Numerical Methods

Project C (no. 30)

Report

Author: Kacper Wojakowski

kwojakow

293064

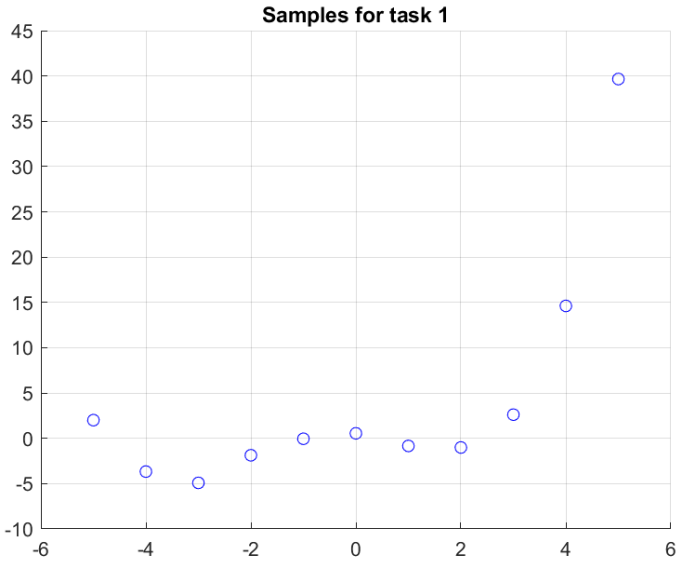
TABLE OF CONTENTS

1 LEAST SQUARES APPROXIMATION

1.1 Description of the task

The aim of this task is to write a MATLAB program determining a polynomial function *y=f(x)* that best first the following set of experimental data using least-squares approximation:

x_i	y_i
-5	2.0081
-4	-3.6689
-3	-4.9164
-2	-1.8700
-1	-0.0454
0	0.5504
1	-0.8392
2	-1.0113
3	2.6133
4	14.6156
5	39.6554



The set will be tested for polynomials of various degrees, and two algorithms will be implemented: a system of normal equations without QR factorization, and one with QR factorization.

1.2 Theory

Approximation is, given a set of values of a function *f(x)*, finding a simpler function *F(x)*, from a chosen class of approximating functions, which is as close as possible to *f(x)*. It can be applied when *f(x)* is too complicated to be effectively applied in analysis or design, or in modelling, to derive a function from a set of known points (samples).

If we denote:

f(x) – an original function, to be approximated

F(x) – an approximating function

And assume:

X – a linear function space, $f \in X$

X_n – a (n+1)-dimensional subspace of X, with a basis $\Phi_0(x), \ldots, \Phi_n(x)$, i.e.,

$$F(x) \in X_n \Leftrightarrow F(x) = a_0\phi_0(x) + a_1\phi_1(x) + \cdots + a_n\phi_n(x)$$

Where $a_i \in \mathbb{R}, i = 0,1, \ldots, n$, are the coefficients

The problem of approximation can be defined as finding a function $F^* \in X_n$ closest to *f* in a certain sense, usually in sense of distance *δ(f-F)* defined by a norm ||·||,

$$\forall F \in X_n \quad \delta(f - F^*) \stackrel{\text{def}}{=} \|f - F^*\| \leq \|f - F\|$$

Therefore, the approximation of the function *f* means finding the coefficients $a_0, \ldots, a_n$ of *F* such that the norm ||*f* - *F*|| is minimized.

There are a few typical approximations, resulting from the choice of norm:

- A *uniform continuous approximation* of a continuous function *f(x)* defined on a closed interval [*a, b*]:

$$\|F - f\| = \sup_{x \in [a,b]} |F(x) - f(x)| \qquad (Tschebyschev\ norm)$$

- A *continuous least-squares approximation* of a function *f(x)*, quadratically integrable over a closed interval [*a, b*], i.e., $f(x) \in L_p^2[a, b]$:

$$\|F - f\| = \sqrt{\int_a^b p(x)[F(x) - f(x)]^2 dx}$$

Where *p(x)* is a weighting function

- A *uniform discrete approximation* of *f(x)*, know on a finite set of N+1 points only:

$$\|F - f\| = \max\{|F(x_0) - f(x_0)|, |F(x_1) - f(x_1)|, \ldots, |F(x_n) - f(x_n)|\}$$

- A *discrete least-squares approximation* (the least squares method) of a function *f(x)* know on a finite set of *N+1* points only:

$$\|F - f\| = \sqrt{\sum_{j=0}^{N} p(x_j)[F(x_j) - f(x_j)]^2}$$

Where *p(x)* is a weighting function.

For this task, we will consider the last approximation, that is *discrete least-squares approximation*.

1.3 Algorithm used

The algorithm(s) used in this task is *discrete least-squares approximation* in two variations: with and without QR shifts. Assuming that we are given a finite number of points $x_0, x_1, \dots, x_N$ $(x_i \neq x_j)$ and their values $y_j = f(x_j)$, $j = 0,1,2, \dots, N$, we can declare $\phi_i(x), i = 0,1, \dots, n$ as a basis of a space $X_n \subseteq \mathrm{X}$ of interpolating functions, i.e.

$$\forall F \in X_n \;\; F(x) = \sum_{i=0}^{n} a_i \phi_i(x)$$

The aim of the approximation problem is to find values of the parameters $a_0, a_1, \dots, a_n$ defining our approximating, which minimize the error defined as

$$H(a_0, \dots, a_n) \stackrel{\text{def}}{=} \sum_{j=0}^{N} \left[f(x_j) - \sum_{i=0}^{n} a_i \phi_i(x_j) \right]^2$$

To simplify the presentation the weighting function *p(·)* is left out.

