Numerical Methods

Project Number: C9

Name Surname: Beste Baydur

Index Number: 295597

Winter Semester 2020

TASK 1

1. Aim of the task

1) Finding function formula based on given points in task (can be found below), with use of approximation method called the least-squares.
2) Finding function formula based on given points in task, with use of approximation method called the least-squares with use of QR distribution.
3) Results comparison obtained by both methods.

x_i	y_i
-5	-18.2370
-4	-7.6583
-3	-1.4146
-2	0.1113
-1	2.3030
0	1.6890
1	0.9738
2	0.9726
3	0.5941
4	-1.8716
5	-6.1512

2. Approximation

The aim of the approximation is to find a simpler function, from a chosen class of approximating functions, which is appropriately close to origin function at given points.

3. Method

The approximation problem can be defined in the following way: to find a function F belonging to set X_n as close as possible to f in a certain sense, usually in the sense of a certain distance δ(f – F) defined by a norm || ||. Thus, approximation of the function f means finding the coefficients $a_0,\ldots,a_n$ of F as the norm ||f-F|| is minimized.

To perform least-square approximation, we must define A as a matrix Nxn, where N – number of samples, n – degree of polynomial, for every

$$A(i,j) = x_j^{i-1}$$

where i = 1,2,3,...,n; j = 1,2,3,...,N

Solving least-square task comes down to finding the vector a which contain coefficients of polynomial. In this case we will research two approaches:

- Using and solving system of normal equations:

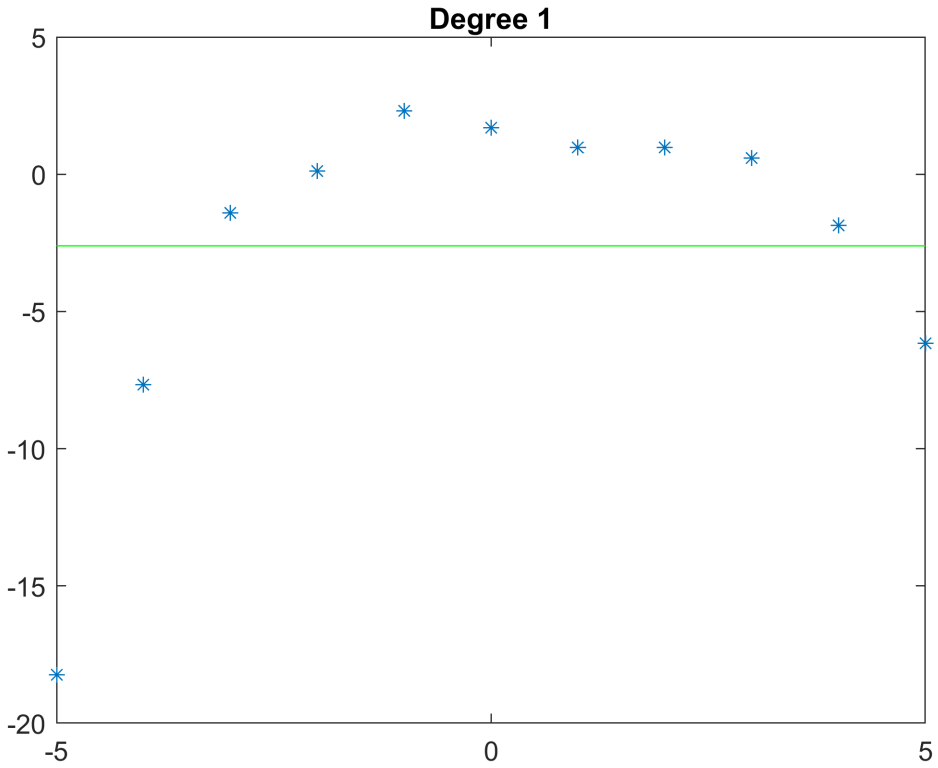
$$A^T A\, a = A^T y$$

- Using and solving system resulting in QR factorisation:

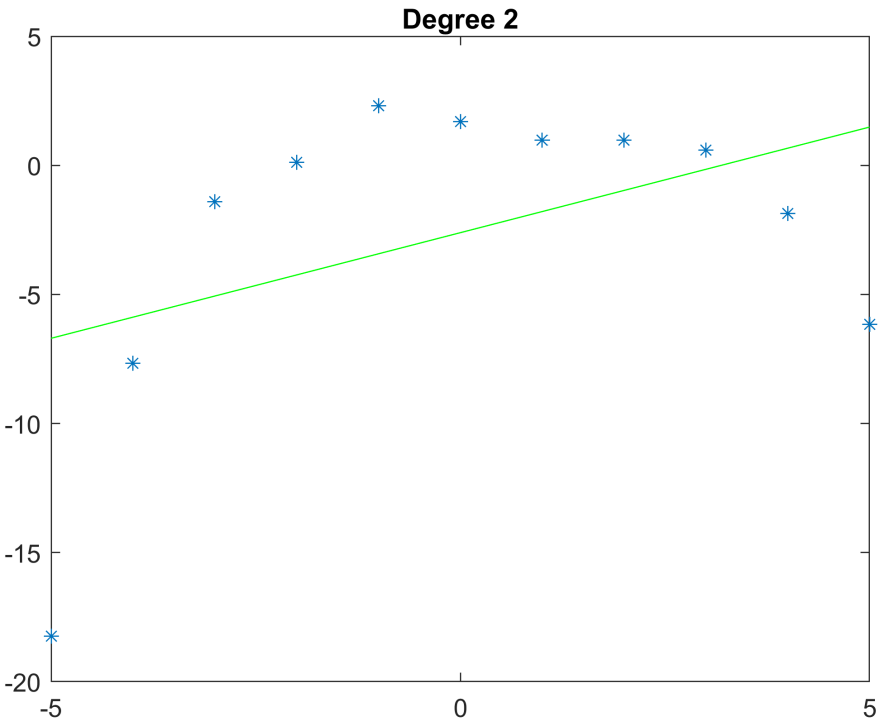
$$A = QR \qquad Ra = Q^T y$$

4. Graphs of polynomial degree 1-10

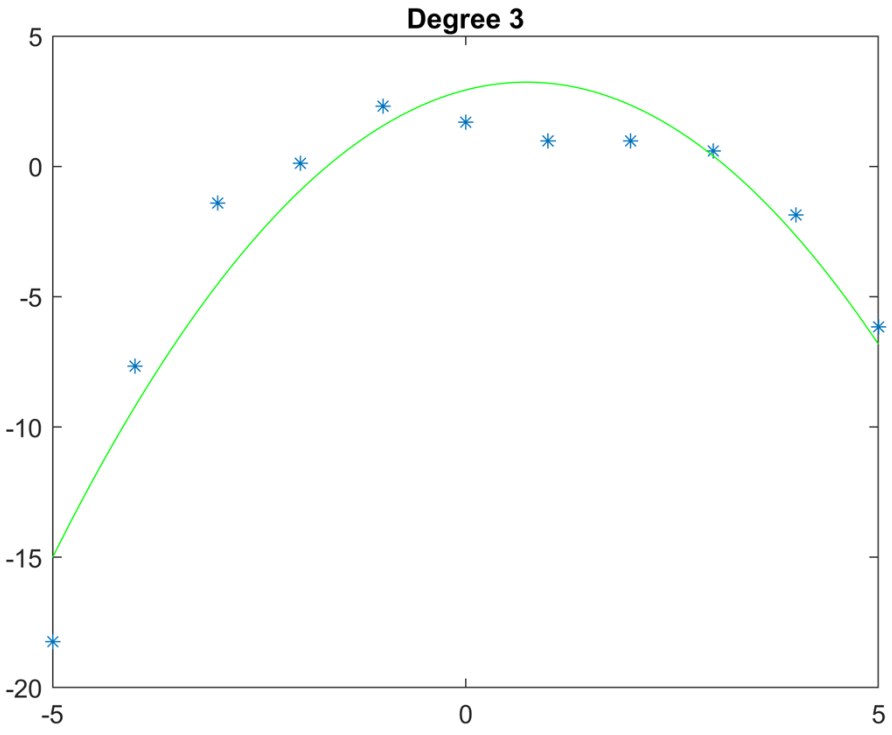
Polynomial degree 1



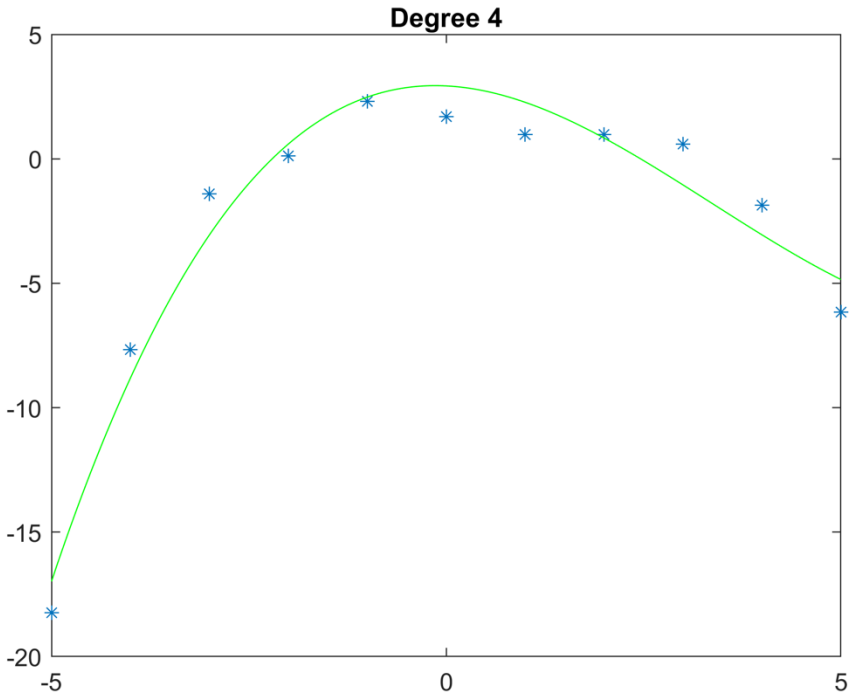
Polynomial degree 2



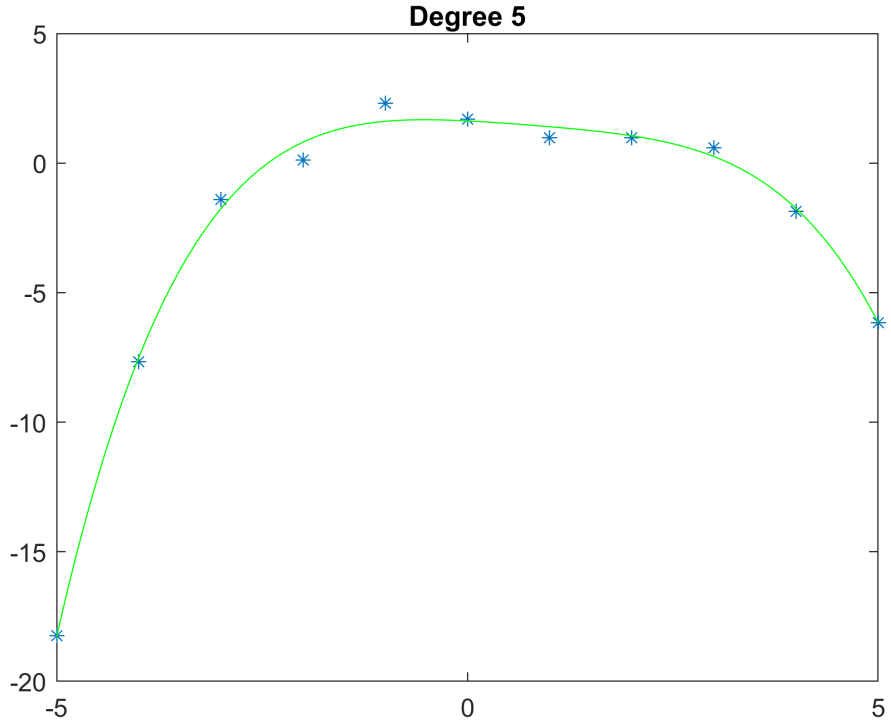
Polynomial degree 3



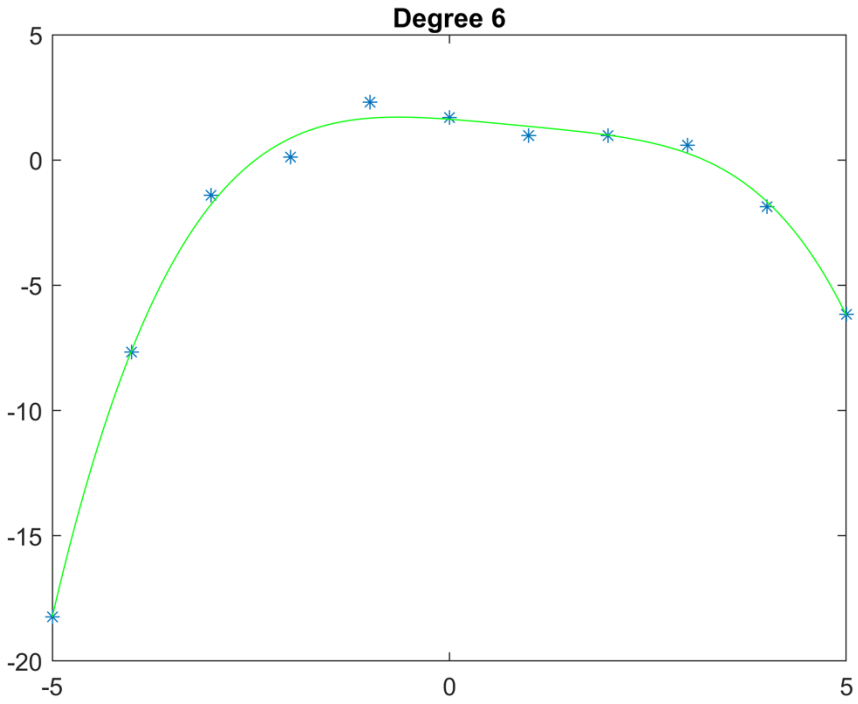
Polynomial degree 4



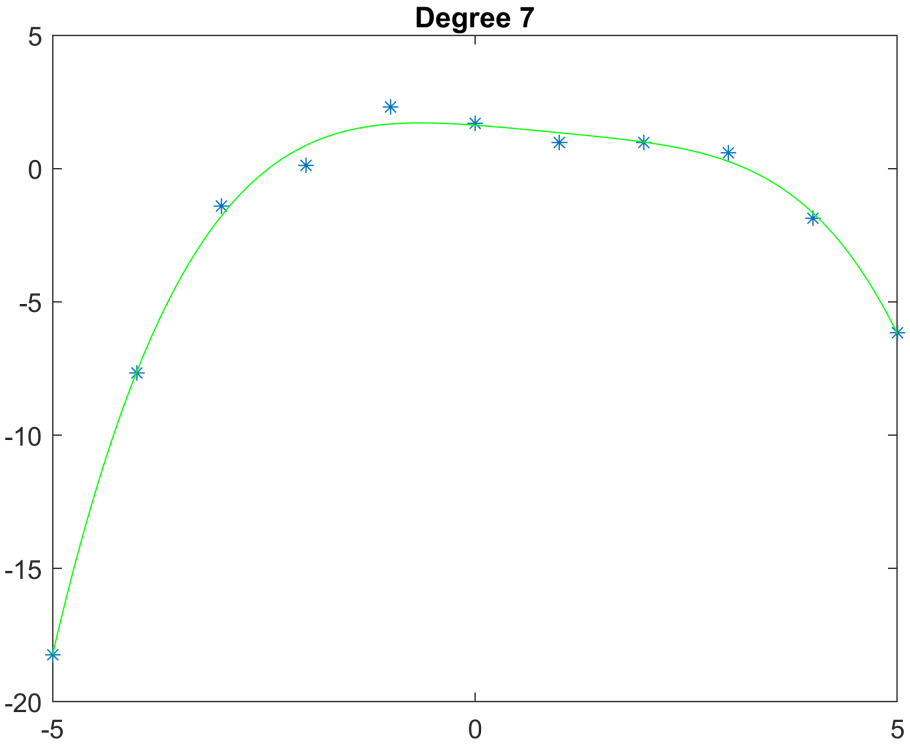
Polynomial degree 5



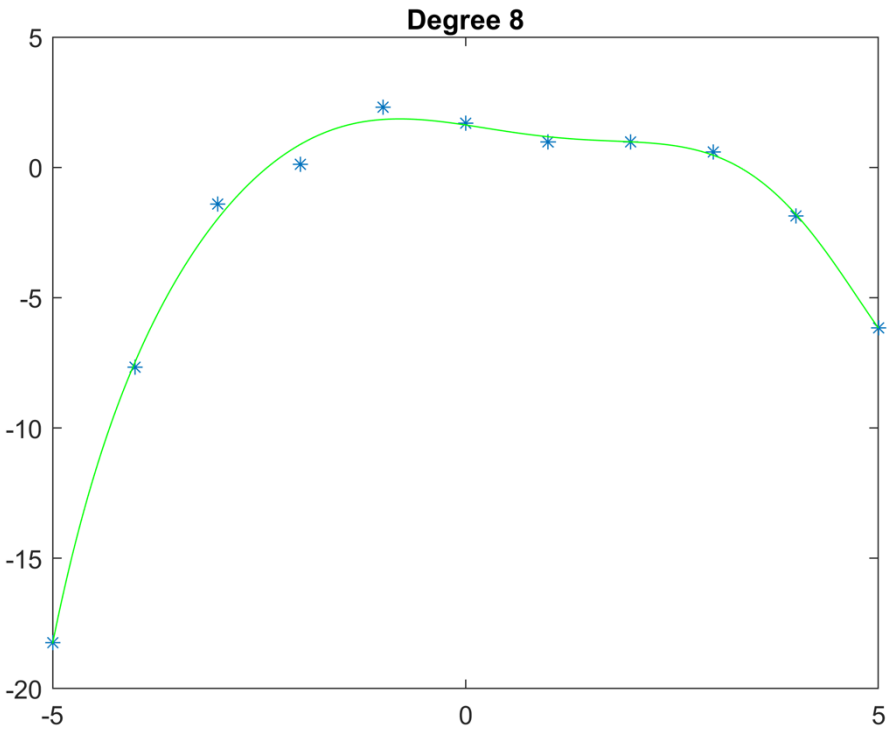
Polynomial degree 6



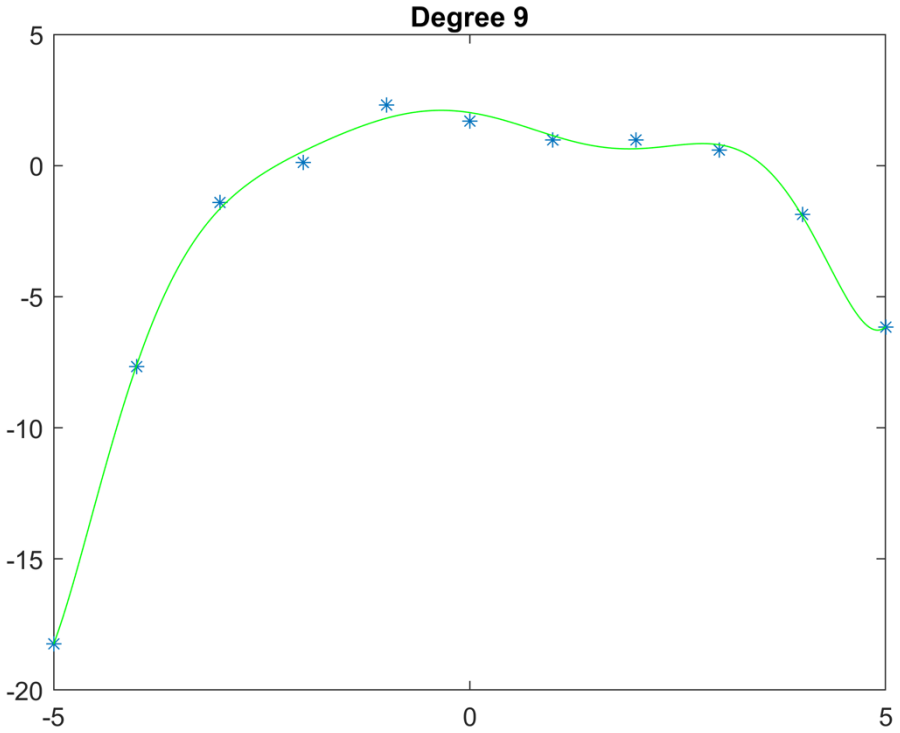
Polynomial degree 7



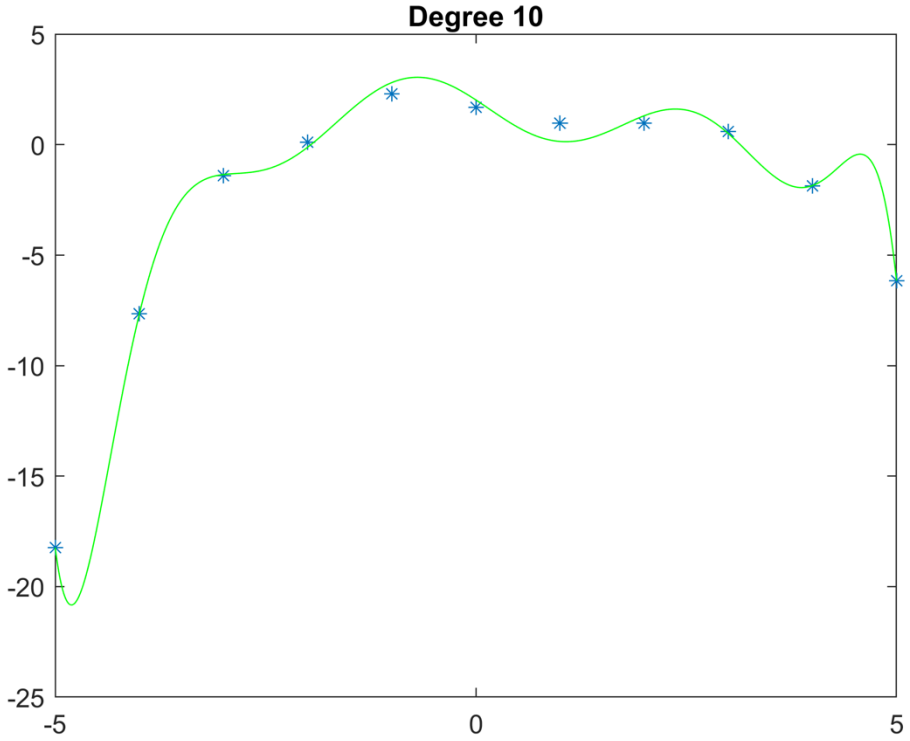
Polynomial degree 8



Polynomial degree 9



Polynomial degree 10



5. Residuum table:

Degree	Residuum error	Condition number of Gram's matrix G
1	19.2399	1
2	17.2204	10
3	5.7882	408.7796
4	3.8608	8.5584e+03
5	1.1980	3.1798e+05
6	1.1832	7.4675e+06
7	1.1832	2.8316e+08
8	1.1070	7.6462e+09
9	0.8845	3.3055e+11
10	1.1085	1.5167e+13

6. Conclusion

We can conclude that along with the increase in the polynomial order, numerical errors increase (matrix G loses a good condition). The QR distribution is an accurate method of approximation. It should also be noted that by increasing the polynomial row, at some point, it stops approximating the original function, and begins with the one that most closely matches the collected data. Taking into account the values of approximation errors, it can be noticed that as the polynomial row increases, the approximation error decreases (new function crosses marked measurements), which indicates that the approximating function strive to value of collected samples.

TASK 2

1. Aim of the task

The second task is to determine the trajectory of the motion of a point define by the equations:

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

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

on the interval [0, 15]. The following initial conditions are given : $x1(0) = 0$, $x2(0) = 0.4$. We will evaluate the solution in two following ways :

