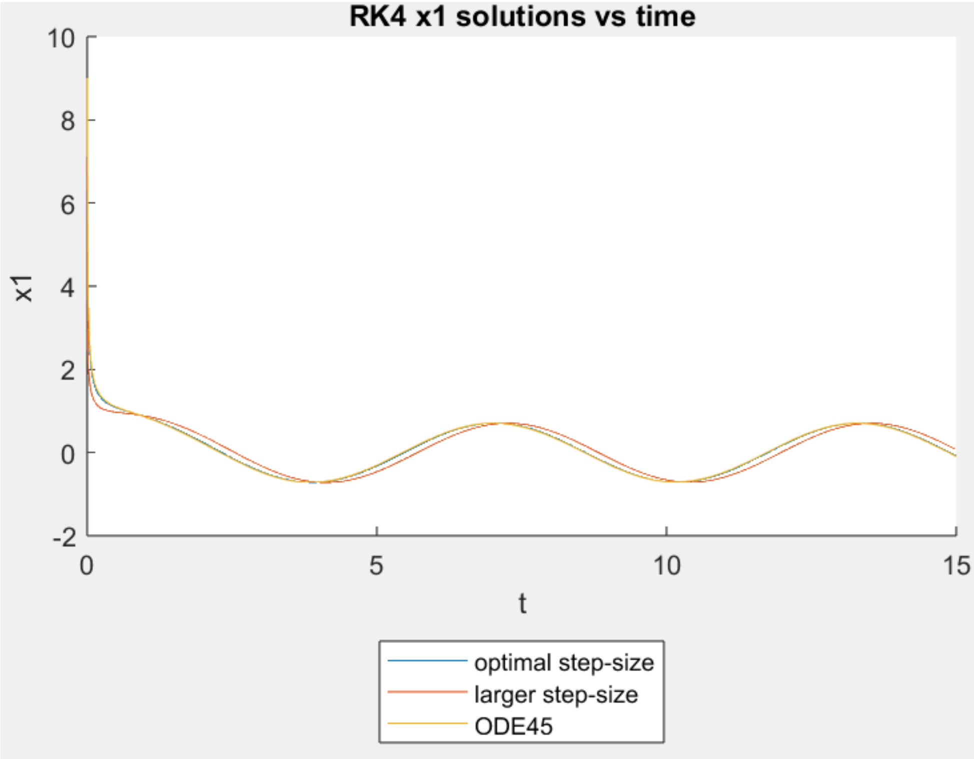


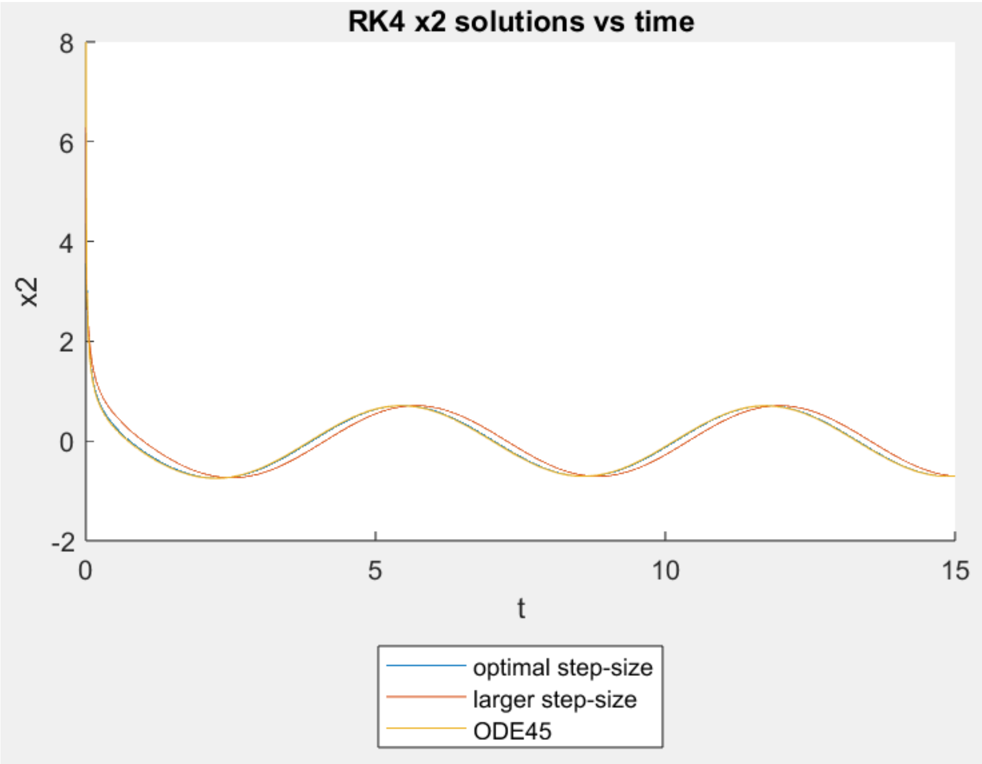
We found the optimal h = 0.002.

RK4 with optimal step-size, too large step-size and ODE45 in one plot:

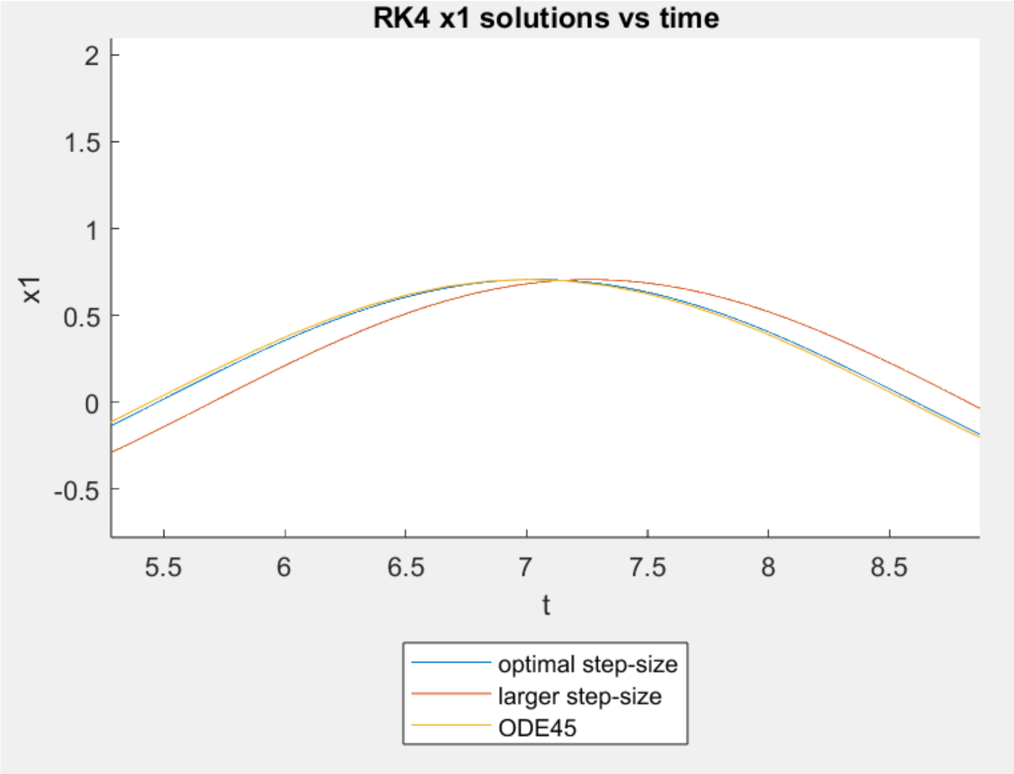


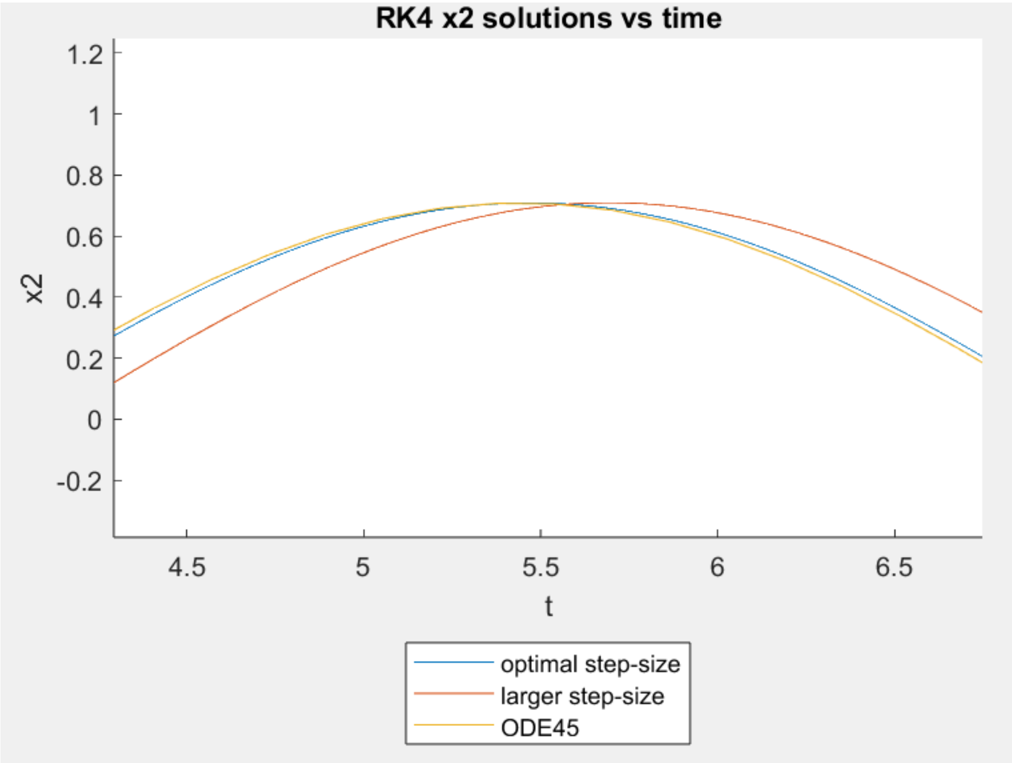
Solution vs time:



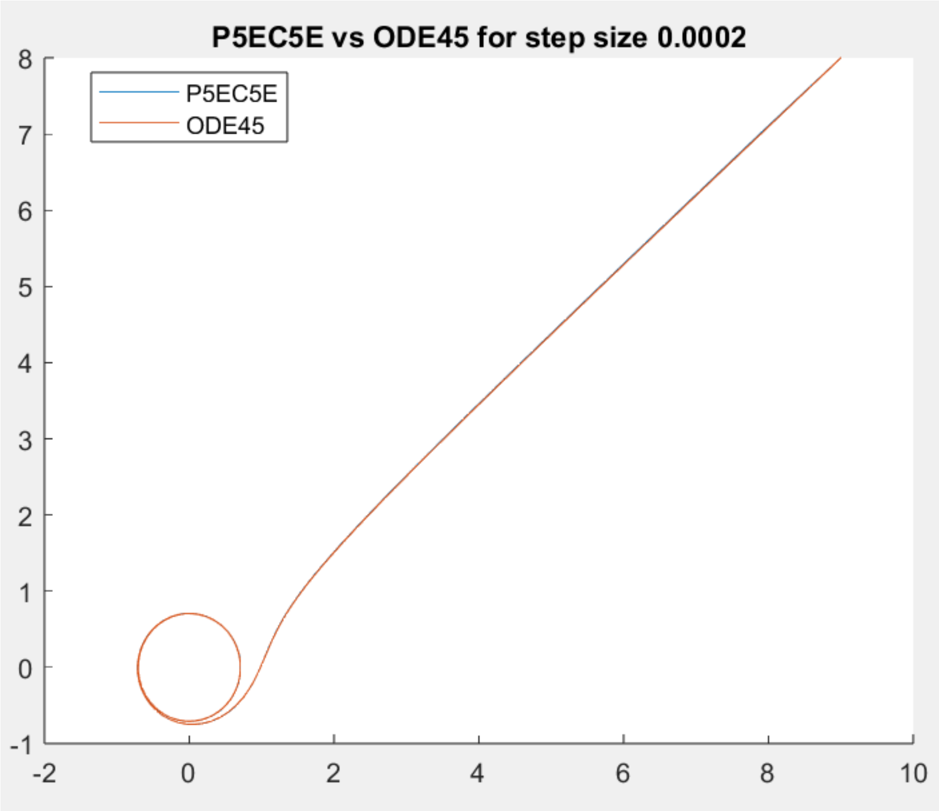


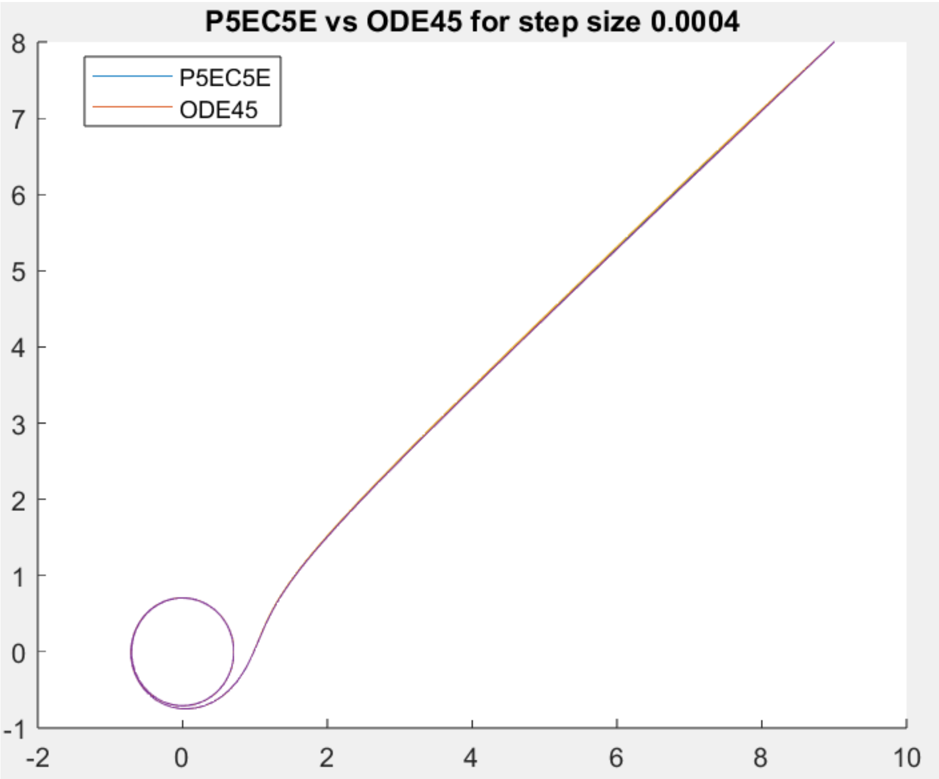
Solution vs time, zoomed in:

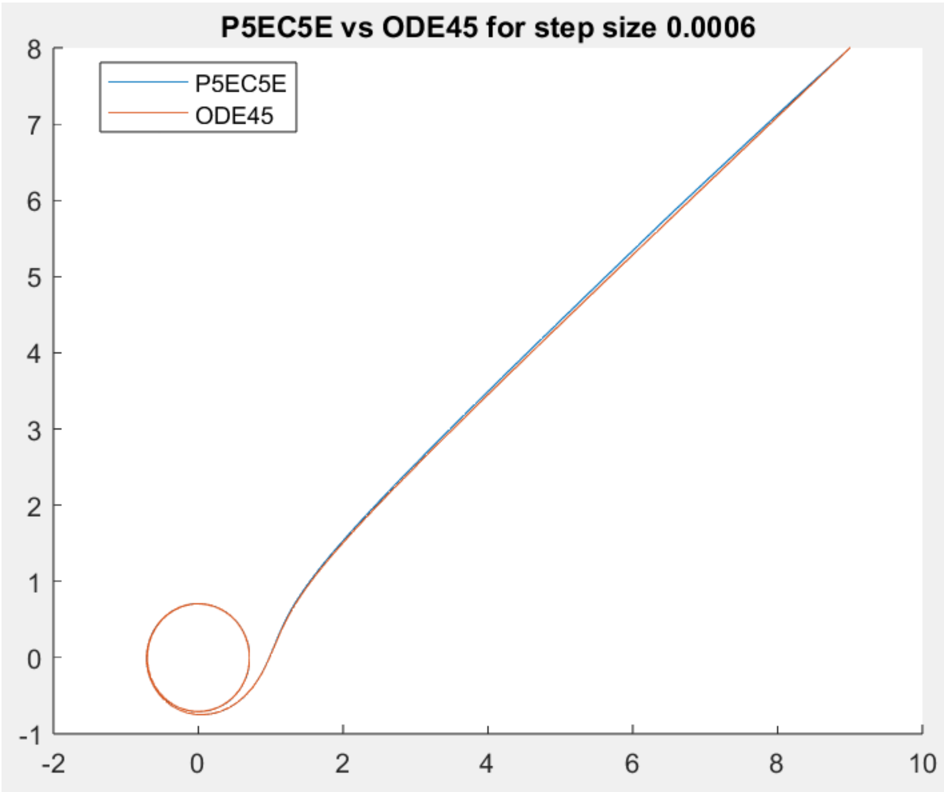


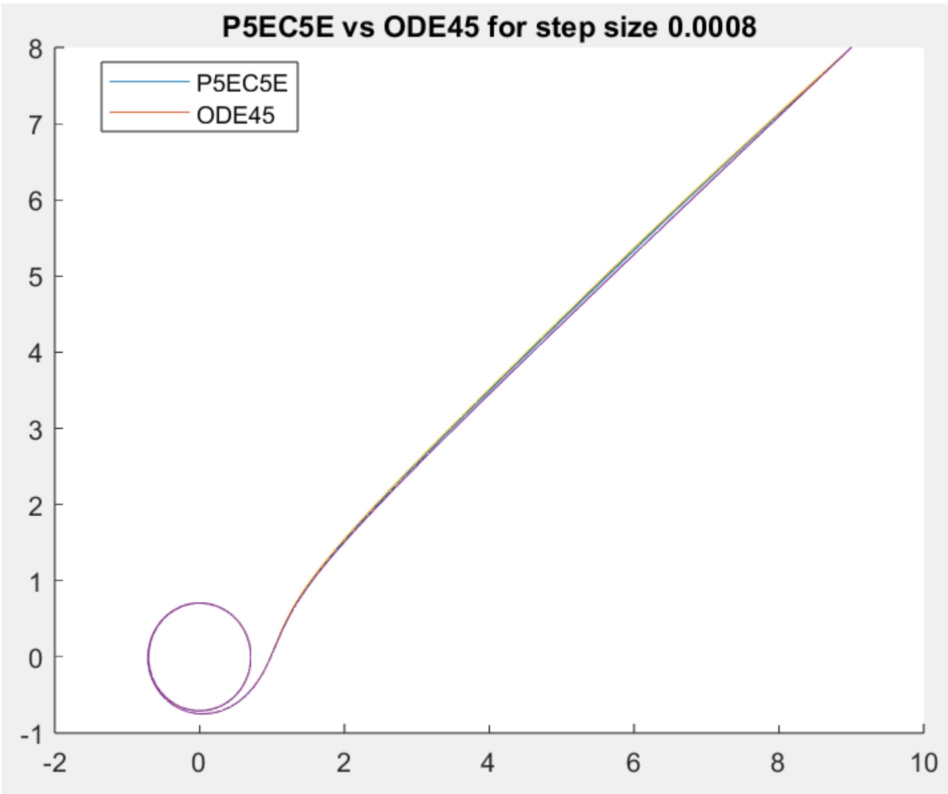


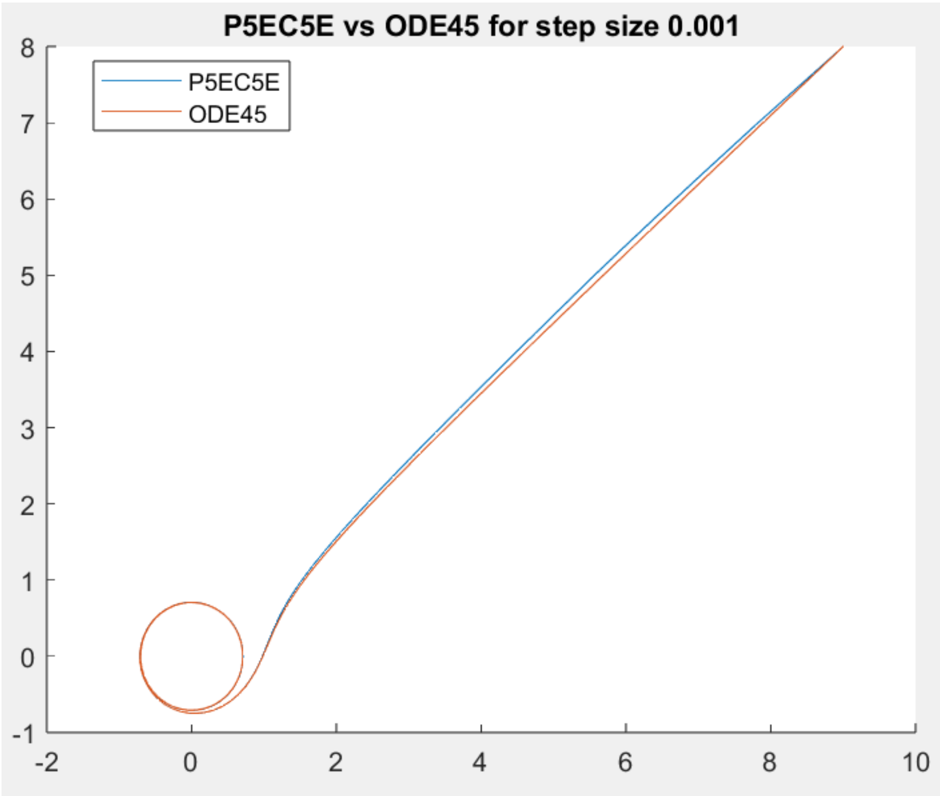
PART A: Adams PC(P5EC5E) with different step-sizes:

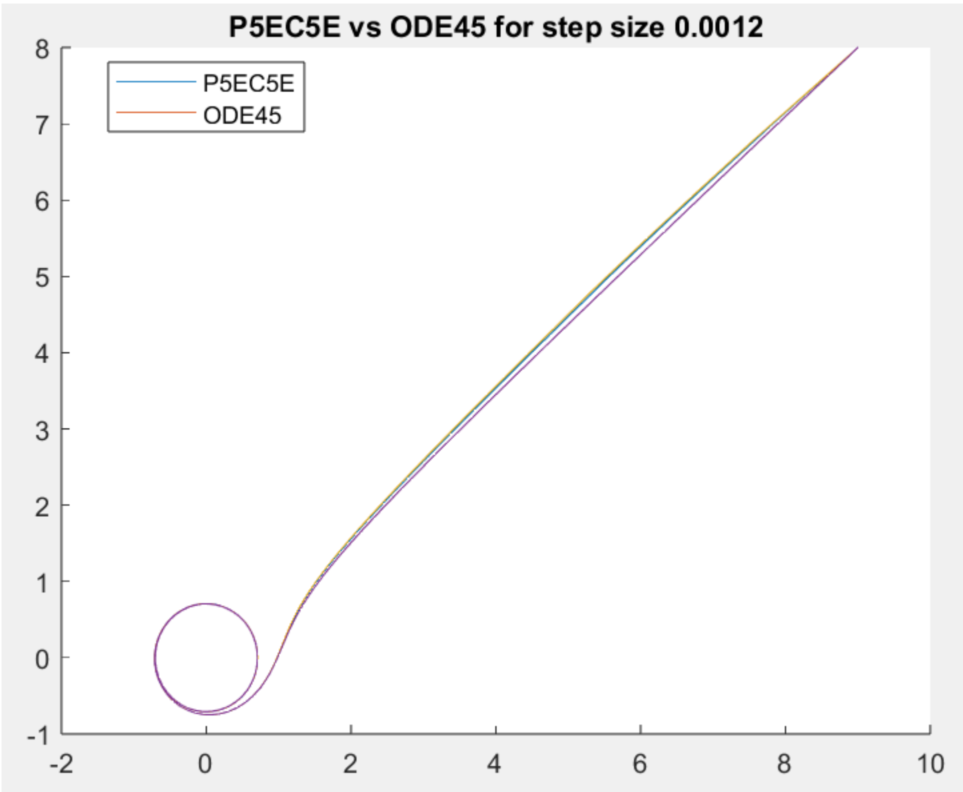


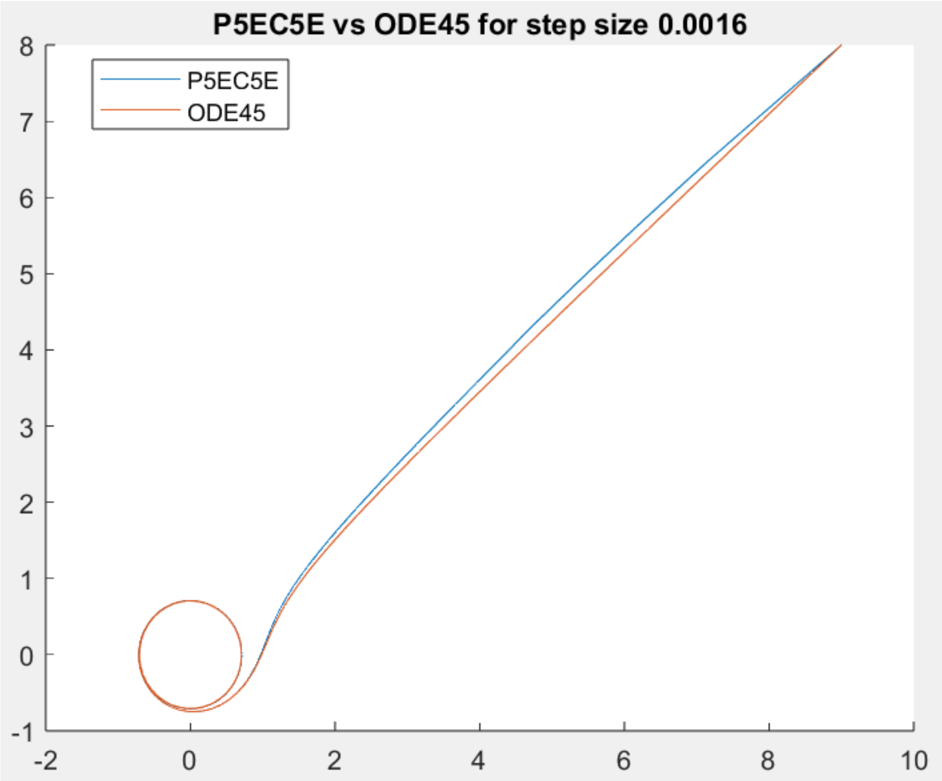


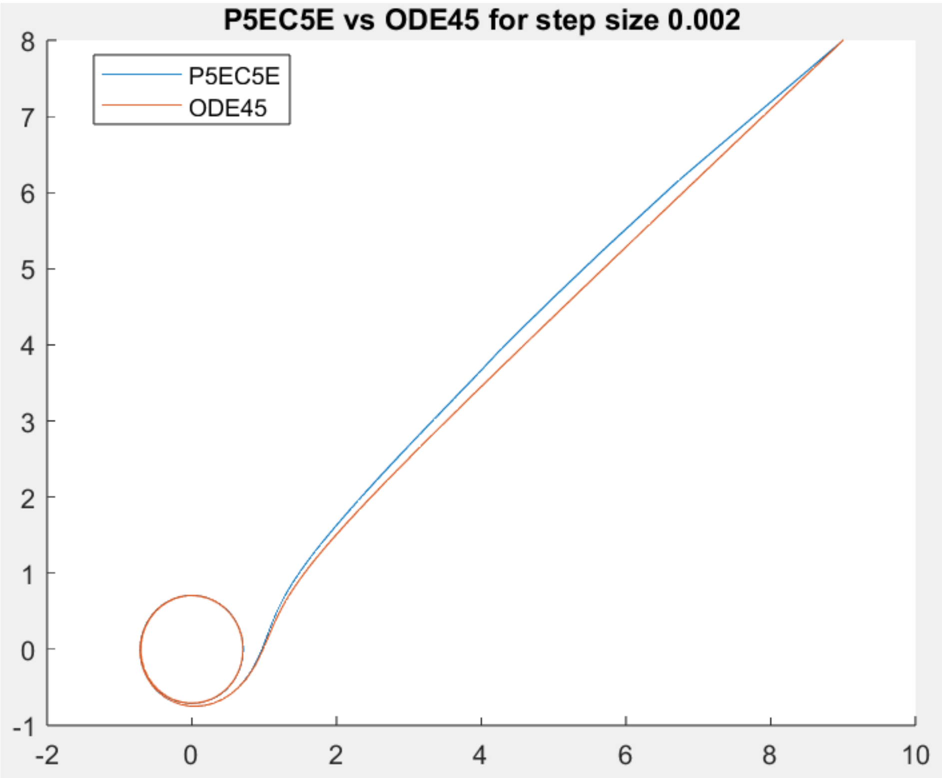

P5EC5E vs ODE45 for step size 0.0006
P5EC5E
ODE45
8
7
6
5
4
3
2
1
0
-1
-2
0
2
4
6
8
10