We can derive the formula for $a_0, a_1, \dots, a_n$ from the necessary (and in this case also sufficient) condition for a minimum:

$$\frac{\partial H}{\partial a_k} = -2 \sum_{j=0}^{N} \left[f(x_j) - \sum_{i=0}^{n} a_i \phi_i(x_j) \right] \cdot \phi_k(x_j) = 0, \qquad k = 0, \dots, n$$

Which gives us a set of linear equations with unknowns $a_0, a_1, \dots, a_n$, which is called a set of *normal equations*. It can be written in a simple form if we define the scalar product:

$$\langle \phi_i, \phi_k \rangle \stackrel{\text{def}}{=} \sum_{j=0}^{N} \phi_i(x_j) \phi_k(x_j)$$

Then the set of normal equations can be represented as:

$$\begin{bmatrix} \langle \phi_0, \phi_0 \rangle & \langle \phi_1, \phi_0 \rangle & \cdots & \langle \phi_n, \phi_0 \rangle \\ \langle \phi_0, \phi_1 \rangle & \langle \phi_1, \phi_1 \rangle & \cdots & \langle \phi_n, \phi_1 \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \phi_0, \phi_n \rangle & \langle \phi_1, \phi_n \rangle & \cdots & \langle \phi_n, \phi_n \rangle \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \langle \phi_0, f \rangle \\ \langle \phi_1, f \rangle \\ \vdots \\ \langle \phi_n, f \rangle \end{bmatrix}$$

Let us define the following matrix as matrix **A**:

$$\boldsymbol{A} = \begin{bmatrix} \phi_0(x_0) & \phi_1(x_0) & \cdots & \phi_n(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \cdots & \phi_n(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \cdots & \phi_n(x_N) \end{bmatrix}$$

And also define

$$\boldsymbol{a} = [a_0, a_1, \dots, a_n]^T$$
$$\boldsymbol{y} = [y_0, y_1, \dots, y_N]^T, \qquad y_j = f(x_j), \qquad j = 0,1, \dots, N$$

Using the definitions above, we can write the set of normal equations as

$$\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{a} = \boldsymbol{A}^T\boldsymbol{y}$$

Because matrix **A** has full rank, then the Gram's matrix $\mathbf{A}^\mathsf{T}\mathbf{A}$ is nonsingular – this implies that the solution of the set of normal equations is unique. However, the matrix $\mathbf{A}^\mathsf{T}\mathbf{A}$ can be badly conditioned – its condition number is a square of the condition number of **A**. In such case it is recommended to use a method basing on the QR factorization of **A**. The recommended factorization is the thin (economical) factorization – the same one as implemented in Project A. - $\boldsymbol{A}_{m\times n} = \boldsymbol{Q}_{m\times n}\boldsymbol{R}_{n\times n}$. The set of equations can then be written in such a form:

$$\boldsymbol{R}^T\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{R}\boldsymbol{x} = \boldsymbol{R}^T\boldsymbol{Q}^T\boldsymbol{b}$$

Which can be simplified into

$$\boldsymbol{R}\boldsymbol{x} = \boldsymbol{Q}^T\boldsymbol{b}$$

We will approximate the function as a polynomial $W_n(x)$ (where n is the order of the polynomial). By definition, the degree of the polynomial is usually much smaller than the given number of points of the original function:

$$N \gg n$$

We will use the following polynomial basis:

$$\phi_0(x) = 1, \qquad \phi_1(x) = x, \qquad \phi_2(x) = x^2, \dots, \qquad \phi_n(x) = x^n$$

Therefore,

$$F(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$

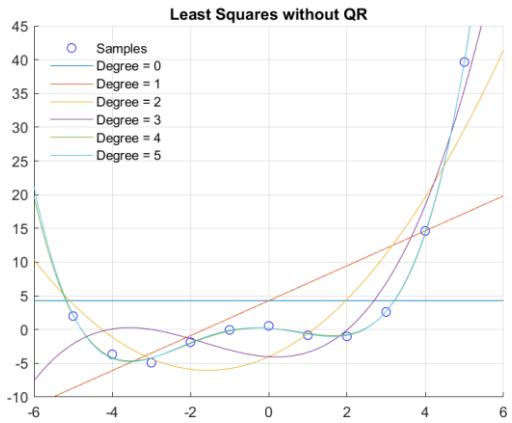
So finally, we can define **A, a** and **y** as:

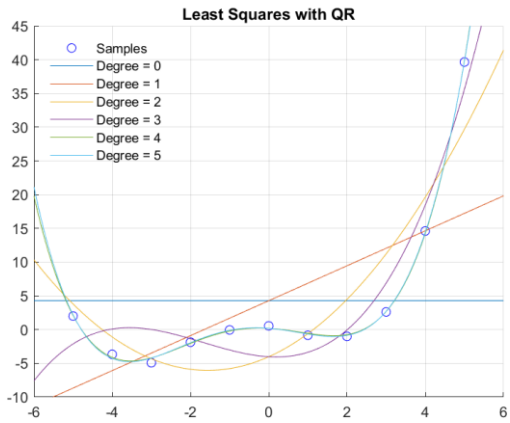
$$\boldsymbol{A} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^n \end{bmatrix}$$

$$\boldsymbol{a} = [a_0, a_1, \dots, a_n]^T$$
$$\boldsymbol{y} = [y_0, y_1, \dots, y_N]^T, \qquad y_j = f(x_j), \qquad j = 0,1, \dots, N$$

1.4 Results

The experimental data was tested for polynomials up to degree **5**, as the drop of error between degrees 4 and 5 is way smaller than between 3 and 4. Here are the results of both algorithms presented as a graph and table:





Without QR

Degree	a_5	a_4	a_3	a_2	a_1	a_0
0	0	0	0	0	0	4.2811
1	0	0	0	0	2.5899	4.2811
2	0	0	0	0.8306	2.5899	-4.0253
3	0	0	0.1642	0.8306	-0.3333	-4.0253
4	0	0.0585	0.1642	-0.6318	-0.3333	0.1866
5	-0.0006	0.0585	0.1847	-0.6318	-0.4566	0.1866

With QR

Degree	a_5	a_4	a_3	a_2	a_1	a_0
0	0	0	0	0	0	4.2811
1	0	0	0	0	2.5899	4.2811
2	0	0	0	0.8306	2.5899	-4.0253
3	0	0	0.1642	0.8306	-0.3333	-4.0253
4	0	0.0585	0.1642	-0.6318	-0.3333	0.1866
5	-0.0006	0.0585	0.1847	-0.6318	-0.4566	0.1866

Difference - |QR-noQR|

Degree	a_5	a_4	a_3	a_2	a_1	a_0
0	0	0	0	0	0	0
1	0	0	0	0	4,4409e-16	0
2	0	0	0	0	4,4409e-16	8,8818e-16
3	0	0	2,7756e-17	0	2,6645e-15	8,8818e-16
4	0	1,5959e-16	2,7756e-17	3,2196e-15	2,6090e-15	1,4211e-14
5	2,4069e-17	1,5959e-16	5,5511e-16	3,2196e-15	3,6082e-15	1,4211e-14

The errors for each approximation were calculated as follows:

$$error = \left\| \sum_{i=0}^{N} |W(x_i) - y_i| \right\|$$

Where *W(x)* is the approximated polynomial and x_i and y_i are the given points.