1) 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 solutions significantly but its increase does.

2) 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.

2. Runge-Kutta method of 4th order(RK4)

Runge-Kutta method 4th order(RK4, “classical”) can be define using following formulas :

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}h\left(k_1 + 2k_2 + 2k_3 + k_4\right) \\ k_1 &= f\left(x_n, y_n\right), \\ k_2 &= f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1), \\ k_3 &= f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2), \\ k_4 &= f\left(x_n + h, y_n + hk_3\right). \end{aligned}$$

The coefficient k_1 represents the derivative at (x_n,y_n). The value k_2 is calculated as in the modified.

Euler's method – as a derivative of the solution calculated by the standard Euler's method at the midpoint $(x_n + 1/2\ h, y_n + 1/2\ hk_1)$, the dashed tangent line correspond to this derivative. Next, the value k_3 is calculated similarly as it was for k_2, but this time at the point $(x_n + 1/2\ h, y_n + 1/2\ hk_2)$ - i.e., with a tangent line corresponding to k_2. Finally, we start with this line from the initial point until the endpoint $(x_n + h)$, i.e., the derivative k_4 of a solution at the point $(x_n + h, y_n + hk_3)$-i.e over the one step interval: one at the initial point, two at the midpoint and on at the endpoint. The final approximation of the solution derivative for the final full step of the method is calculated as a weighted mean value of these derivatives, with the weight 1 for the initial and end points and the weight 2 for the midpoint.

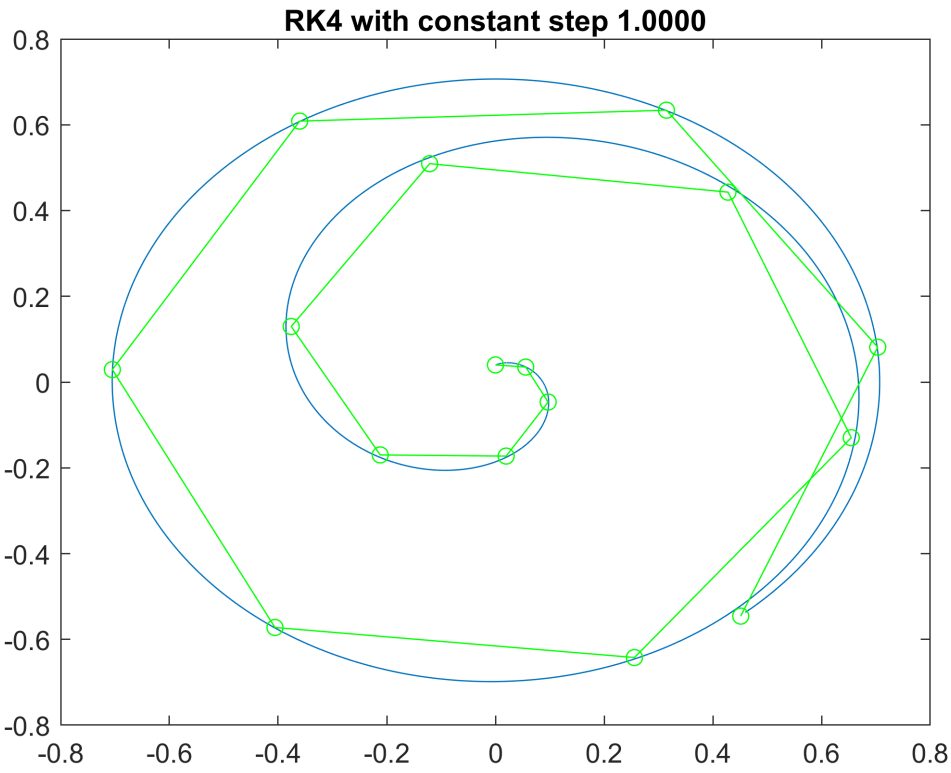
The step was decreased until the plot started to present sufficient accuracy. Error of single step can be calculated using formula :

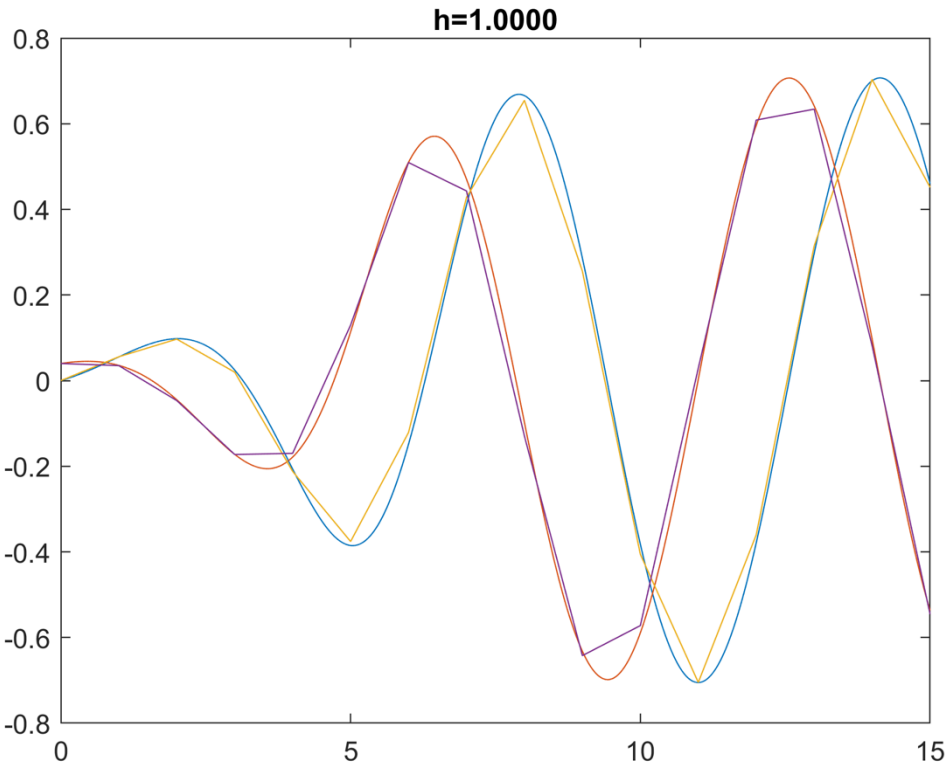
$$\delta_n(h) = \frac{2^p}{2^p - 1}(y_n^{(2)} - y_n^{(1)})$$