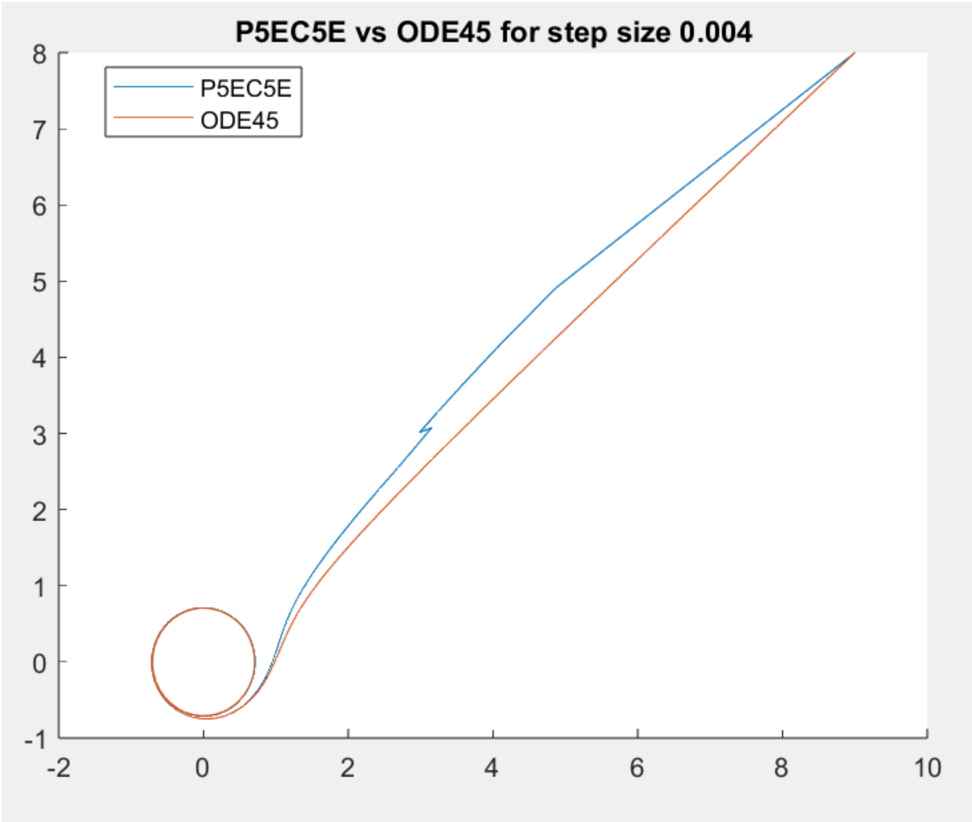

P5EC5E vs ODE45 for step size 0.0008
P5EC5E
ODE45
8
7
6
5
4
3
2
1
0
-1
-2
0
2
4
6
8
10

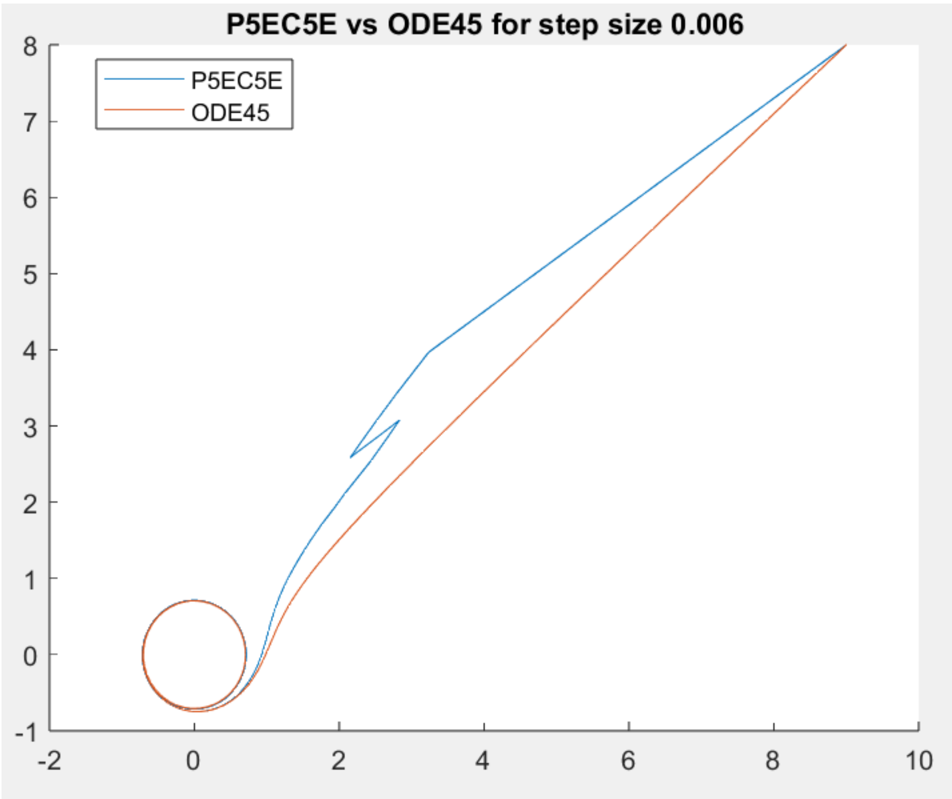

P5EC5E vs ODE45 for step size 0.001
P5EC5E
ODE45
8
7
6
5
4
3
2
1
0
-1
-2
0
2
4
6
8
10

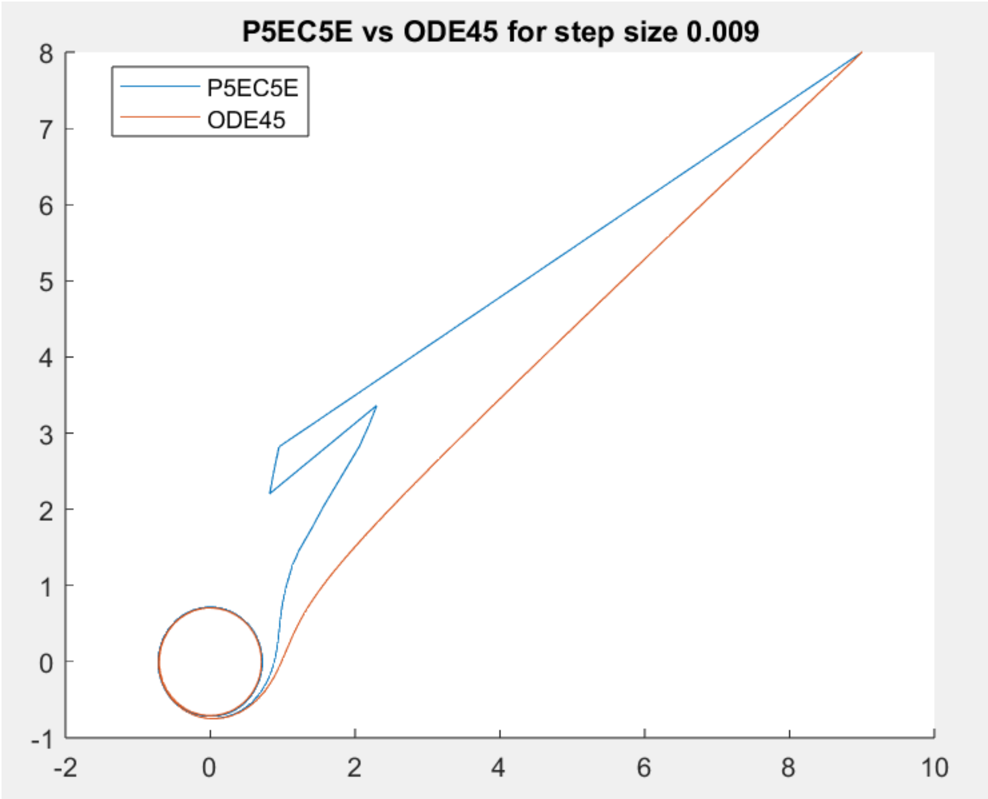

P5EC5E vs ODE45 for step size 0.0012
P5EC5E
ODE45
8
7
6
5
4
3
2
1
0
-1
-2
0
2
4
6
8
10

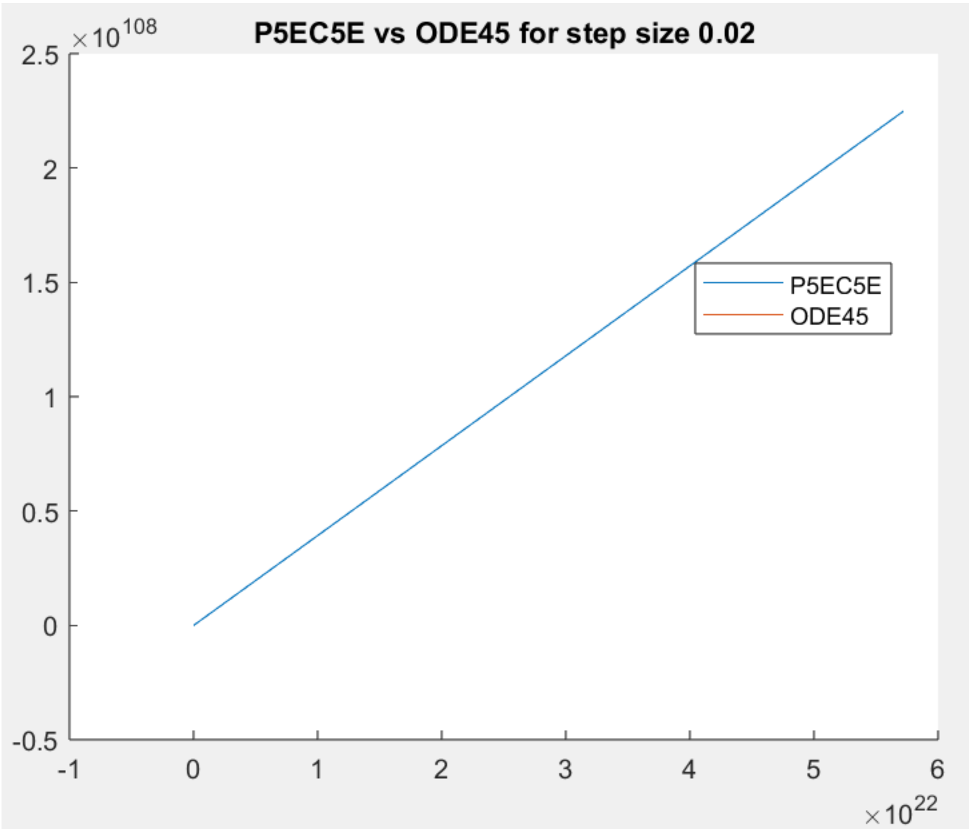

P5EC5E vs ODE45 for step size 0.0016
P5EC5E
ODE45
8
7
6
5
4
3
2
1
0
-1
-2
0
2
4
6
8
10