Degree	**0**	**1**	**2**	**3**	**4**	**5**
No QR	40,479	30,013	17,573	11,924	1,1184	1,0707
QR	40,479	30,013	17,573	11,924	1,1184	1,0707
No QR – QR	0	0	0	1,7764e-15	6,6613e-16	6,6613e-16

As we can see, the method with QR factorization starts showing its advantages with the degree = 3. The advantage in this case is not too drastic, as its magnitude is around 10^{-16} (machine epsilon is equal to ~ $2.2204*10^{-16}$), which is probably because the matrix $\mathbf{A}^T\mathbf{A}$ is not badly conditioned in this set of data. With a different set of data, the difference could be much bigger.

2 DIFFERENTIAL EQUATIONS

2.1 Description of the task .....

The aim of this task is to write a MATLAB program that will determine a trajectory of a motion of a point described by following equations:

$$x_1' = x_2 + x_1(0.5 - x_1^2 - x_2^2)$$
$$x_2' = -x_1 + x_2(0.5 - x_1^2 - x_2^2)$$

This means solving a system of differential equations. The interval given is [0, 20], and the initial conditions are $x_1(0) = 10$ *and* $x_2(0) = 0$.
The results will be graphs of functions $x_1(t)$, $x_2(t)$ and $x_2(x_1)$.

2.2 Theory

Differential equations are commonly used in modeling. Systems of differential equations describing real problems are usually nonlinear, though, and classical mathematical (analytical) solutions are in most cases impossible to obtain. That is why we use numerical methods. In this problem, we will consider a system of first order ordinary differential equations (ODEs) with given initial conditions.

We will denote our independent variable as $t \in \mathbb{R}$ - which will represent time – and our dependent variables – solutions – as $x_i(t),\ i = 1, \dots, m$. The considered system can be represented as:

$$\frac{dx_i(t)}{dt} = f_i\big(t, x_1(t), \dots, x_m(t)\big), \qquad i = 1, \dots, m$$
$$t \in [a, b], \qquad x_i(a) = x_{ia}$$

Or, in a simpler vector notation:

$$\boldsymbol{x}'(t) = \boldsymbol{f}(t, \boldsymbol{x}), \qquad \boldsymbol{x}(a) = x_a, \qquad t \in [a, b]$$

Where $\boldsymbol{x} = [x_1\ x_2\ \dots\ x_m]^T \in \mathbb{R}^m, \boldsymbol{f} = [f_1\ f_2\ \dots\ f_m]^T, f_i \colon \mathbb{R}^{1+m} \to \mathbb{R}, i = 1, \dots, m$
In all following text vectors will not be explicitly denoted by bold letters, as we will assume the usage of vector notation.

The methods used will be *single-step* methods. They can be defined as a general formula (for a fixed step-size *h*):

$$x_{n+1} = x_n + h\Phi_f(t_n, x_n; h)$$

Where

$$t_n = t_0 + nh, \qquad n = 0,1, \dots \qquad x(t_0) = x_0 = x_a\ (given)$$

The function $\Phi_f(t_n, x_n; h)$ defines the method.

2.3 Algorithms used

The methods we will use in this task are three different single-step methods: the Runge-Kutta method of 4th order (RK4) with constant step-size, RK4 with variable step-size, and the Adams PC (P_5EC_5E) method with constant step-size. The results will be also compared with the built-in function *ode45().*

The family of Runge-Kutta methods can be defined as:

$$x_{n+1} = x_n + h \cdot \sum_{i=1}^{m} w_i k_i$$

Where

$$k_1 = f(t_n, x_n)$$
$$k_i = f\left(t_n + c_i h, x_n + h \cdot \sum_{j=1}^{i-1} a_{ij} k_j\right), \qquad i = 2,3,\dots,m$$
$$\sum_{j=1}^{i-1} a_{ij} = c_i, \qquad i = 2,3,\dots m$$

The most important Runge-Kutta method, and the one we will use, is the method of order 4, also called **RK4**. The equations for RK4 look as follows:

$$x_{n+1} = x_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4)$$
$$k_1 = f(t_n, x_n)$$
$$k_2 = f\left(t_n + \frac{1}{2}h, x_n + \frac{1}{2}hk_1\right)$$
$$k_3 = f\left(t_n + \frac{1}{2}h, x_n + \frac{1}{2}hk_2\right)$$
$$k_4 = f(t_n + h, x_n + hk_3)$$

A fundamental problem of using numerical methods for solving ODEs is the selection of the step-size (*h*). There are two counteracting phenomena:

- If *h* becomes smaller, the approximation error becomes smaller, but
- If *h* becomes smaller, then the number of steps needed for the solution increases, which increases the number of calculations, and therefore also the numerical errors

Therefore, the steps should be small, but not too small. Because of this, constant step-size is only suitable for functions with a similar rate of change over the whole interval.