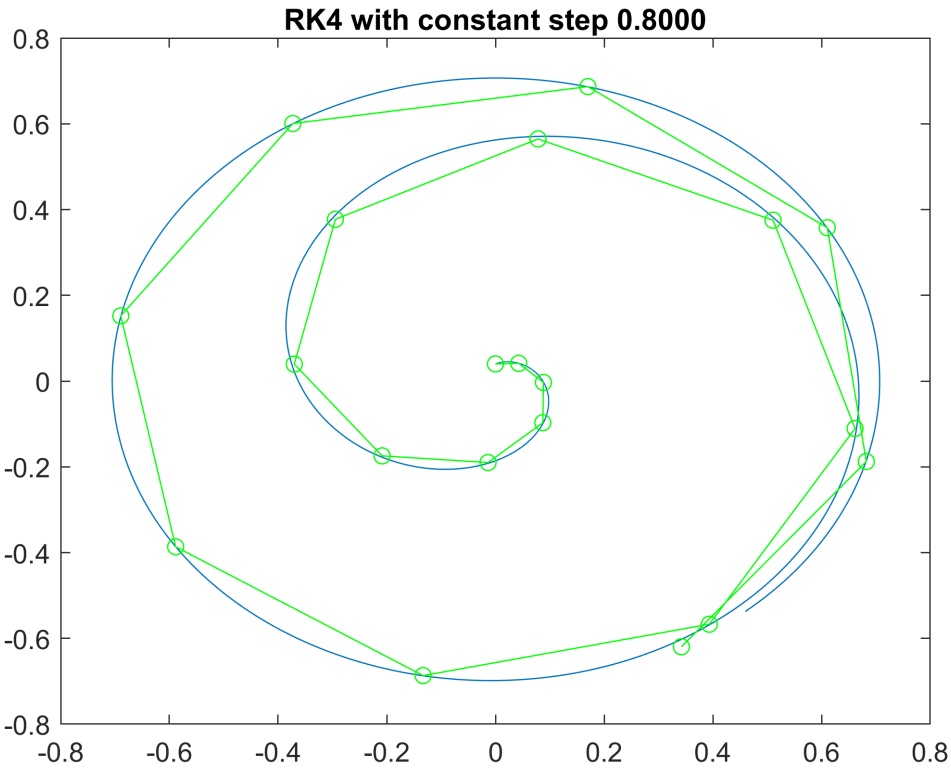
2.1 Runge-Kutta method of 4th order, a constant step size

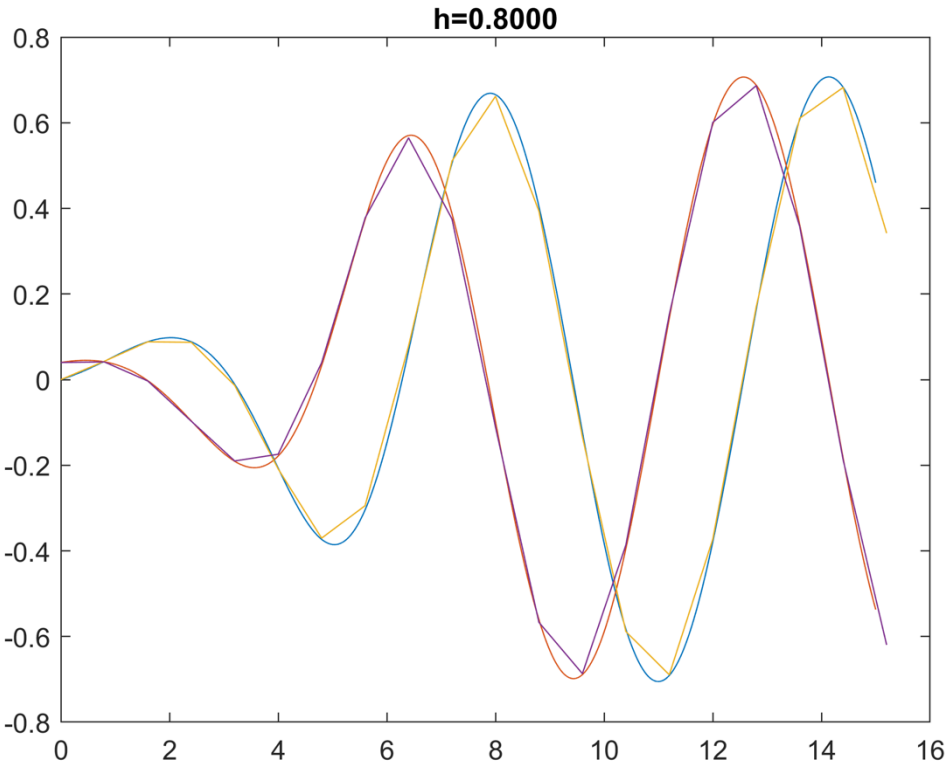
Results for different step size obtained using Runge-Kutta algorithm of 4th order are presented below:

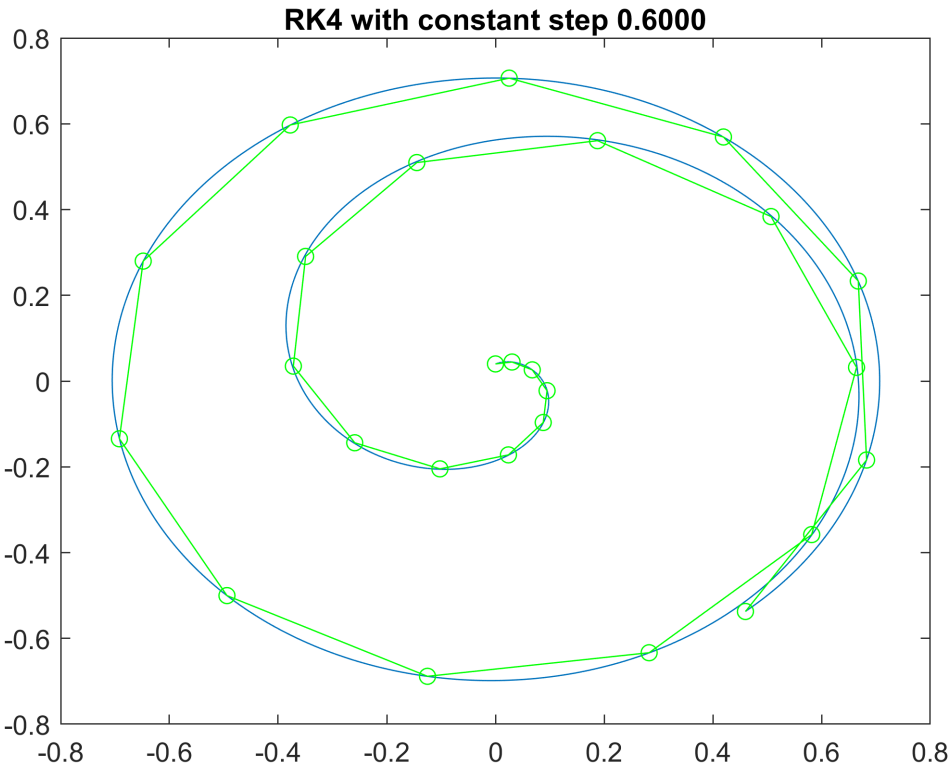
Graphs of RK4 with constant steps:

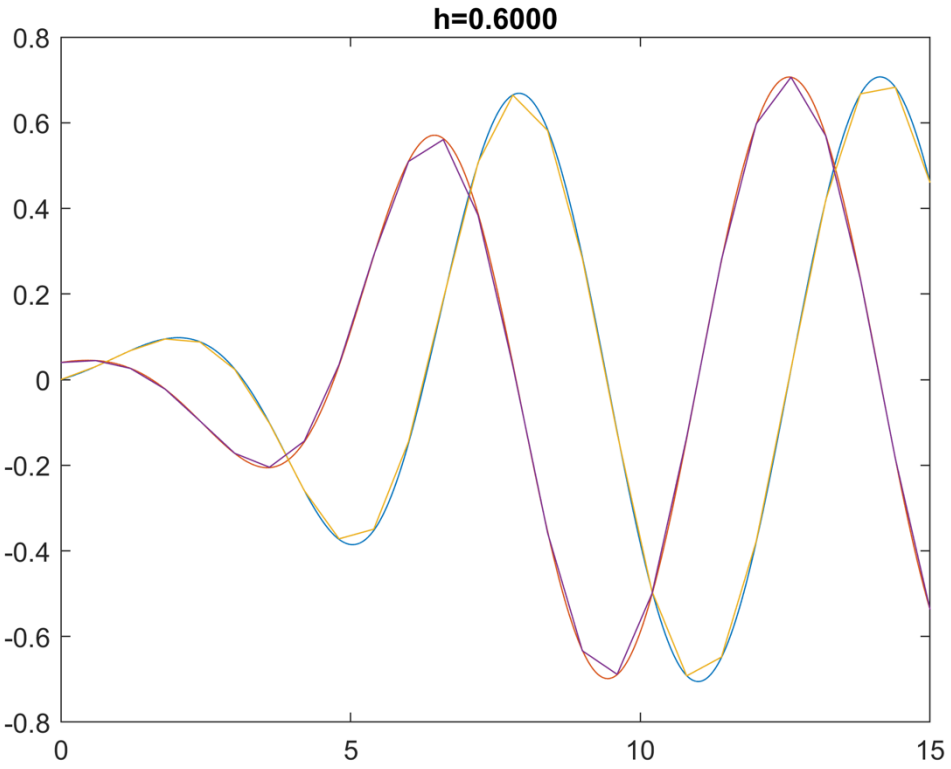


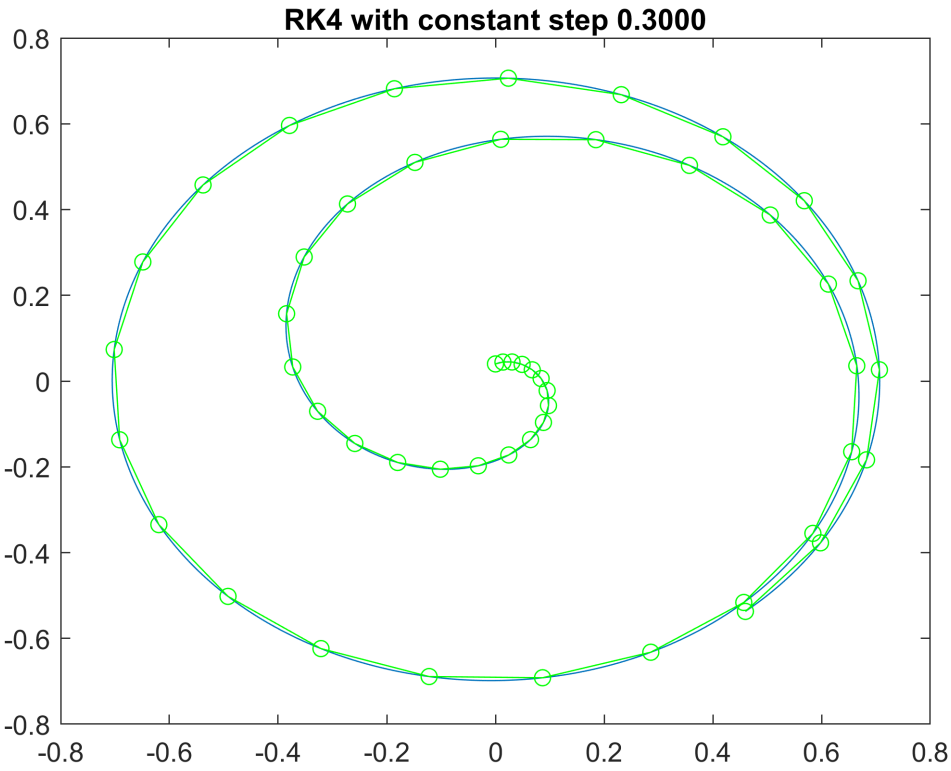


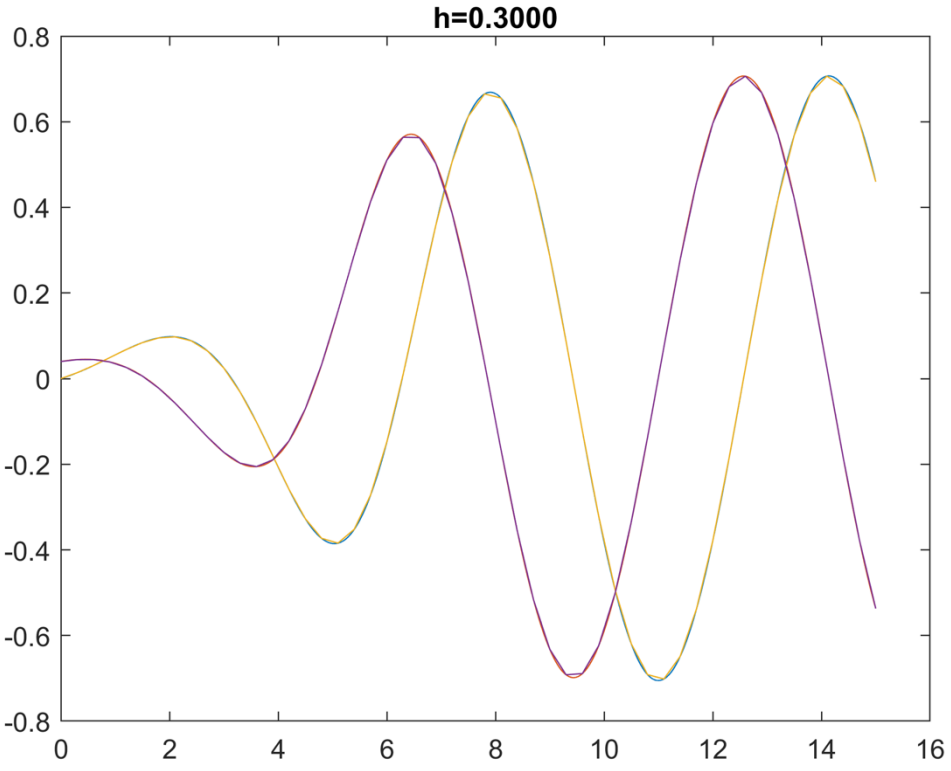

RK4 with constant step 0.8000
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8

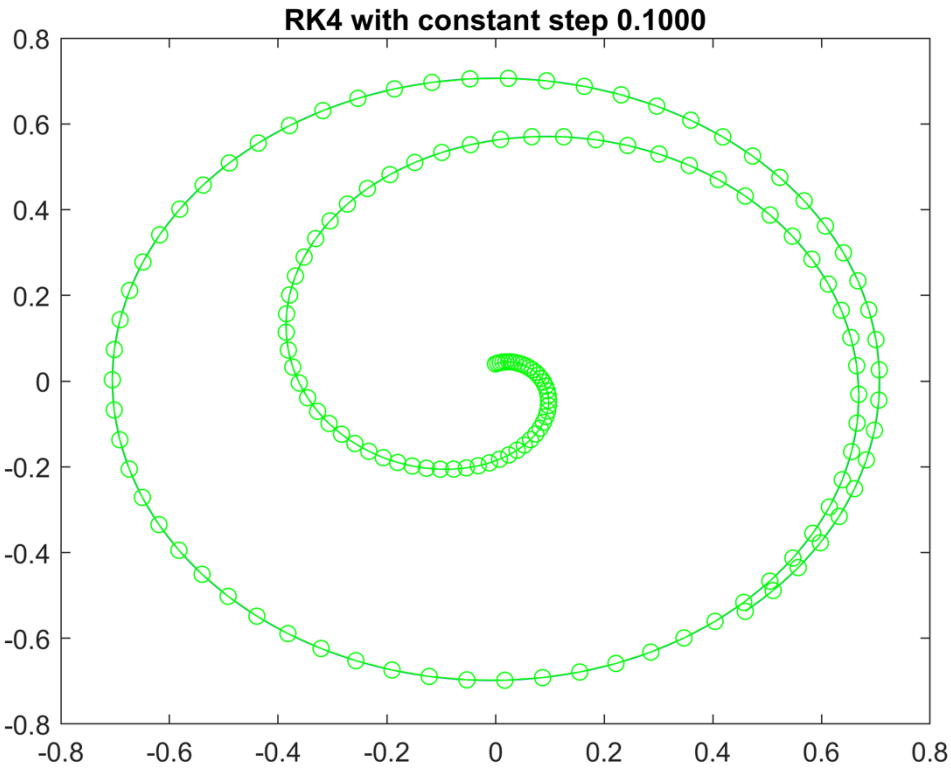

h=0.8000
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
2
4
6
8
10
12
14
16

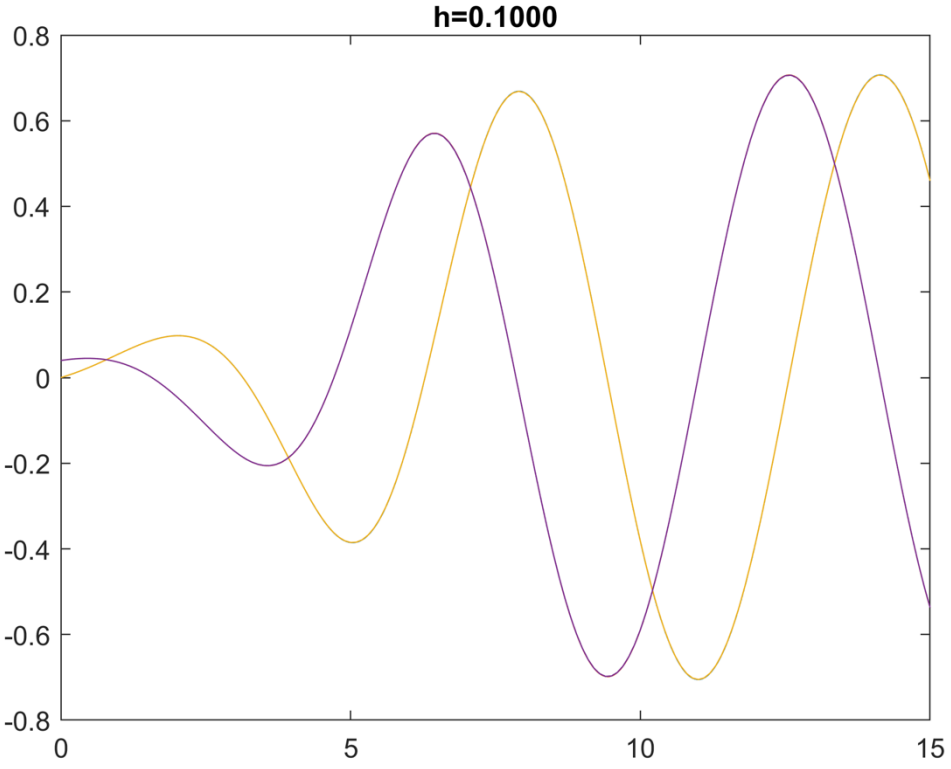

RK4 with constant step 0.6000
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8


h=0.6000
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
5
10
15


RK4 with constant step 0.3000
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8


h=0.3000
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
2
4
6
8
10
12
14
16

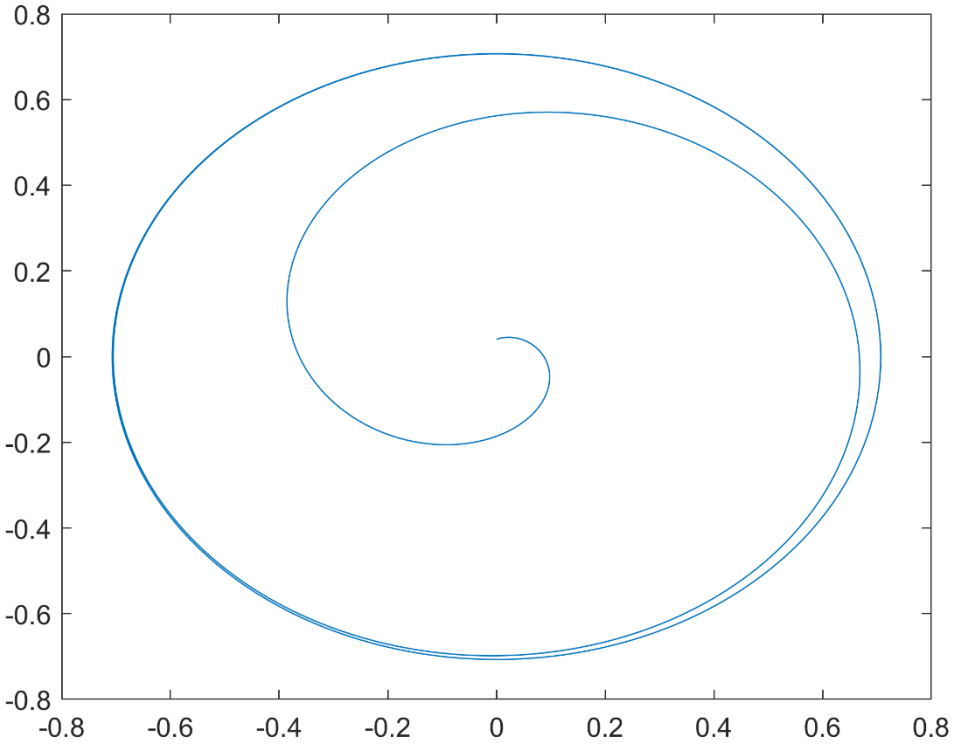

RK4 with constant step 0.1000
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8


h=0.1000
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
5
10
15

Conclusion

Function that has been represented is proper, because trajectories are being noticed to be very similar, to observe some bigger difference we would have to have a closer look at plots.