P5EC5E vs ODE45 for step size 0.002
P5EC5E
ODE45
8
7
6
5
4
3
2
1
0
-1
-2
0
2
4
6
8
10


P5EC5E vs ODE45 for step size 0.004
P5EC5E
ODE45
8
7
6
5
4
3
2
1
0
-1
-2
0
2
4
6
8
10

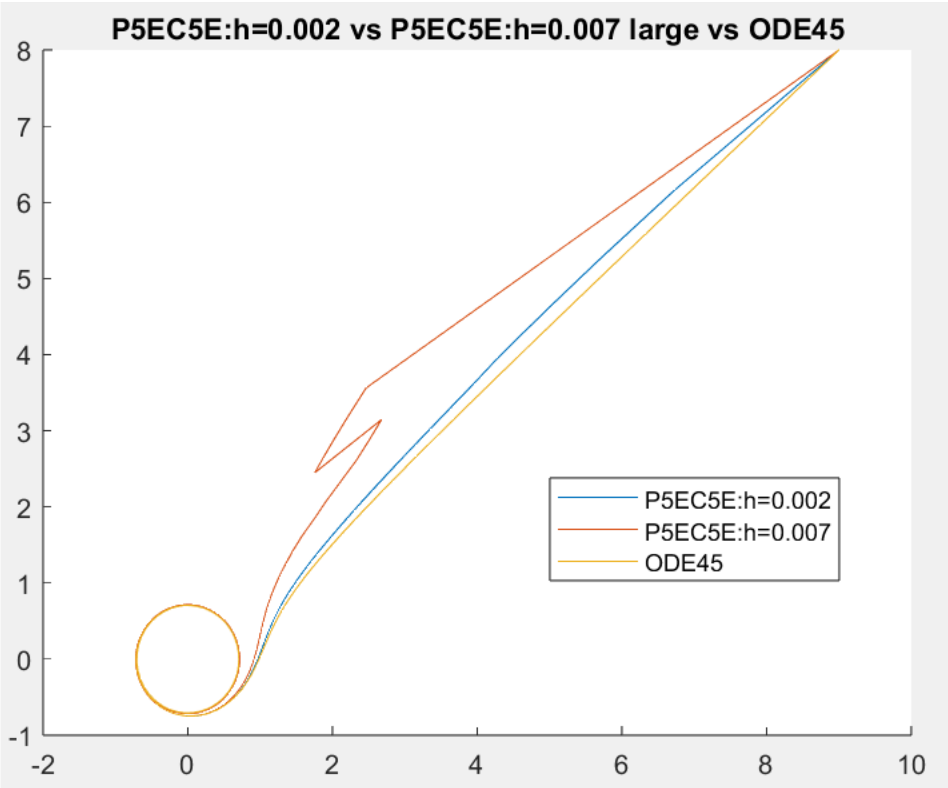

P5EC5E vs ODE45 for step size 0.006
P5EC5E
ODE45
8
7
6
5
4
3
2
1
0
-1
-2
0
2
4
6
8
10


P5EC5E vs ODE45 for step size 0.009
P5EC5E
ODE45
8
7
6
5
4
3
2
1
0
-1
-2
0
2
4
6
8
10

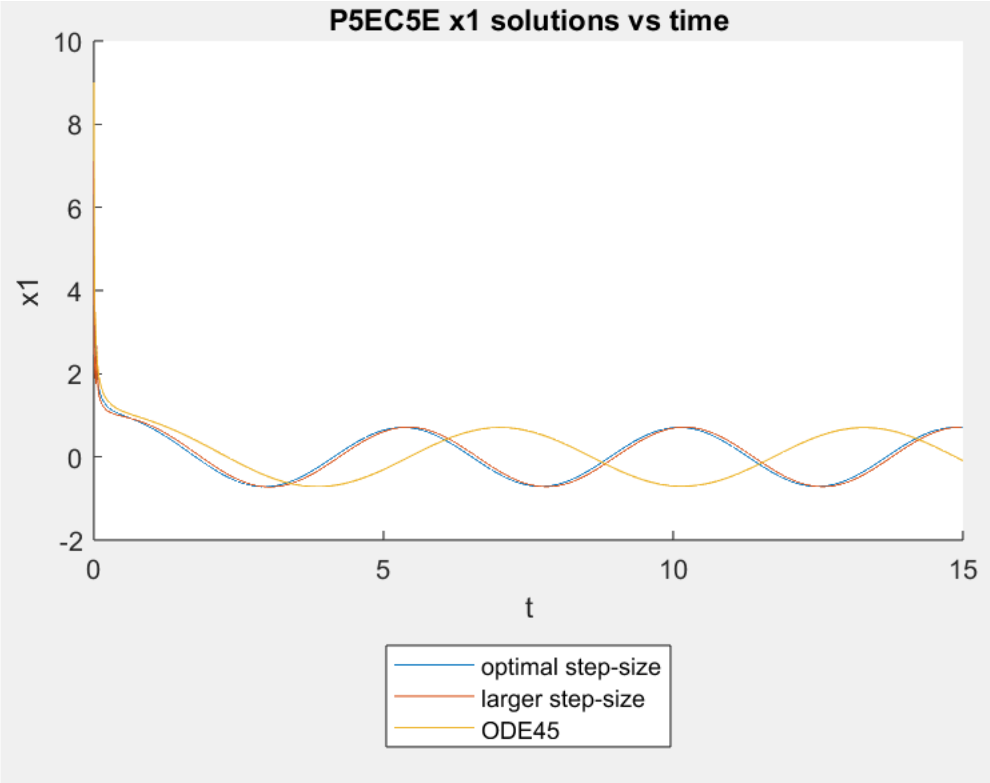

P5EC5E vs ODE45 for step size 0.02
$\times 10^{108}$
2.5
2
1.5
1
0.5
0
-0.5
P5EC5E
ODE45
-1
0
1
2
3
4
5
6
$\times 10^{22}$

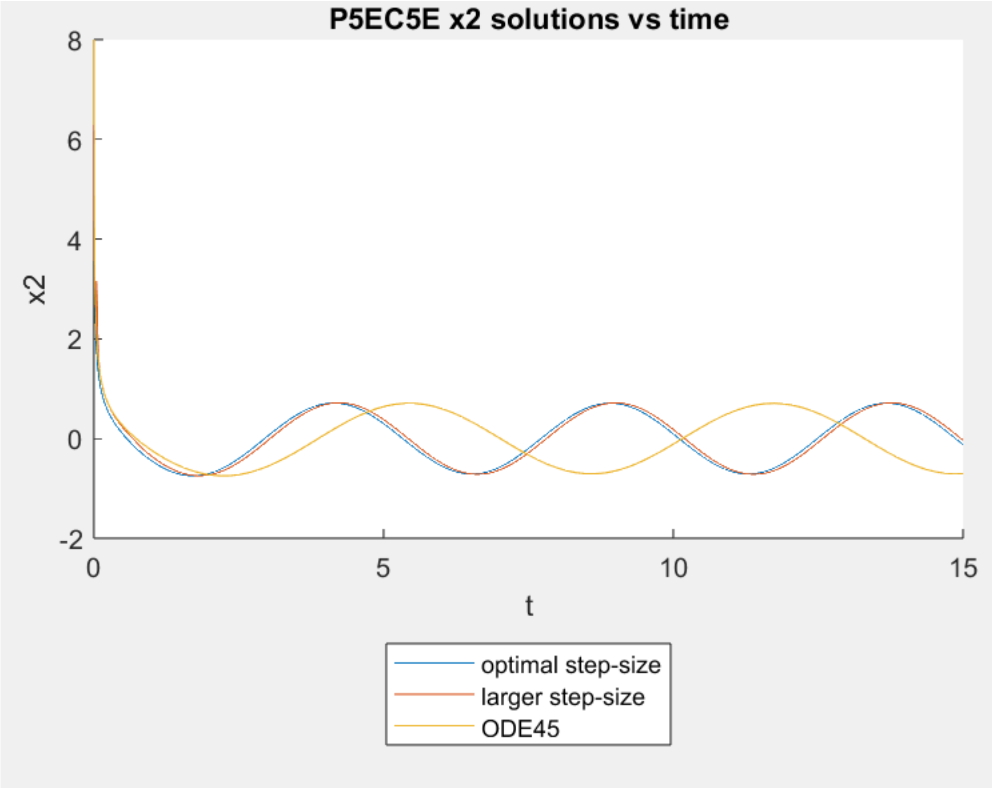
We found the optimal h = 0.002.