Step doubling approach:

To estimate the approximation error, for every step of size h we perform (in parallel) two additional steps of size $h/2$. Let's denote:

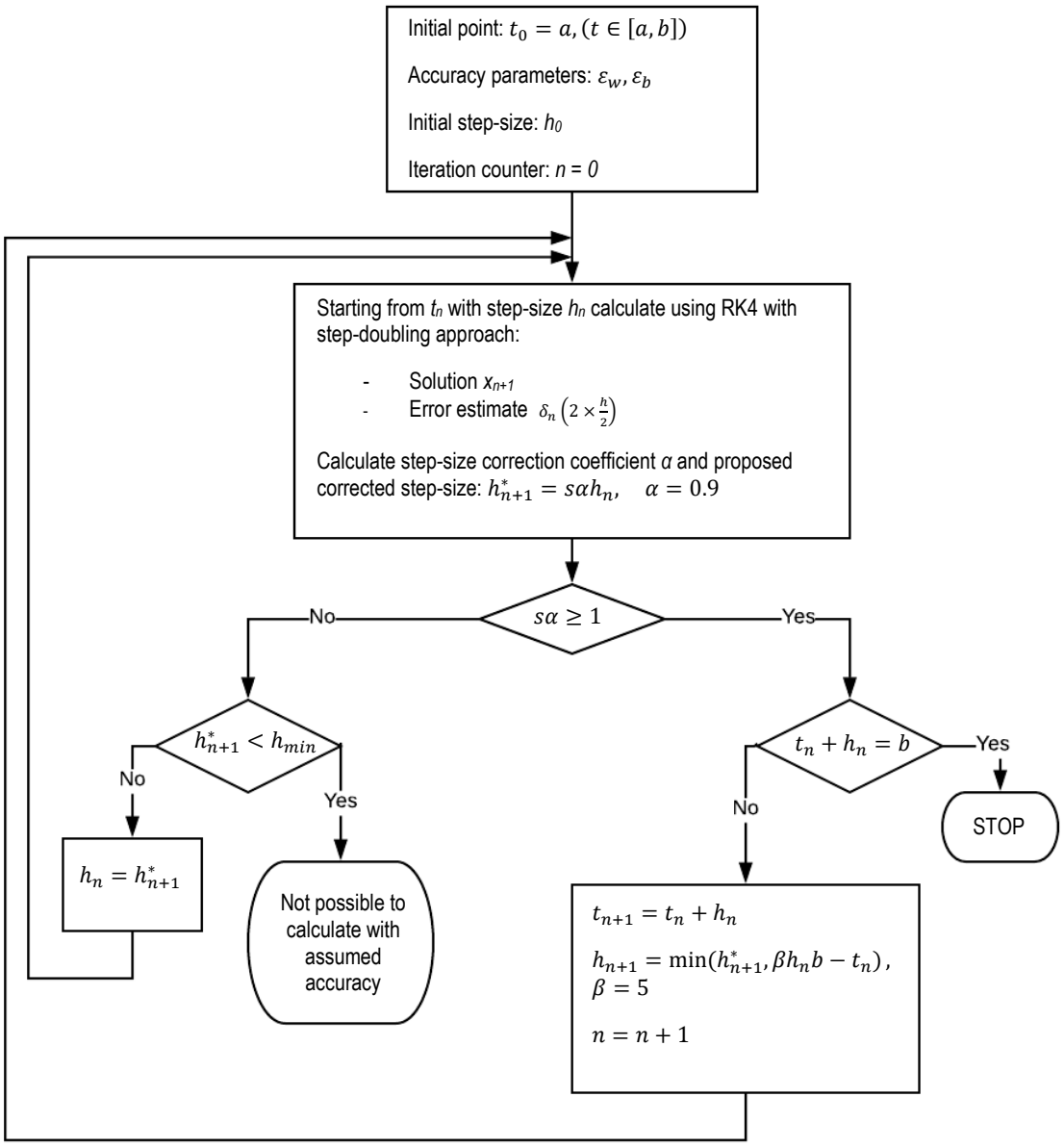
$x_n^{(1)}$- a new point obtained using full step-size
$x_n^{(2)}$- a new point obtained using two half-sized steps

After some calculations we come to those error formulas:

$$\delta_n(h) = \frac{2^p}{2^p - 1}\left(x_n^{(2)} - x_n^{(1)}\right)$$
$$\delta_n\left(2 \times \frac{h}{2}\right) = \frac{x_n^{(2)} - x_n^{(1)}}{2^p - 1}$$

From this we can see, that it is more practical to use the point obtained by using two half-sized steps, as it gives us an error estimate smaller by 2^p.

Another way to improve the RK4 algorithm is introducing variable step size. The algorithm can be represented by the following flowchart:



Where h_{min} is the assumed minimal step, ε_r is assumed relative tolerance, ε_a is the assumed absolute tolerance, and:

$$\delta_n\left(2 \times \frac{h}{2}\right) = \frac{x_n^{(2)} - x_n^{(1)}}{2^p - 1}$$

$$\varepsilon_i = \left|(x_i)_n^{(2)}\right| \cdot \varepsilon_r + \varepsilon_a$$

$$\alpha = \min_{1 \leq i \leq m} \left(\frac{\varepsilon_i}{|\delta_n(h)_i|}\right)^{\frac{1}{p+1}}$$

Adams method (P_5EC_5E)

The P_kEC_kE methods can be represented in this form:

P: $x_n^{[0]} = x_{n-1} + h\sum_{j=1}^{k}\beta_j f_{n-j}$

E: $f_n^{[0]} = f(t_n, x_n^{[0]})$

C: $x_n = x_{n-1} + h\sum_{j=1}^{k}\beta_j^* f_{n-j} + h\beta_0^* f_n^{[0]}$

E: $f_n = f(t_n, x_n)$

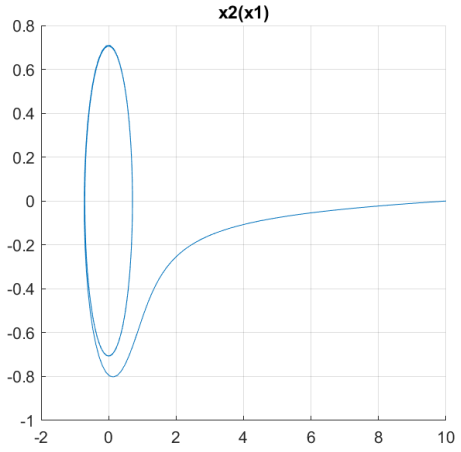
Where

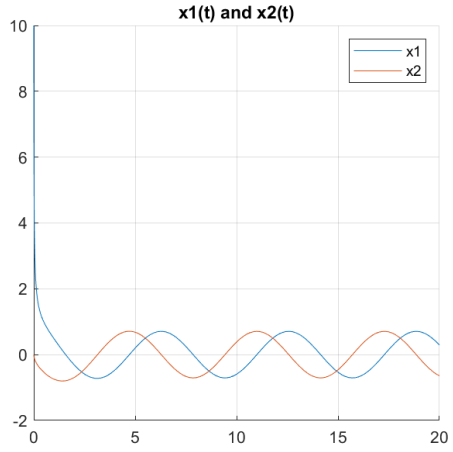
β_1	β_2	β_3	β_4	β_5
$\frac{1901}{720}$	$-\frac{2774}{720}$	$\frac{2616}{720}$	$-\frac{1274}{720}$	$\frac{251}{720}$