3. RK4 with a variable step



Conclusion

Variable step method is definitely better than constant step, because it checks steps on trajectory and adjusts for it, which gives better result for example, on the curves. It approximates and adapts to the trajectory much better, and results into better accuracy. Additionally, it is taking less tie, rather than estimating better constant step.

4. Adams PC (P_5EC_5E)

An initial value problem

$$y'(x) = f(x, y(x)),$$
$$y(a) = y_a, \quad x \in [a, b],$$

Can be equivalently formulated in the form of integral equation

$$y(x) = y(a) + \int_a^x f(t, y(t))\, dt.$$

Considering this integral on the interval $[x_{n-1}, x_n]$

$$y(x_n) = y(x_{n-1}) + \int_{x_{n-1}}^{x_n} f(t, y(t))\, dt,$$

leads to Adams methods.

Adams Methods can be in two different forms which are explicit.

- Explicit Adams methods (Adams-Bashforth methods)

The function f (x, y (x)) is replaced by an interpolation polynomial W (x) of order k−1, calculated at the points x_{n-1}, ..., x_{n-k} with the corresponding function values y (x_{n-j}) ≈ y_{n-j} . Using the Lagrange interpolation formula) we have:

$$f(x, y(x)) \approx W(x) = \sum_{j=1}^{k} f(x_{n-j}, y_{n-j}) \cdot L_j(x),$$

$$y_n = y_{n-1} + \sum_{j=1}^{k} f(x_{n-j}, y_{n-j}) \cdot \int_{x_{n-1}}^{x_n} L_j(t)\, dt,$$

where L_j (x) are the Lagrange polynomials,

$$L_j(x) = \prod_{m=1, m\neq j}^{k} \frac{x - x_{n-m}}{x_{n-j} - x_{n-m}}.$$

Assuming that the points are equally spaced, $x_{n-j} = x_n - jh$, $j = 1, 2, ..., k$, the integration process yields

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

where values of the coefficients β_j are given

- Implicit Adams methods (Adams-Moulton methods)

  The function f (x, y (x)) is now replaced by an interpolation poly- nomial W * (x) of order k calculated at the points x_n , x_{n-1} , ..., x_{n-k} with the corresponding solution values $y(x_{n-j}) \approx y_{n-j}$. Reasoning in the same way as it was done in the case of the explicit methods, we finally get

$$\begin{aligned} y_n &= y_{n-1} + h \sum_{j=0}^{k} \beta_j^* \cdot f\left(x_{n-j}, y_{n-j}\right) \\ &= y_{n-1} + h \cdot \beta_0^* \cdot f\left(x_n, y_n\right) + h \sum_{j=1}^{k} \beta_j^* \cdot f\left(x_{n-j}, y_{n-j}\right) \end{aligned}$$

where values of the parameters β_j^*, for k = 1, ..., 7, that are given.

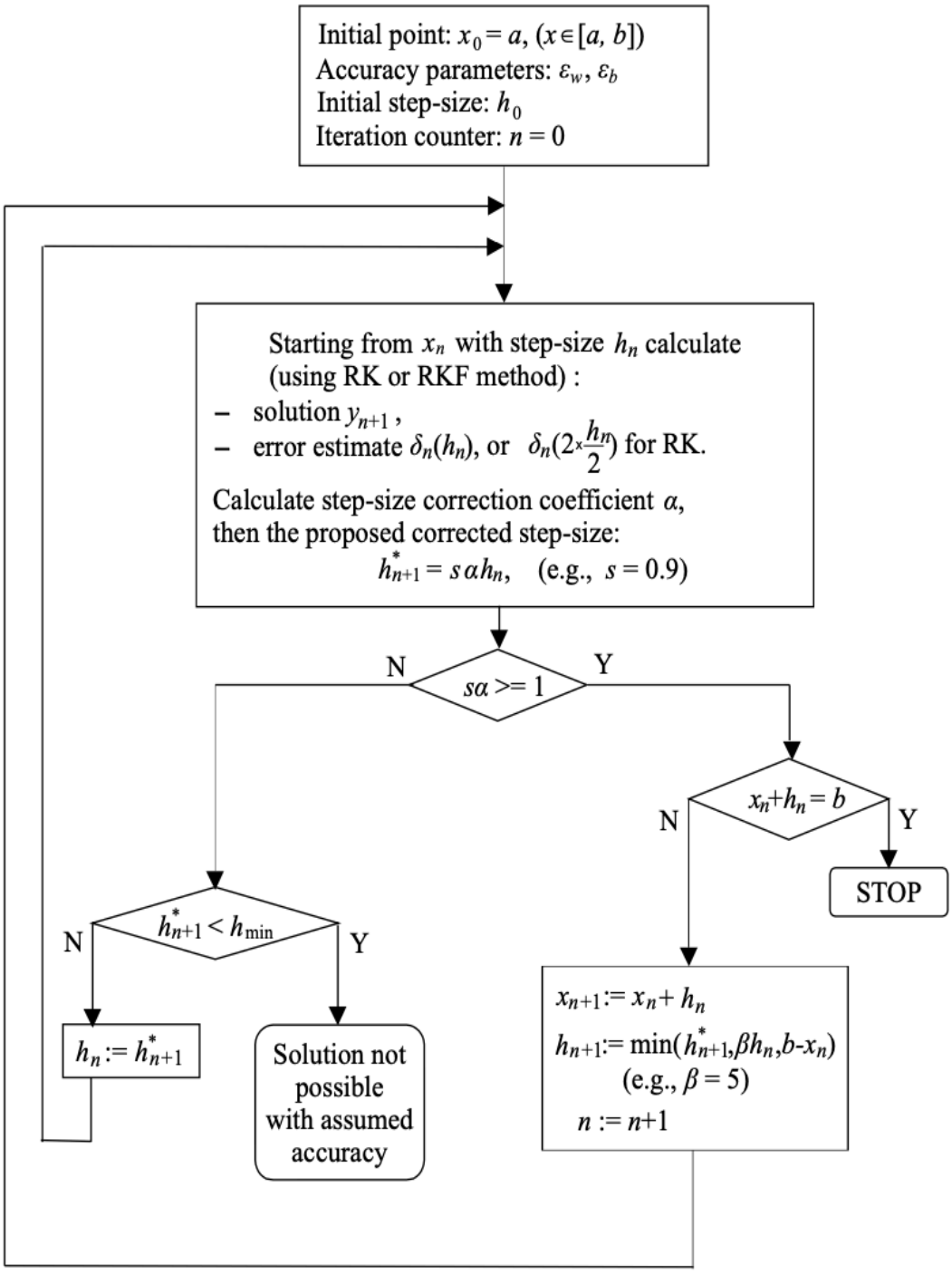
The predictor-corrector method P_kEC_kE:

For the Adams Methods the P_kEC_kE algorithm has the following form:

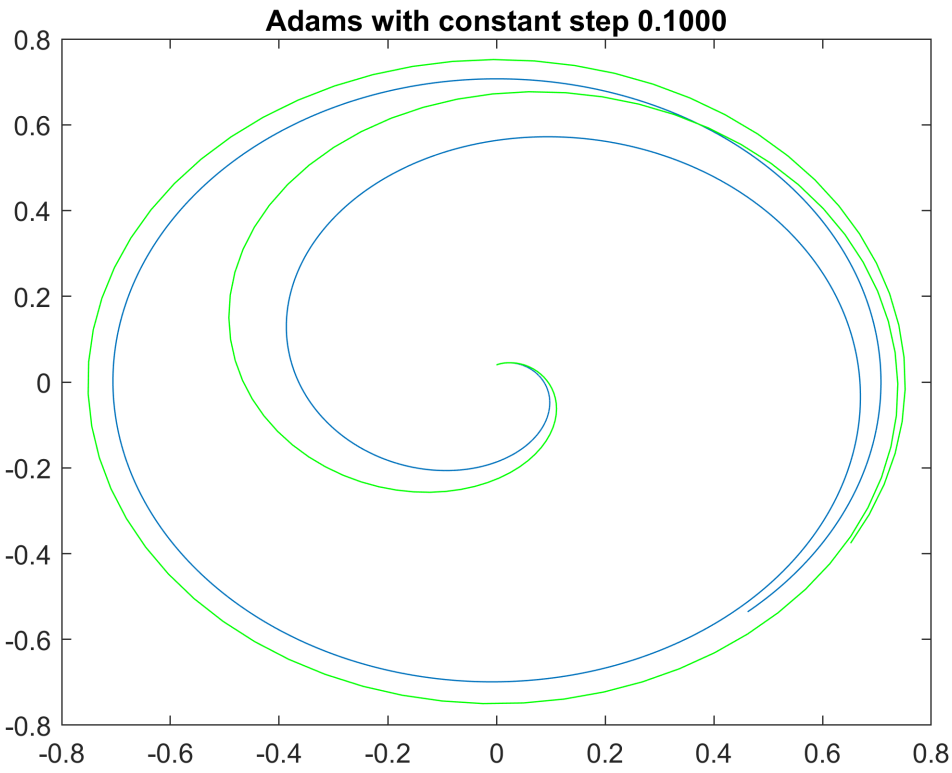
$$\begin{aligned} &\text{P:} && y_n^{[0]} = \sum_{j=1}^{k} \alpha_i y_{n-j} + h \sum_{j=1}^{k} \beta_j f_{n-j}, && \text{(P – prediction)} \\ &\text{E:} && f_n^{[0]} = f(x_n, y_n^{[0]}), && \text{(E – evaluation)} \\ &\text{C:} && y_n = \sum_{j=1}^{k} \alpha_j^* y_{n-j} + h \sum_{j=1}^{k} \beta_j^* f_{n-j}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