P5EC5E with optimal step-size, too large step-size and ODE45 in one plot:



Solution vs time:

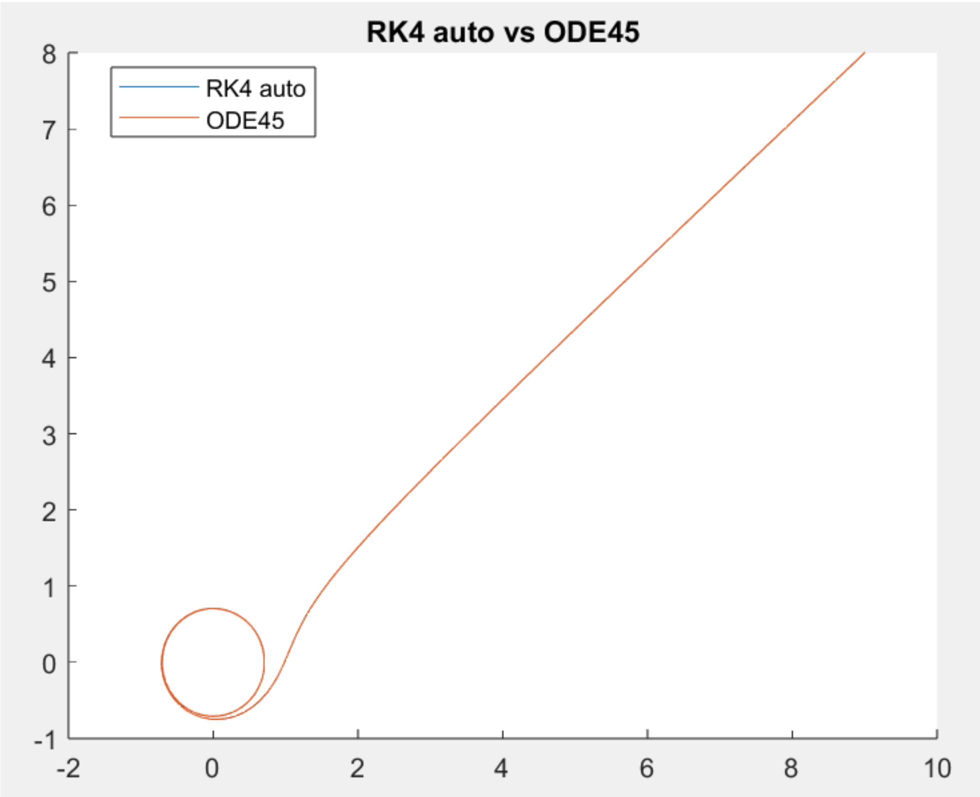




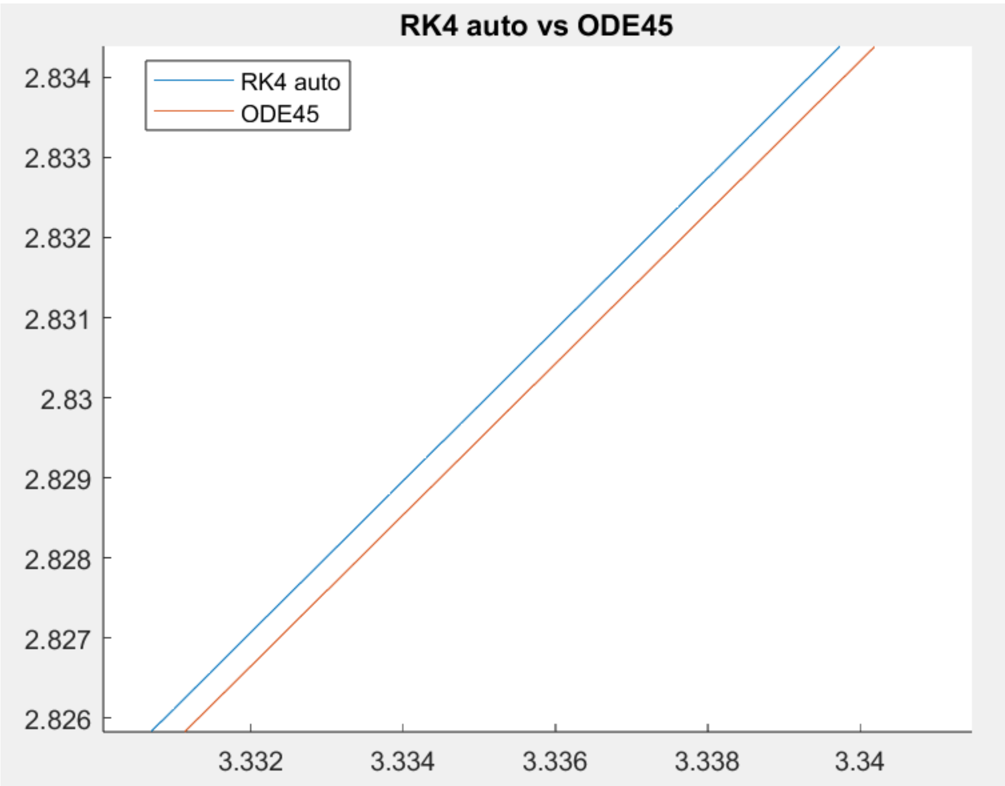
PART A: Conclusion:

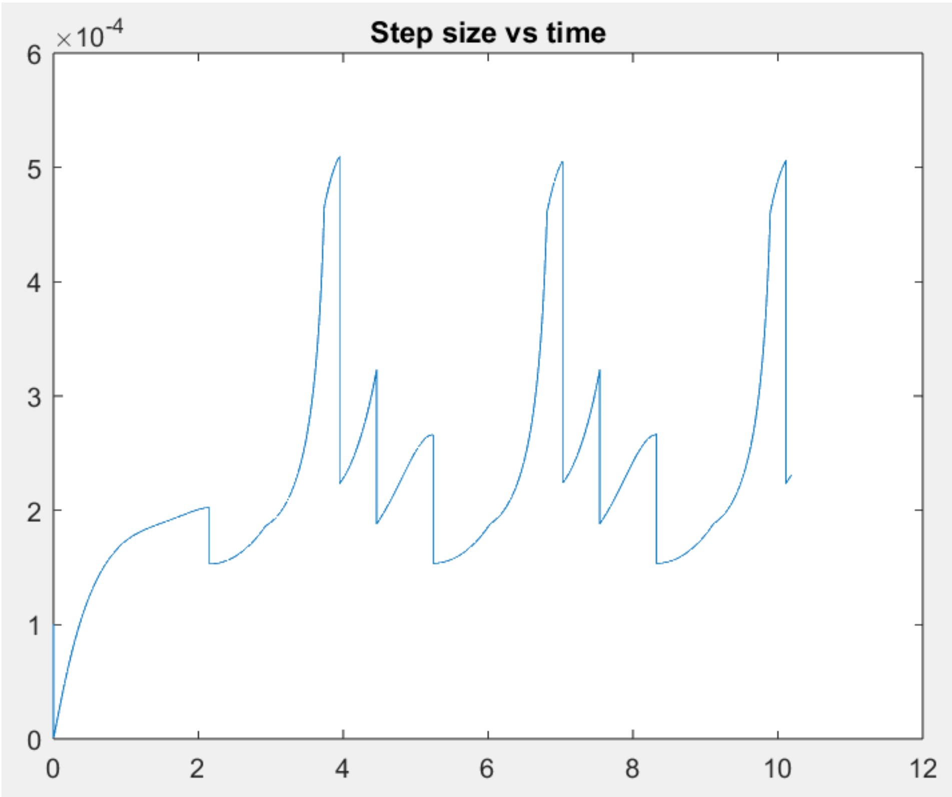
It's not easy to find the step size, I can only try from very small numbers. But generally the results are satisfied, the difference between ODE45 and my own method is very small.

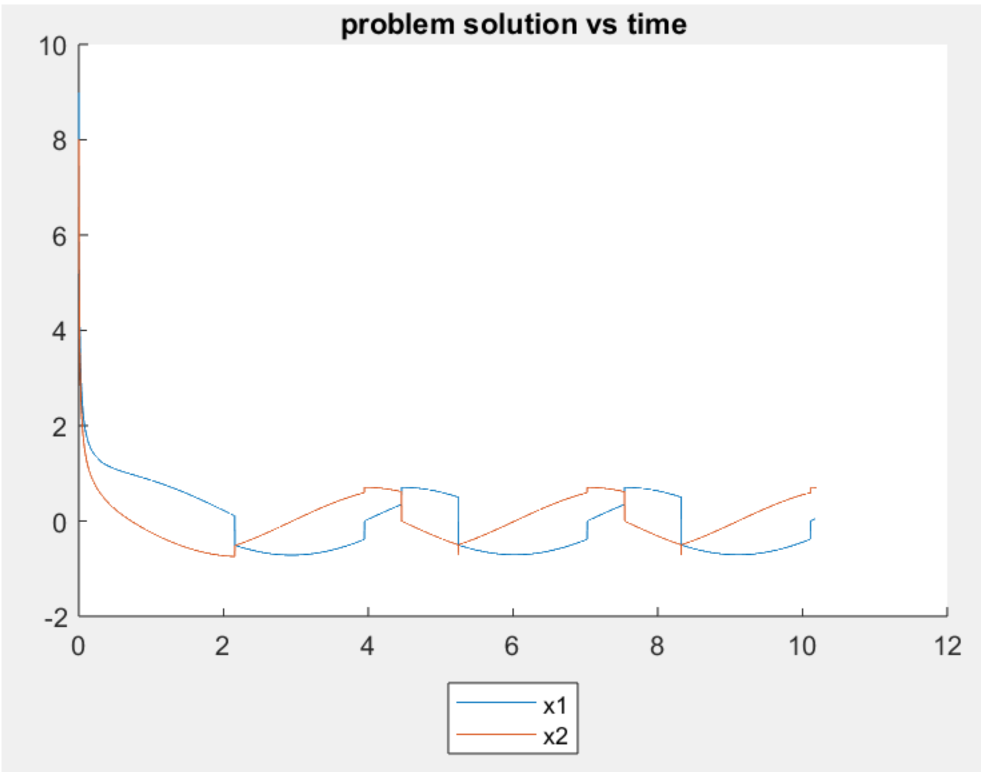
PART B:



Zoomed in:




Step size vs time
×10^-4
6
5
4
3
2
1
0
0
2
4
6
8
10
12


problem solution vs time
10
8
6
4
2
0
-2
0
2
4
6
8
10
12
x1
x2

PART B: Conclusion:

The beginning of the step-size is 0.0001, later we found the best step-size 0.005. We can see from the plot that the result is quite accurate. It’s a great idea to use automatic RK4, so we don’t to decide the step-size by using our eyes to see the difference between ODE45 and RK4.