β_0	β_1	β_2	β_3	β_4	β_5
$\frac{475}{1440}$	$\frac{1427}{1440}$	$-\frac{798}{1440}$	$\frac{482}{1440}$	$-\frac{173}{1440}$	$\frac{27}{1440}$

2.4 Results

Firstly, to have a comparison to our results, here are the solutions generated by the function **ode45()**:

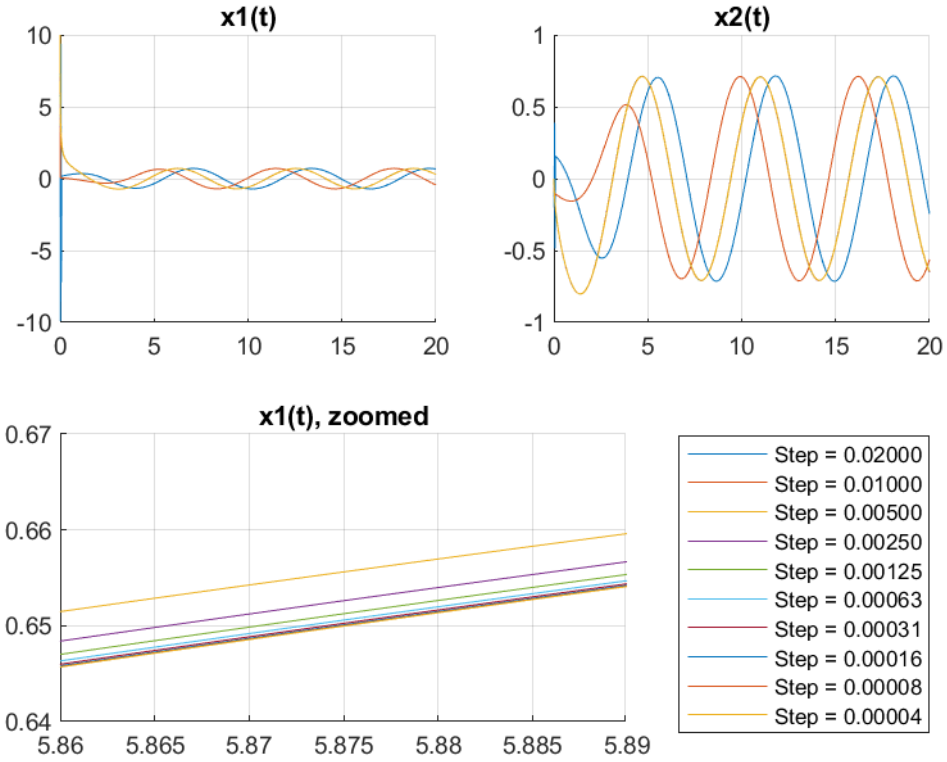




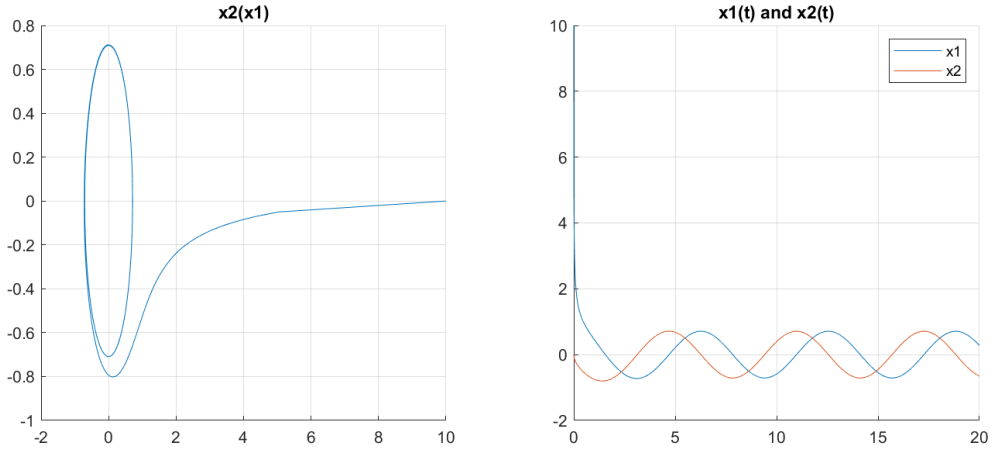
RK4 with constant step

When finding the appropriate step size, a few iterations were performed. In the first iteration $h = 0.02$ and then, for 10 iterations, it was being divided by 2. For too large step sizes the algorithm would return results close to, or equal infinity in some points, so to find a correct step-size, a smaller starting point was chosen. The chosen final step-size is **0.005**.

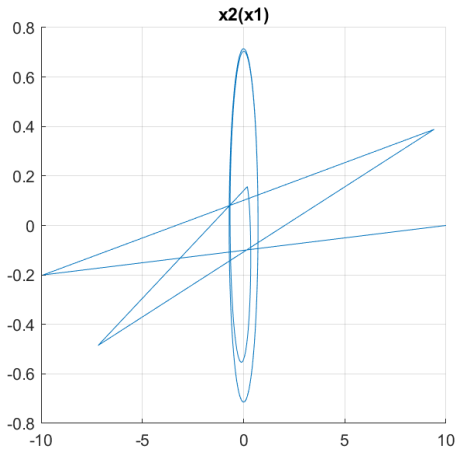
Graphs showing the search for a correct *h*:

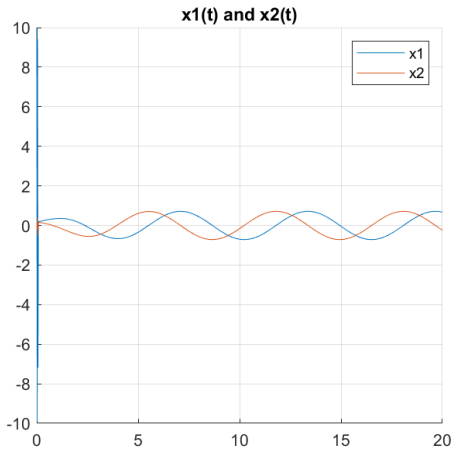


Graphs of results with step-size = 0.005:



Graphs of results with a bigger step-size(0.02):

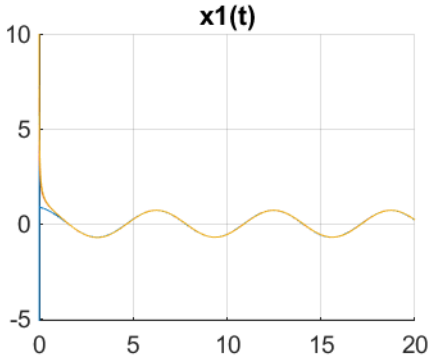


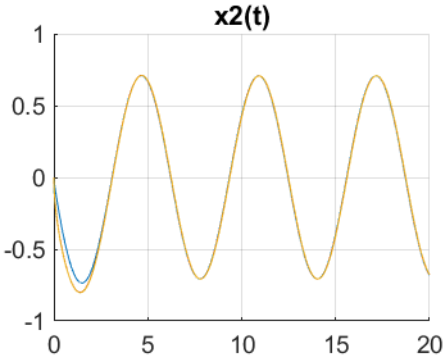


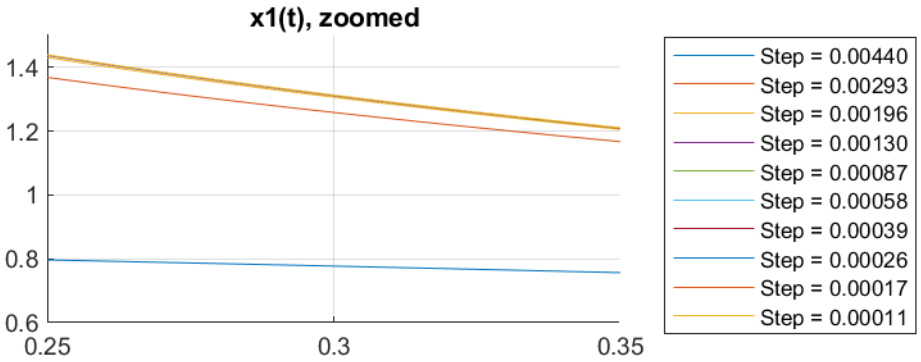
Adams Method (P_5EC_5E)

For the Adams method, the minimal step-size required for convergence was lower: 0,0044. Then, iteratively, the final chosen step-size is **0.002**.

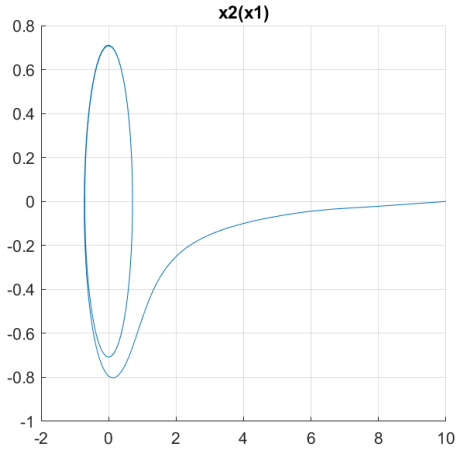
Graphs showing the search for *h:*

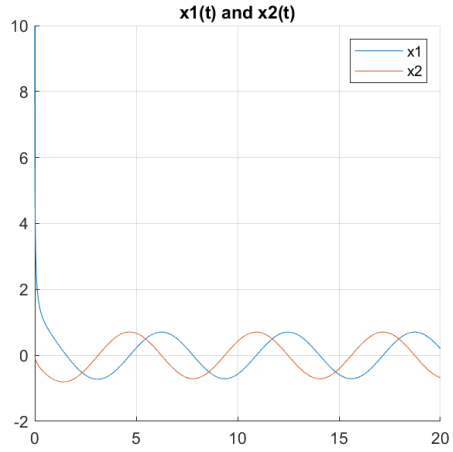




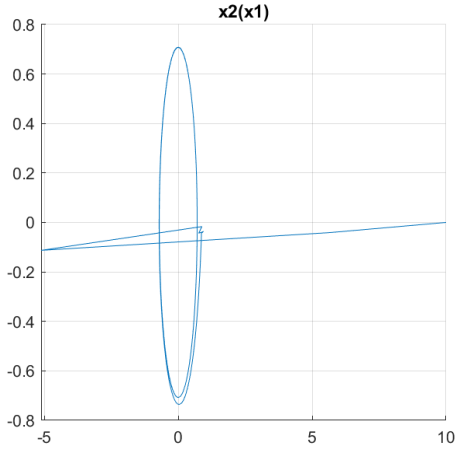


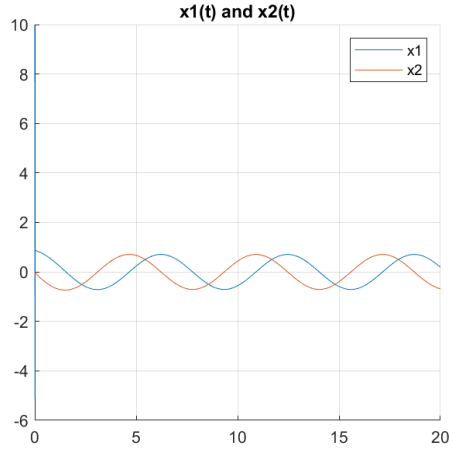
Results with step-size = 0.002:





Results with step-size = 0.0044

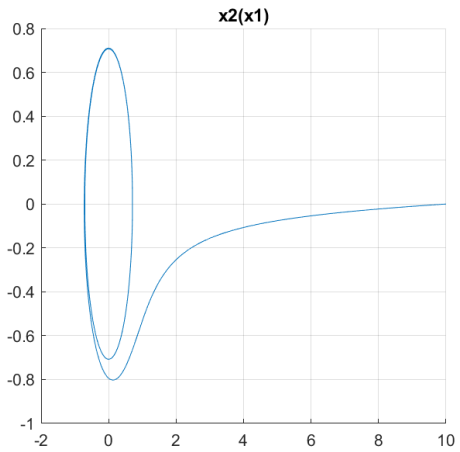


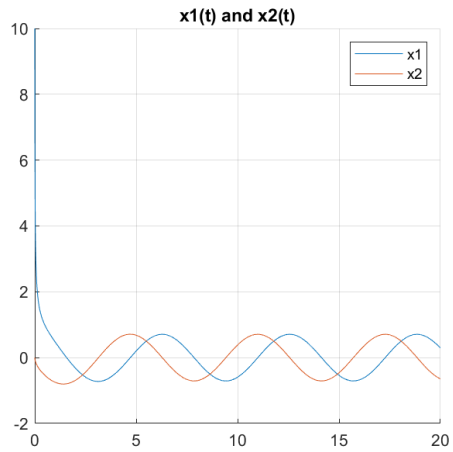


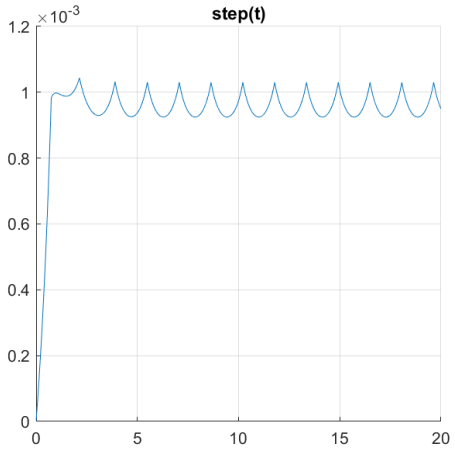
RK4 with variable step-size (step-doubling)

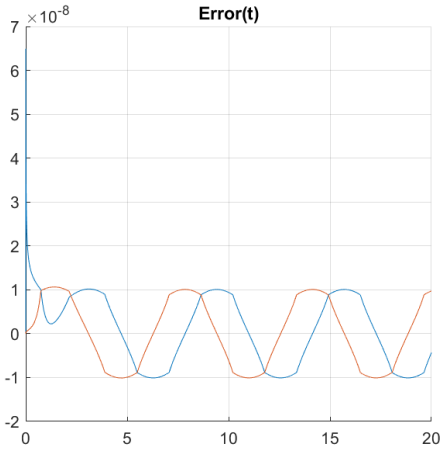
The values of h_{min}, ε_r and ε_a are input parameters of the function, to choose the precision to ones liking. In my program, in order to achieve the best precision without big calculation times, I used $h_{min} = \varepsilon_r = \varepsilon_a = 10^{-8}$. A higher precision can be chosen if needed, but it greatly increases the calculation time.

These are the graphs resulting from the RK4 method with variable step:









3 APPENDIX

dx.m

```
function [dxdt] = dx(t,x)
%UNTITLED5 Summary of this function goes here
%   Detailed explanation goes here

dxdt = [x(2)+x(1)*(0.5-x(1)^2-x(2)^2); -x(1)+x(2)*(0.5-x(1)^2-x(2)^2)];

end
```

RK4constant.m

```
function [t, x] = RK4constant(dxdt,interval,x0,h)
%RK4CONSTANT solves a system of equations using Runge-Kutta 4
%   Detailed explanation goes here

%Error handling
if (interval(2) - interval(1) > 0 && h < 0) || (interval(2) - interval(1) <
0 && h > 0)
    error("Wrong sign of step");
end

%Allocation and initial values
steps = floor(abs(interval(2) - interval(1))/abs(h));
t = zeros(1, steps+1);
t(1) = interval(1);
x(:, 1) = x0;
index = 1;

%Loop
for i = interval(1)+h:h:interval(2)
    %k1
    k(:, 1) = dxdt(t(index), x(:, index));
    %k2
    k(:, 2) = dxdt(t(index), x(:, index)+h*k(:, 1)/2);
    %k3
    k(:, 3) = dxdt(t(index), x(:, index)+h*k(:, 2)/2);
    %k4
    k(:, 4) = dxdt(t(index), x(:, index)+h*k(:, 3));
    %new x
```

```
    index = index + 1;
    t(index) = i;
    x(:, index) = x(:, index-1) + h/6*(k(:, 1)+2*k(:, 1)+2*k(:, 1)+k(:,
1));
end
```

AdamsP5EC5E.m

```
function [t, x] = AdamsP5EC5E(dxdt,interval,x0,h)
%ADAMSP5EC5E solves a system of equations using Adams Prediction-Correction
%order 5
%   Detailed explanation goes here

%Error handling
if (interval(2) - interval(1) > 0 && h < 0) || (interval(2) - interval(1) <
0 && h > 0)
    error("Wrong sign of step");
end

%Allocation and initial values
steps = floor(abs(interval(2) - interval(1))/abs(h));
t = zeros(1, steps+1);
t(1) = interval(1);
x(:, 1) = x0;
index = 1;

%Beta parameters
Bex = [1901, -2774, 2616, -1274, 251]/720;
Bim = [475, 1427, -789, 482, -173, 27]/1440;

%Loop
for i = interval(1)+h:h:interval(2)
    index = index+1;
    t(index) = i;
    %Prediction
    sum = 0;
    for j = 1:1:5
        if index - j < 1
            break
        end
        sum = sum + Bex(j)*dxdt(t(index-j), x(:, index-j));
    end
    xstep = x(:, index-1) + h*sum;
    %Corrector
    sum = 0;
    for j = 1:1:5
        if index - j < 1
            break
        end
        sum = sum + Bim(j+1)*dxdt(t(index-j), x(:, index-j));
    end
    x(:, index) = x(:, index-1) + h*sum + h*Bim(1)*dxdt(t(index-1), xstep);
end
```