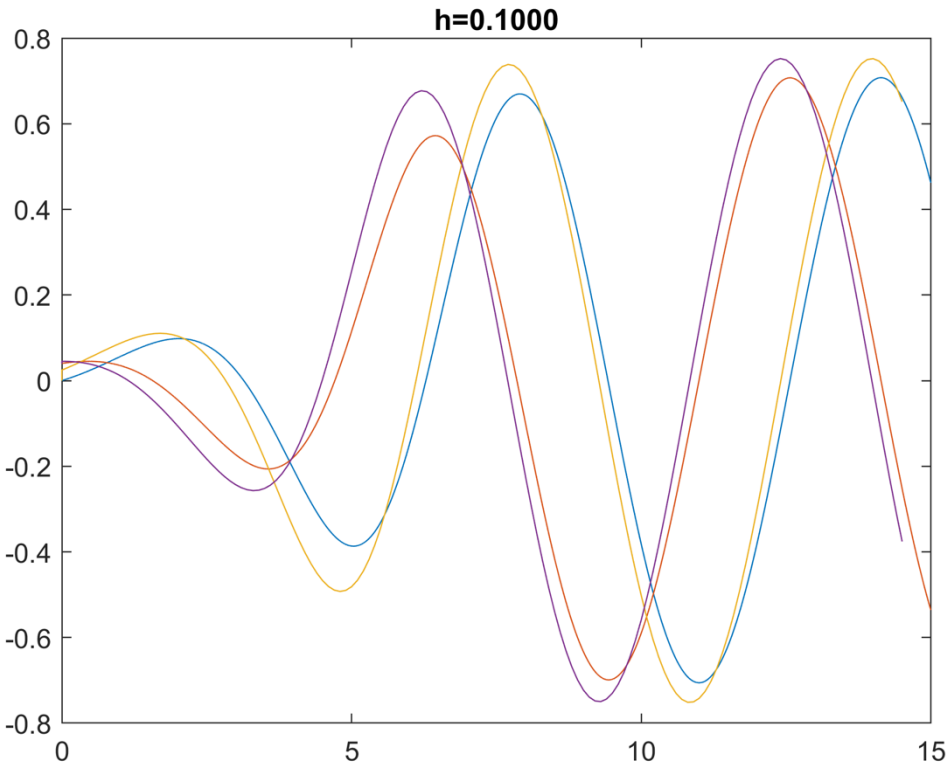
And the error approximation can be calculated by the formula:

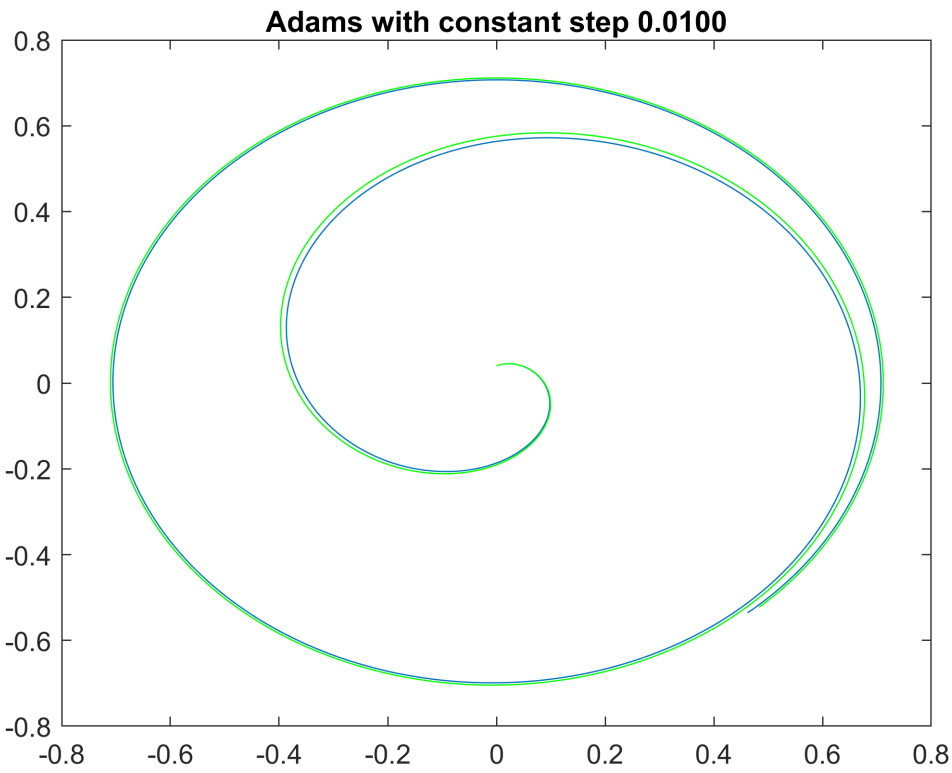
$$\delta_n\left(h_{n-1}\right) = \frac{\gamma_k^*}{\gamma_k - \gamma_k^*}(y_n^{[0]} - y_n).$$

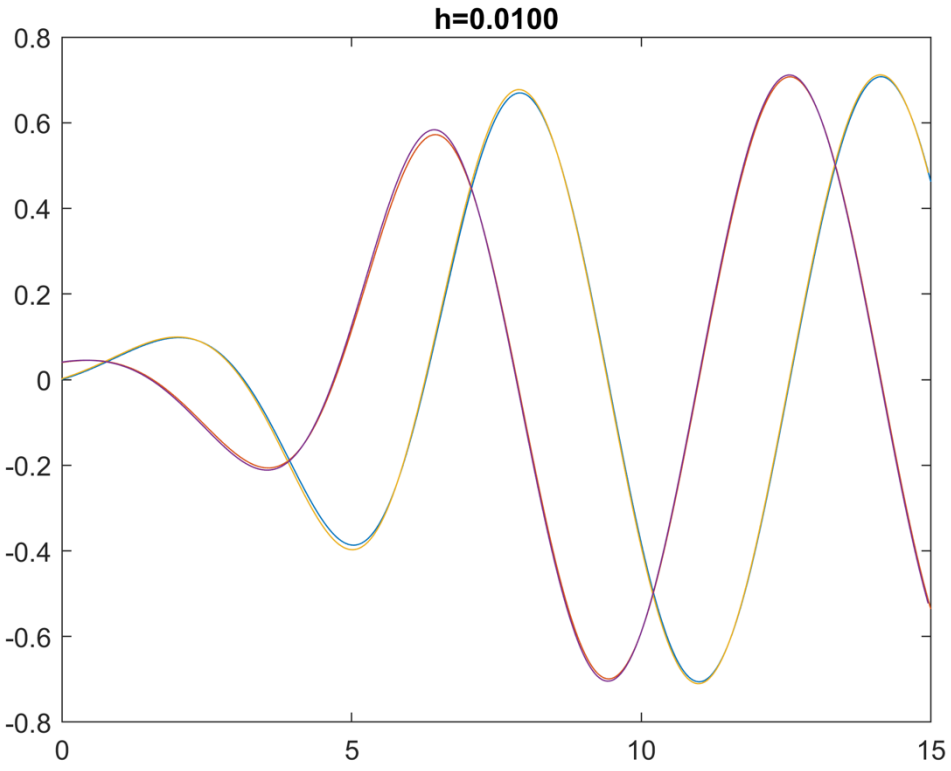

Initial point: $x_0 = a$, $(x \in [a, b])$
Accuracy parameters: ε_w, ε_b
Initial step-size: h_0
Iteration counter: $n = 0$
Starting from x_n with step-size h_n calculate (using RK or RKF method) :
– solution y_{n+1} ,
– error estimate $\delta_n(h_n)$, or $\delta_n(2\times\frac{h_n}{2})$ for RK.
Calculate step-size correction coefficient α, then the proposed corrected step-size:
$h^*_{n+1} = s\alpha h_n$, (e.g., $s = 0.9$)
N
$s\alpha >= 1$
Y
N
$x_n+h_n = b$
Y
STOP
N
$h^*_{n+1} < h_{\min}$
Y
$h_n := h^*_{n+1}$
Solution not possible with assumed accuracy
$x_{n+1} := x_n + h_n$
$h_{n+1} := \min(h^*_{n+1}, \beta h_n, b-x_n)$
(e.g., $\beta = 5$)
$n := n+1$

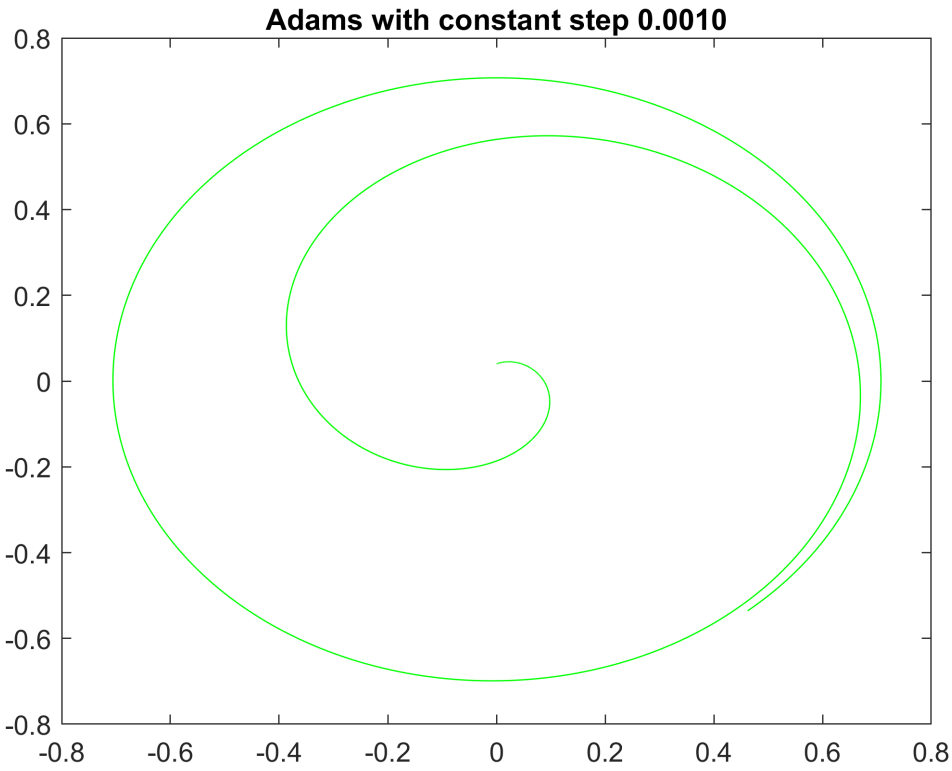
Graphs of adams with constant step:

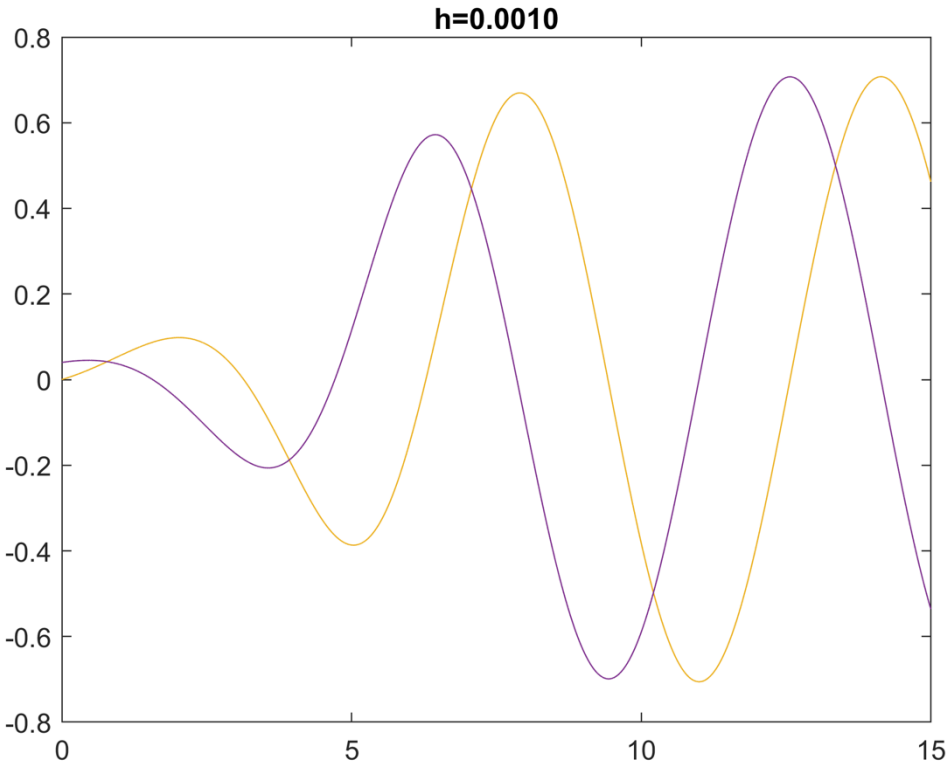





Adams with constant step 0.0100
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8


h=0.0100
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
5
10
15





Conclusion:

Function that has been represented is proper, because trajectories are being noticed to be very similar, to observe some bigger difference we would have to have a closer look at plots.

Appendix

1. Task1

```
clear;
close all;
X=-5:5;
Y=[-18.2370,-7.6583,-1.4146,0.1113,2.3030,1.6890,0.9738,0.9726,0.5941, -
1.8716,-6.1512];
for n=1:10
 n
 figure
 plot(X,Y,'*');
 hold on
 N=length(Y);
 G=zeros(n,n);
 b=zeros(n,1);
 for i=1:n
 for k=1:n
 G(i,k)=0;
 for j=1:N
 G(i,k)=G(i,k)+(X(j))^(i+k-2);
 end;
 end;
 for j=1:N
 b(i)=b(i)+Y(j)*((X(j))^(i-1));
 end;
 end
 [Q,R]=qrmgs(G);
 a=R\(Q'*b) ;
 a_=flip(a);
% a_

 disp('Error as Eucledian norm = || y - Fx ||:');
 er=Y-polyval(a_',X);
 norm(er)

 disp('Condition number of Gram''s matrix G:');
 cond(G)
 plot_title = sprintf("Degree %i",n);
 for i=1:n
 plot_title = strcat(plot_title,'+', sprintf("%.4fx^%i", a(i),i-1));
 end
 X_=-5:0.01:5;
 P_=polyval(a_',X_);
 plot(X_,P_,'g');
 title(plot_title)
 hold off
 print(sprintf('Degree %i',n),'-dpng','-r400');
end
function [Q,R]=qrmgs(A)
 [m n]=size(A);
 Q=zeros(m,n);
 R=zeros(n,n);
 d=zeros(1,n);
 for i=1:n
 Q(:,i)=A(:,i);
20
 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
 for i=1: ndd == norm(Q(:,i));
 Q(:,i)=Q(:,i)/dd;
 R(i,i:n)=R(i,i:n)*dd;
 end
end
```

2. RK4 variable step:

```
function [] = rk4change(xs1,xs2,step)
tic
stepd=step/2;
i=int32(15/stepd);
Error1=zeros(int32(i/2)+1,1);
Error2=zeros(int32(i/2)+1,1);
X1=zeros(int32(i/2)+1,1);
X2=zeros(int32(i/2)+1,1);
Y=zeros(int32(i/2)+1,1);
x1=xs1;                          %x1,x2,x1d,x1d
x2=xs2;
x1d=xs1;
x2d=xs2;
X1(1,1)=x1;
X2(1,1)=x2;
Y (1,1)=0;
Error1(1,1)=xs1;
Error2(1,1)=xs2;
halfstep=step/2
halfstepd=stepd/2;
for n=1:i
    if(mod(n,2) == 0)
    k11=dx1(x1,x2);
    k12=dx2(x1,x2);
    k21=dx1((x1+halfstep*k11),(x2+halfstep*k12));
    k22=dx2((x1+halfstep*k11),(x2+halfstep*k12));
    k31=dx1((x1+halfstep*k21),(x2+halfstep*k22));
    k32=dx2((x1+halfstep*k21),(x2+halfstep*k22));
    k41=dx1((x1+step*k31),(x2+step*k32));
    k42=dx2((x1+step*k31),(x2+step*k32));
    x1=x1+(1/6)*step*(k11+2*k21+2*k31+k41);
    x2=x2+(1/6)*step*(k12+2*k22+2*k32+k42);
    X1((n/2)+1,1)=x1;
    X2((n/2)+1,1)=x2;
    end
    k11d=dx1(x1d,x2d);
    k12d=dx2(x1d,x2d);
    k21d=dx1((x1d+halfstepd*k11d),(x2d+halfstepd*k12d));
    k22d=dx2((x1d+halfstepd*k11d),(x2d+halfstepd*k12d));
    k31d=dx1((x1d+halfstepd*k21d),(x2d+halfstepd*k22d));
    k32d=dx2((x1d+halfstepd*k21d),(x2d+halfstepd*k22d));
    k41d=dx1((x1d+stepd*k31d),(x2d+stepd*k32d));
    k42d=dx2((x1d+stepd*k31d),(x2d+stepd*k32d));
    x1d=x1d+(1/6)*stepd*(k11d+2*k21d+2*k31d+k41d);
    x2d=x2d+(1/6)*stepd*(k12d+2*k22d+2*k32d+k42d);
    if(mod(n,2) == 0)
        Error1((n/2)+1,1)=(16/15)*abs(x1d-X1((n/2)+1,1)); %15 to 2^k-1 ,
where k is deg (e.g. 4)
        Error2((n/2)+1,1)=(16/15)*abs(x2d-X2((n/2)+1,1));
```

```
        Y ((n/2)+1,1)= double((n/2)+1)*step;
    end
end
toc;
%plot(Y,Error1,'--',Y,Error2,'-');
axis([0 15 0 0.2]);
%plot(Y,X1,'--',Y,X2,'-');
plot(X1,X2);
end
```

3. RK4 constant step

```
function[y] = rk4()
h = 0.002;
x1=10;
x2=9;
10 9; 0 7; 7 0; 0.001 0.001
tic;
t=0:h:20;

y(:,1) = [x1 x2];

for i=1:(length(t)-1)
    k11=dx1(x1,x2);
    k12=dx2(x1,x2);

    k21=dx1(x1+0.5*h,x2+0.5*h*k11);
    k22=dx2(x1+0.5*h,x2+0.5*h*k12);

    k31=dx1(x1+0.5*h,x2+0.5*h*k21);
    k32=dx2(x1+0.5*h,x2+0.5*h*k22);

    k41=dx1(x1+h,x2+h*k31);
    k42=dx2(x1+h,x2+h*k31);

    x1=x1+(h/6)*(k11+k41+2*(k21+k31));
    x2=x2+(h/6)*(k12+k42+2*(k22+k32));

    y(:,i+1)=[x1 x2];
end
plot(y(1,:),y(2,:));
toc;
end
```

4. Adams

```
function [y] = pc()
tic;
h = 0.005;
x1=7;
x2=0;
t=0:h:20;

y(:,1) = [x1 x2];

for i=1:3
```

```
    k11=dx1(x1,x2);
    k12=dx2(x1,x2);

    k21=dx1(x1+0.5*h,x2+0.5*h*k11);
    k22=dx2(x1+0.5*h,x2+0.5*h*k12);

    k31=dx1(x1+0.5*h,x2+0.5*h*k21);
    k32=dx2(x1+0.5*h,x2+0.5*h*k22);

    k41=dx1(x1+h,x2+h*k31);
    k42=dx2(x1+h,x2+h*k31);

    x1=x1+(h/6)*(k11+k41+2*(k21+k31));
    x2=x2+(h/6)*(k12+k42+2*(k22+k32));

    y(:,i+1)=[x1 x2];
end
for i = 4:(length(t))
    tmp1 = x1 + (h/24)*55*dx1(x1,x2) - 59*(h/24)*dx1(y(1,i-1),y(2,i-1)) +
37*(h/24)*dx1(y(1,i-2),y(2,i-2)) - 9*(h/24)*dx1(y(1,i-3),y(2,i-3));
    tmp2 = x2 + (h/24)*55*dx2(x1,x2) - 59*(h/24)*dx2(y(1,i-1),y(2,i-1)) +
37*(h/24)*dx2(y(1,i-2),y(2,i-2)) - 9*(h/24)*dx2(y(1,i-3),y(2,i-3));
    x1 = x1 + (h/720)*646*dx1(x1,x2) - 264*(h/720)*dx1(y(1,i-1),y(2,i-1)) +
106*(h/720)*dx1(y(1,i-2),y(2,i-2)) - 19*(h/720)*dx1(y(1,i-3),y(2,i-3)) +
h*(251/720)*dx1(tmp1, tmp2);
    x2 = x2 + (h/720)*646*dx2(x1,x2) - 264*(h/720)*dx2(y(1,i-1),y(2,i-1)) +
106*(h/720)*dx2(y(1,i-2),y(2,i-2)) - 19*(h/720)*dx2(y(1,i-3),y(2,i-3)) +
h*(251/720)*dx2(tmp1, tmp2);

    y(:,i)=[x1 x2];
end
plot(y(1,:),y(2,:));
%plot(0:h:20,y(1,:),'-',0:h:20,y(2,:),'-')
toc;
end
```