Reference:

1. *Numerical Methods* by Piotr Tatjewski

Code:

Task 1:

Main:

```
x = [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5];
y = [-77.9639, -39.5900, -17.5814, -5.1530, 0.7608, 2.0270, 1.2585, -
0.5477, -2.2384, -5.1580, -8.1875];

least_square(x, y, 1);
least_square(x, y, 2);
least_square(x, y, 3);
least_square(x, y, 4);
least_square(x, y, 5);
least_square(x, y, 6);
least_square(x, y, 7);
least_square(x, y, 8);

result(x, y);
```

Gram’s matrix

```
function [A,b] = grams_matrix(x, y, degree)

    degree = degree + 1;

    % initialize matrices
    A = zeros(degree, degree);
    b = zeros(degree, 1);

    n = length(x);

    % matrix A
    for i=1:degree
        for j=1:degree
            for k=1:n
                A(i,j) = A(i, j) + (x(1, k)^(i+j-1-1));
            end
        end
    end

    % vector b
    for i=1:degree
        for k=1:n
            b(i, 1) = b(i, 1) + (y(k)*(x(k)^(i-1)));
        end
    end
end
```

Least square:

```
function least_square(x, y, degree)

    % Gram's matrix
    [A,b] = grams_matrix(x, y, degree);

    % solve it
    [~,~,C] = solution(A, b);

    %disp(C);
    %D = A\b;
    %disp(D);
```

```
    % plot
    figure(degree);
    plot(x,y, 'r*');
    hold on;

    % functions based on degrees
    if(degree == 1)
        f = @(x)C(2)*x + C(1);
    end

    if(degree == 2)
        f = @(x)C(3)*x^2 + C(2)*x + C(1);
    end

    if(degree == 3)
        f = @(x)C(4)*x^3 + C(3)*x^2 + C(2)*x + C(1);
    end

    if(degree == 4)
        f = @(x)C(5)*x^4 + C(4)*x^3 + C(3)*x^2 + C(2)*x + C(1);
    end

    if(degree == 5)
        f = @(x)C(6)*x^5 + C(5)*x^4 + C(4)*x^3 + C(3)*x^2 + C(2)*x + C(1);
    end

    if(degree == 6)
        f = @(x)C(7)*x^6 + C(6)*x^5 + C(5)*x^4 + C(4)*x^3 + C(3)*x^2 +
C(2)*x + C(1);
    end

    if(degree == 7)
        f = @(x)C(8)*x^7 + C(7)*x^6 + C(6)*x^5 + C(5)*x^4 + C(4)*x^3 +
C(3)*x^2 + C(2)*x + C(1);
    end

    if(degree == 8)
        f = @(x)C(9)*x^8 + C(8)*x^7 + C(7)*x^6 + C(6)*x^5 + C(5)*x^4 +
C(4)*x^3 + C(3)*x^2 + C(2)*x + C(1);
    end

    % plotting function
    fplot(f,[-5 5],'k');
    grid on;
```

```
    hold off;
    title("Degree: " + degree);
    % put legend in less conflict area
    legend("Samples","Function",'Location','Best');

end
```

QR factorization:

```
function [Q, R] = QR(A)

    [m, n] = size(A);

    Q = zeros(m,n);
    R = zeros(n,n);
    d = zeros(1,n);

    % factorization
    for i=1:n
        Q(:,i) = A(:,i);
        R(i,i) = 1;
        d(i) = Q(:,i)'*Q(:,i);

        for j=i+1:n
            R(i,j) = (Q(:,i)'*A(:,j))/d(i);
            A(:,j) = A(:,j)-R(i,j)*Q(:,i);
        end
    end

    % normalization
    for i=1:n
        dd = norm(Q(:,i));
        Q(:,i) = Q(:,i)/dd;
        R(i,i:n) = R(i,i:n)*dd;
    end
end
```

Result:

```
function result(x, y)

    % from degree 1 to degree 8
    for degree = 1:8
```

```
        [A,b] = grams_matrix(x, y, degree);
        %disp(A);
        %disp(b);
        % solve
        [error, cond_num, ~] = solution(A,b);

        disp("degree "+ degree + " error " + error + " cond_num " +
cond_num);
    end

end
```

Solution:

```
function [error, cond_num, C] = solution(A, b)

    [Q, R] = QR(A); % QR factorization

    n = size(A,1);
    C = zeros(n,1);

    % connect matrix with vector
    D = Q' * b;

    % back substitution
    for i=n:-1:1
        C(i) = D(i)/R(i,i);
        D(1:i-1) = D(1:i-1) - (C(i)*R(1:i-1,i));
    end

    % residuum
    residuum = A*C - b;

    % norm of residuum
    error = norm(residuum);

    % condition number
    cond_num = cond(A);
end
```

Task 2:

Task 2 part A:

```
% Part A
% RK4
[x1, x2] = RK4(0.09, 0, 15);
% [x3, x4] = RK4(0.007, 0, 15);
% ode45( functions, interval, initial_conditions )
[~,y_ode45] = ode45(@(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)], [0, 15], [9, 8]);
% t is a time vector, y is a vector solutions of a function

figure(1);
hold on;

plot(x1,x2);
% plot(x3,x4);
plot(y_ode45(:,1),y_ode45(:,2));
% legend("RK4:h=0.002", "RK4:h=0.007", "ODE45", 'Location', 'Best');
% title("RK4:h=0.002 vs RK4:h=0.007 large vs ODE45");
legend("RK4:h=0.09", "ODE45", 'Location', 'Best');
title("RK4 vs ODE45 for step size 0.09");
hold off;

%P5EC5E
[x1, x2] = P5EC5E(0.002, 0, 15);
[x3, x4] = P5EC5E(0.007, 0, 15);
% ode45( functions, interval, initial_conditions )
[~,y_ode45] = ode45(@(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)], [0, 15], [9, 8]);
% t is a time vector, y is a vector solutions of a function

figure(1);
hold on;

plot(x1,x2);
plot(x3,x4);
plot(y_ode45(:,1),y_ode45(:,2));
legend("P5EC5E:h=0.002", "P5EC5E:h=0.007", "ODE45", 'Location', 'Best');
title("P5EC5E:h=0.002 vs P5EC5E:h=0.007 large vs ODE45");
% legend("P5EC5E", "ODE45", 'Location', 'Best');
% title("P5EC5E vs ODE45 for step size 0.02");
hold off;
```

```
% Solution versus time %

%RK4
compare_time(1,1); % x1
compare_time(1,0); % x2

%P5EC5E
compare_time(0,1); % x1
compare_time(0,0); % x2
```

RK4:

```
function [y1,y2,x] = RK4(h, beginning, ending)

    x = beginning:h:ending;

    % prelocate with zeros
    y1 = zeros(1,length(x));
    y2 = zeros(1,length(x));

    % initial conditions
    y1(1) = 9;
    y2(1) = 8;

    % motion of a point
    f_1 = @(t, x1, x2) x2 + x1*(0.5 - x1^2 - x2^2);
    f_2 = @(t, x1, x2) -x1 + x2*(0.5 - x1^2 - x2^2);

    % loop
    for i=1:(length(x)-1)
        %k1
        k1_1 = f_1( x(i), y1(i), y2(i) );
        k1_2 = f_2( x(i), y1(i), y2(i) );

        %k2
        k2_1 = f_1( x(i)+0.5*h, y1(i)+0.5*h, y2(i)+0.5*h*k1_1 );
        k2_2 = f_2( x(i)+0.5*h, y1(i)+0.5*h, y2(i)+0.5*h*k1_2 );

        %k3
        k3_1 = f_1( x(i)+0.5*h, y1(i)+0.5*h, y2(i)+0.5*h*k2_1 );
        k3_2 = f_2( x(i)+0.5*h, y1(i)+0.5*h, y2(i)+0.5*h*k2_2 );
```

```
        %k4
        k4_1 = f_1( x(i)+h, y1(i)+h, y2(i) + h*k3_1 );
        k4_2 = f_2( x(i)+h, y1(i)+h, y2(i) + h*k3_2 );

        %y_(n+1)
        y1(i+1) = y1(i) + (1/6)*(k1_1 + 2*k2_1 + 2*k3_1 + k4_1)*h;
        y2(i+1) = y2(i) + (1/6)*(k1_2 + 2*k2_2 + 2*k3_2 + k4_2)*h;
    end
end
```