task2.m

```
clear;
close all;

%Interval and starting values
interval = [0, 20];
xStart = [10, 0];

% ------------------------------------------
%
% Runge-Kutta 4 constant step - finding step
%
% ------------------------------------------
h = 0.02;
figure(1);
title("Runge-Kutta 4th order, constant step");

subplot(2,2,1);
hold on;
grid on;
title("x1(t)");

subplot(2,2,2);
hold on;
grid on;
title("x2(t)");

subplot(2,2,[3, 4]);
hold on;
grid on;
title("x1(t), zoomed");
legend('show');
legend('Location', 'eastoutside');
xlim([5.86, 5.89]);
ylim([0.64, 0.67]);

for i = 1:1:10
    [t, x] = RK4constant(@dx, interval, xStart, h);
    subplot(2,2,1);
    plot(t, x(1,:), 'DisplayName', sprintf("Step = %0.5f", h));

    subplot(2,2,[3, 4]);
    plot(t, x(1,:), 'DisplayName', sprintf("Step = %0.5f", h));

    subplot(2,2,2);
    plot(t, x(2,:), 'DisplayName', sprintf("Step = %0.5f", h));

    h = h/2;
end

saveas(1, "./plots/RK4constant.png");
saveas(1, "./plots/RK4constant.fig");

%Runge-Kutta 4 constant step - optimal step
[t, x] = RK4constant(@dx, interval, xStart, 0.005);
figure('Position', [10,10, 1000, 400]);
subplot(1,2,1);
plot(x(1, :), x(2, :));
grid on;
```

```
box off;
title("x2(x1)");

subplot(1,2,2);
hold on;
plot(t, x(1, :), 'DisplayName', 'x1');
plot(t, x(2, :), 'DisplayName', 'x2');
grid on;
box off;
title("x1(t) and x2(t)");
legend('show');

saveas(2, "./plots/RK4constantOPTIMAL.png");
saveas(2, "./plots/RK4constantOPTIMAL.fig");

%Runge-Kutta 4 constant step - bigger step
[t, x] = RK4constant(@dx, interval, xStart, 0.02);
figure('Position', [10,10, 1000, 400]);
subplot(1,2,1);
plot(x(1, :), x(2, :));
grid on;
box off;
title("x2(x1)");

subplot(1,2,2);
hold on;
plot(t, x(1, :), 'DisplayName', 'x1');
plot(t, x(2, :), 'DisplayName', 'x2');
grid on;
box off;
title("x1(t) and x2(t)");
legend('show');

saveas(3, "./plots/RK4constantBIG.png");
saveas(3, "./plots/RK4constantBIG.fig");

% ------------------------------------------
%
% Adams P5EC5E constant step - finding step
%
% ------------------------------------------
h = 0.0044;

figure(4);
title("Adams P5EC5E, constant step");

subplot(2,2,1);
hold on;
grid on;
title("x1(t)");

subplot(2,2,2);
hold on;
grid on;
title("x2(t)");

subplot(2,2,[3, 4]);
hold on;
grid on;
```

```
title("x1(t), zoomed");
legend('show');
legend('Location', 'eastoutside');
xlim([0.25, 0.35]);
ylim([0.6, 1.5]);

for i = 1:1:10
    [t, x] = AdamsP5EC5E(@dx, interval, xStart, h);
    subplot(2,2,1);
    plot(t, x(1,:), 'DisplayName', sprintf("Step = %0.5f", h));

    subplot(2,2,[3, 4]);
    plot(t, x(1,:), 'DisplayName', sprintf("Step = %0.5f", h));

    subplot(2,2,2);
    plot(t, x(2,:), 'DisplayName', sprintf("Step = %0.5f", h));

    h = h/1.5;
end

saveas(4, "./plots/AdamConstant.png");
saveas(4, "./plots/AdamConstant.fig");

%Adams PC constant step - optimal step
[t, x] = AdamsP5EC5E(@dx, interval, xStart, 0.002);
figure('Position', [10,10, 1000, 400]);
subplot(1,2,1);
plot(x(1, :), x(2, :));
grid on;
box off;
title("x2(x1)");

subplot(1,2,2);
hold on;
plot(t, x(1, :), 'DisplayName', 'x1');
plot(t, x(2, :), 'DisplayName', 'x2');
grid on;
box off;
title("x1(t) and x2(t)");
legend('show');

saveas(5, "./plots/AdamOPTIMAL.png");
saveas(5, "./plots/AdamOPTIMAL.fig");

%Adams PC constant step - bigger step
[t, x] = AdamsP5EC5E(@dx, interval, xStart, 0.0044);
figure('Position', [10,10, 1000, 400]);
subplot(1,2,1);
plot(x(1, :), x(2, :));
grid on;
box off;
title("x2(x1)");

subplot(1,2,2);
hold on;
plot(t, x(1, :), 'DisplayName', 'x1');
plot(t, x(2, :), 'DisplayName', 'x2');
grid on;
box off;
```

```
title("x1(t) and x2(t)");
legend('show');

saveas(6, "./plots/AdamBIG.png");
saveas(6, "./plots/AdamBIG.fig");

% ------------------------------------------
%
% Runge-Kutta 4 with variable step
%
% ------------------------------------------

[t, x, e, h] = RK4variable(@dx, interval, xStart, 1, 1e-8, 1e-8, 1e-8);
figure('Position', [10,10, 1000, 400]);
subplot(1,2,1);
plot(x(1, :), x(2, :));
grid on;
box off;
title("x2(x1)");

subplot(1,2,2);
hold on;
plot(t, x(1, :), 'DisplayName', 'x1');
plot(t, x(2, :), 'DisplayName', 'x2');
grid on;
box off;
title("x1(t) and x2(t)");
legend('show');

saveas(7, "./plots/RK4variable.png");
saveas(7, "./plots/RK4variable.fig");

figure('Position', [10,10, 1000, 400]);
subplot(1,2,1);
plot(t, h);
grid on;
box off;
title("step(t)");

subplot(1,2,2);
hold on;
plot(t, e);
grid on;
box off;
title("Error(t)");

saveas(8, "./plots/RK4variableSE.png");
saveas(8, "./plots/RK4variableSE.fig");

% ------------------------------------------
%
% ode45
%
% ------------------------------------------

[t, x] = ode45(@dx, interval, xStart);
figure('Position', [10,10, 1000, 400]);
subplot(1,2,1);
plot(x(:, 1), x(:, 2));
```

```
grid on;
box off;
title("x2(x1)");

subplot(1,2,2);
hold on;
plot(t, x(:, 1), 'DisplayName', 'x1');
plot(t, x(:, 2), 'DisplayName', 'x2');
grid on;
box off;
title("x1(t) and x2(t)");
legend('show');

saveas(9, "./plots/ode45.png");
saveas(9, "./plots/ode45.fig");
```