P5EC5E:

```
function [y1,y2,x] = P5EC5E(h, beginning, ending)

    %P5EC5E method so k=5
    k = 5;

    % values taken from book
    betaExplicit = [1901/720, -2774/720, 2616/720, -1274/720, 251/720];
    betaImplicit = [475/1440, 1427/1440, -789/1440, 482/1440, -173/1440,
27/1440];

    % motion of a point
    f_1 = @(t, x1, x2) x2 + x1*(0.5 - x1^2 - x2^2);
    f_2 = @(t, x1, x2) -x1 + x2*(0.5 - x1^2 - x2^2);

    % perform RK4 for first 5 iterations
    [y1,y2,x] = RK4(h, beginning, (k*h)-beginning);

    % start the loop from 6th iteration and do untill rounded integer value
    % of (end of interval-begin of interval)/(step size)
    for i=(k+1):(ceil(ending-beginning)/h)

        % at each iteraion x is bigger for h value
        x(i+1) = x(i) + h;

        %calculate sum of beta and function in order to have P
        sumP_1 = 0;
        sumP_2 = 0;
        for j=1:k
            sumP_1 = sumP_1 + betaExplicit(j)*f_1( x(i-j), y1(i-j), y2(i-
j) );
```

```
            sumP_2 = sumP_2 + betaExplicit(j)*f_2( x(i-j), y1(i-j), y2(i-
j) );
        end

        %P prediction
        y1(i+1) = y1(i) + h*sumP_1;
        y2(i+1) = y2(i) + h*sumP_2;

        %E evaluation
        f1 = f_1( x(i), y1(i+1), y2(i+1) );
        f2 = f_2( x(i), y1(i+1), y2(i+1) );

        %calculate sum of beta and function in order to have C
        sumC_1 = 0;
        sumC_2 = 0;
        for j=1:k
            sumC_1 = sumC_1 + betaImplicit(j)*f_1( x(i-j), y1(i-j), y2(i-
j) );
            sumC_2 = sumC_2 + betaImplicit(j)*f_2( x(i-j), y1(i-j), y2(i-
j) );
        end

        %C correction
        y1(i+1) = y1(i) + h*sumC_1 + h*betaImplicit(1)*f1;
        y2(i+1) = y2(i) + h*sumC_2 + h*betaImplicit(1)*f2;

        %E evaluation
        f1 = f_1( x(i), y1(i+1), y2(i+1) );
        f2 = f_2( x(i), y1(i+1), y2(i+1) );
    end
end
```

Compare time:

```
function compare_time(rk4,x1)
    if(rk4 == 1) % use RK4
        if(x1 == 1) % for x1
            [x1,~,t1] = RK4(0.002, 0, 15);

            [x1_2,~,t1_2] = RK4(0.007, 0, 15);

            % ode45( functions, interval, initial_conditions )
```

```
            [t_ode45,y_ode45] = ode45(@(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)], [0, 15], [9, 8]);
            % t is a time vector, y is a vector solutions of a function

            figure(15);
            hold on;

            plot(t1,x1);
            plot(t1_2,x1_2);
            plot(t_ode45,y_ode45(:,1));
            legend("optimal step-size","larger step-
size","ODE45",'Location','southoutside');
            title("RK4 x1 solutions vs time");
            xlabel("t");
            ylabel("x1");
            hold off;
        else % for x2
            [~,x2,t1] = RK4(0.002, 0, 15);

            [~,x2_2,t1_2] = RK4(0.007, 0, 15);

            % ode45( functions, interval, initial_conditions )
            [t_ode45,y_ode45] = ode45(@(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)], [0, 15], [9, 8]);
            % t is a time vector, y is a vector solutions of a function

            figure(16);
            hold on;

            plot(t1,x2);
            plot(t1_2,x2_2);
            plot(t_ode45,y_ode45(:,2));
            legend("optimal step-size","larger step-
size","ODE45",'Location','southoutside');
            title("RK4 x2 solutions vs time");
            xlabel("t");
            ylabel("x2");
            hold off;
        end
    else % use P5EC5E
        if(x1 == 1) % for x1
            [x1,~,t1] = P5EC5E(0.002, 0, 15);

            [x1_2,~,t1_2] = P5EC5E(0.007, 0, 15);
```

```
            % ode45( functions, interval, initial_conditions )
            [t_ode45,y_ode45] = ode45(@(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)], [0, 15], [9, 8]);
            % t is a time vector, y is a vector solutions of a function

            figure(17);
            hold on;

            plot(t1,x1);
            plot(t1_2,x1_2);
            plot(t_ode45,y_ode45(:,1));
            xlim([0 15]);
            legend("optimal step-size","larger step-
size","ODE45",'Location','southoutside');
            title("P5EC5E x1 solutions vs time");
            xlabel("t");
            ylabel("x1");
            hold off;
        else % for x2
            [~,x2,t1] = P5EC5E(0.002, 0, 15);

            [~,x2_2,t1_2] = P5EC5E(0.007, 0, 15);

            % ode45( functions, interval, initial_conditions )
            [t_ode45,y_ode45] = ode45(@(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)], [0, 15], [9, 8]);
            % t is a time vector, y is a vector solutions of a function

            figure(18);
            hold on;

            plot(t1,x2);
            plot(t1_2,x2_2);
            plot(t_ode45,y_ode45(:,2));
            xlim([0 15]);
            legend("optimal step-size","larger step-
size","ODE45",'Location','southoutside');
            title("P5EC5E x2 solutions vs time");
            xlabel("t");
            ylabel("x2");
            hold off;
        end
    end
```

```
end
```

Task 2 part B:

```
[x1,x2,t,h,error1,error2] = RK4_auto(0.0001,0,15,1e-9,1e-9);

% ode45( functions, interval, initial_conditions )
[~,y_ode45] = ode45(@(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)], [0, 15], [9, 8]);

figure(1);
hold on;

plot(x1,x2);
plot(y_ode45(:,1),y_ode45(:,2));
legend("RK4 auto", "ODE45", 'Location', 'Best');
title("RK4 auto vs ODE45");
hold off;

figure(2);
plot(t,h);
title("Step size vs time");

figure(3);
hold on;
plot(t,x1);
plot(t,x2);
hold off;
title("problem solution vs time");
legend("x1","x2",'Location','southoutside');
```

RK4 initial:

```
function [root1,root2,x] = RK4_initial(h, beginning, ending, initial_1,
initial_2)

    x = beginning:h:ending;

    % initial conditions
    y1(1) = initial_1;
    y2(1) = initial_2;
```

```
    % motion of a point
    f_1 = @(t, x1, x2) x2 + x1*(0.5 - x1^2 - x2^2);
    f_2 = @(t, x1, x2) -x1 + x2*(0.5 - x1^2 - x2^2);

    maxi = 1;
    % loop
    for i=1:(length(x)-1)
        %k1
        k1_1 = f_1( x(i), y1(i), y2(i) );
        k1_2 = f_2( x(i), y1(i), y2(i) );

        %k2
        k2_1 = f_1( x(i)+0.5*h, y1(i)+0.5*h, y2(i)+0.5*h*k1_1 );
        k2_2 = f_2( x(i)+0.5*h, y1(i)+0.5*h, y2(i)+0.5*h*k1_2 );

        %k3
        k3_1 = f_1( x(i)+0.5*h, y1(i)+0.5*h, y2(i)+0.5*h*k2_1 );
        k3_2 = f_2( x(i)+0.5*h, y1(i)+0.5*h, y2(i)+0.5*h*k2_2 );

        %k4
        k4_1 = f_1( x(i)+h, y1(i)+h, y2(i) + h*k3_1 );
        k4_2 = f_2( x(i)+h, y1(i)+h, y2(i) + h*k3_2 );

        %y_(n+1)
        y1(i+1) = y1(i) + (1/6)*(k1_1 + 2*k2_1 + 2*k3_1 + k4_1)*h;
        y2(i+1) = y2(i) + (1/6)*(k1_2 + 2*k2_2 + 2*k3_2 + k4_2)*h;

        % maxi needed for the last element
        maxi = i+1;
    end

    % only the last element
    root1 = y1(maxi);
    root2 = y2(maxi);
end
```

RK4 auto:

```
function [y1,y2,x,auto_h,error_1,error_2] =
RK4_auto(h,beginning,ending,epsr,epsa)

    % store different values of h
```

```
    auto_h(1) = h;

    % this time whole x cannot be defined at this point
    x(1) = beginning;

    % initial conditions
    y1(1) = 9;
    y2(1) = 8;

    % 2^p and order p=4 so 2^4=16

    % places for errors
    error_1(1) = 0;
    error_2(1) = 0;

    for i=1:100000

        % after single step
        [y1_single,y2_single,time_single] =
RK4_initial(auto_h(i),x(i),auto_h(i)+x(i),y1(i),y2(i));


        % after double step
        [y1_double,y2_double,time_double] =
RK4_initial(auto_h(i)*0.5,x(i),auto_h(i)+x(i),y1(i),y2(i));


        % from book
        y1(i+1) = y1_single + (16/15)*(y1_double-y1_single); %x1
        y2(i+1) = y2_single + (16/15)*(y2_double-y2_single); %x2

        % error estimate for a double step (for RK)
        delta_y1_double = ((y1_double - y1_single)/15);
        error_1(i+1)= delta_y1_double; % store error
        delta_y2_double = ((y2_double - y2_single)/15);
        error_2(i+1)= delta_y2_double; % store error

        % accuracy parameters
        eps1 = (abs(y1(i))*epsr)+epsa; %epsr=relative tolerance,
epsa=absolute tolerance
        eps2 = (abs(y2(i))*epsr)+epsa;

        % step-size correction
        alpha_1 = (eps1/abs(delta_y1_double))^(1/5);
```

```
        alpha_2 = (eps2/abs(delta_y2_double))^(1/5);

        if(alpha_1 < alpha_2)
            smallest_alpha = alpha_1;
        else
            smallest_alpha = alpha_2;
        end

        % with the use of safety factor s=0.9
        auto_h(i+1) = 0.9*smallest_alpha*auto_h(i);

        x(i+1)=x(i);

        % from the block diagram on page 174
        if(0.9*smallest_alpha >=1) %s*alpha
            if(x(i)+auto_h(i)>=ending) % meets b or after b
                break;
            else
                x(i+1) = x(i) + auto_h(i);
                % beta=5 taken from book, page 174
                minimum_temp = min(5*auto_h(i),auto_h(i+1));
                auto_h(i+1) = min(minimum_temp,ending-x(i));
            end
        else
            if(auto_h(i+1) < eps) % eps as h_min
                disp("Error: No solution for assumed accuracy")
                break;
            end
        end
    end
end
```