Numerical Methods

Project C No. 61

Jakub Tomkiewicz
300183

Problem 1

Description of the problem

For the following experimental measurements:

x_i	y_i
-5	-36.4986
-4	-20.1300
-3	-10.4370
-2	-3.8635
-1	-1.3503
0	0.8879
1	0.8176
2	1.4830
3	4.4024
4	10.3910
5	21.3003

We are going to determine a polynomial function $y = f(x)$ that best fits the experimental data by using the least-squares approximation. Graphs of obtained functions along with experimental data will be presented. QR factorisation of matrix $\mathbf{A}$ needs to be used in order to solve the least-squares problem. For each solution, calculation of error defined as the Euclidean norm of the vector of residuum and the condition number of the Gram's matrix will be made. Results needs to be compared in terms of solutions' errors.

Theoretical grounds

Discrete least-squares approximation

In the following formula:

$$\forall F \in X_n \quad F(x) = \sum_{i=0}^{N} a_i \Phi_i(x)$$

we are looking for a values of parameters $a_0, a_1, \ldots, a_n$. To find them system of linear equations, called a set of normal equations, needs to be solved. It's matrix is called the Gram's matrix.

We can write normal equations in a simple form, as a scalar product:

$$\langle \Phi_i \ , \ \Phi_k \rangle \stackrel{df}{=} \sum_{j=0}^{N} \Phi_i(x_j) \ \Phi_k(x_j)$$

Set of normal equations has following form:

$$\begin{bmatrix} \langle \Phi_0 \ , \ \Phi_0 \rangle & \langle \Phi_1 \ , \ \Phi_0 \rangle & \dots & \langle \Phi_n \ , \ \Phi_0 \rangle \\ \langle \Phi_0 \ , \ \Phi_1 \rangle & \langle \Phi_1 \ , \ \Phi_1 \rangle & \dots & \langle \Phi_n \ , \ \Phi_1 \rangle \\ \dots & \dots & \dots & \dots \\ \langle \Phi_0 \ , \ \Phi_n \rangle & \langle \Phi_1 \ , \ \Phi_n \rangle & \dots & \langle \Phi_n \ , \ \Phi_n \rangle \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \dots \\ a_n \end{bmatrix} = \begin{bmatrix} \langle \Phi_0 \ , \ f \rangle \\ \langle \Phi_1 \ , \ f \rangle \\ \dots \\ \langle \Phi_n \ , \ f \rangle \end{bmatrix}$$

It is recommended to solve the approximation problem using the method based on QR factorisation, as matrix composed of the set of equations can be badly conditioned.

If the QR factorisation is evaluated to solve the linear least-squares problem, then the narrow, not normalised factorisation is used.

Polynomial approximation

Polynomials are often used as a approximating functions.

Order of the polynomial n is usually much lower then the number of points at which the values of the original function are given:

$$N \gg n$$

Natural polynomial basis (the power basis) is considered as:

$$\Phi_0(x) = 1, \quad \Phi_1(x) = x, \quad \Phi_2(x) = x^2, \ \dots \ , \Phi_n(x) = x^n$$

In more familiar form, polynomial looks as following:

$$F(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n$$

However, Gram's matrix for the system of normal equations obtained for the approximating polynomials with the natural basis quickly becomes ill-conditioned with the increase of n (it is fine if n does not exceed 4).

System of normal equations can be presented in the following form:

$$G \cdot a = \varrho$$

where a is a set of unknown parameters $a_0, a_1, \ldots, a_n$.

Having auxiliary definitions, we can create G (Gram's matrix) out of g_{ik} elements using:

$$g_{ik} \stackrel{df}{=} \sum_{j=0}^{N} (x_j)^{i+k}$$

and ϱ (vector of elements) with equation:

$$\varrho_k \stackrel{df}{=} \sum_{j=0}^{N} f(x_j)(x_j)^k$$

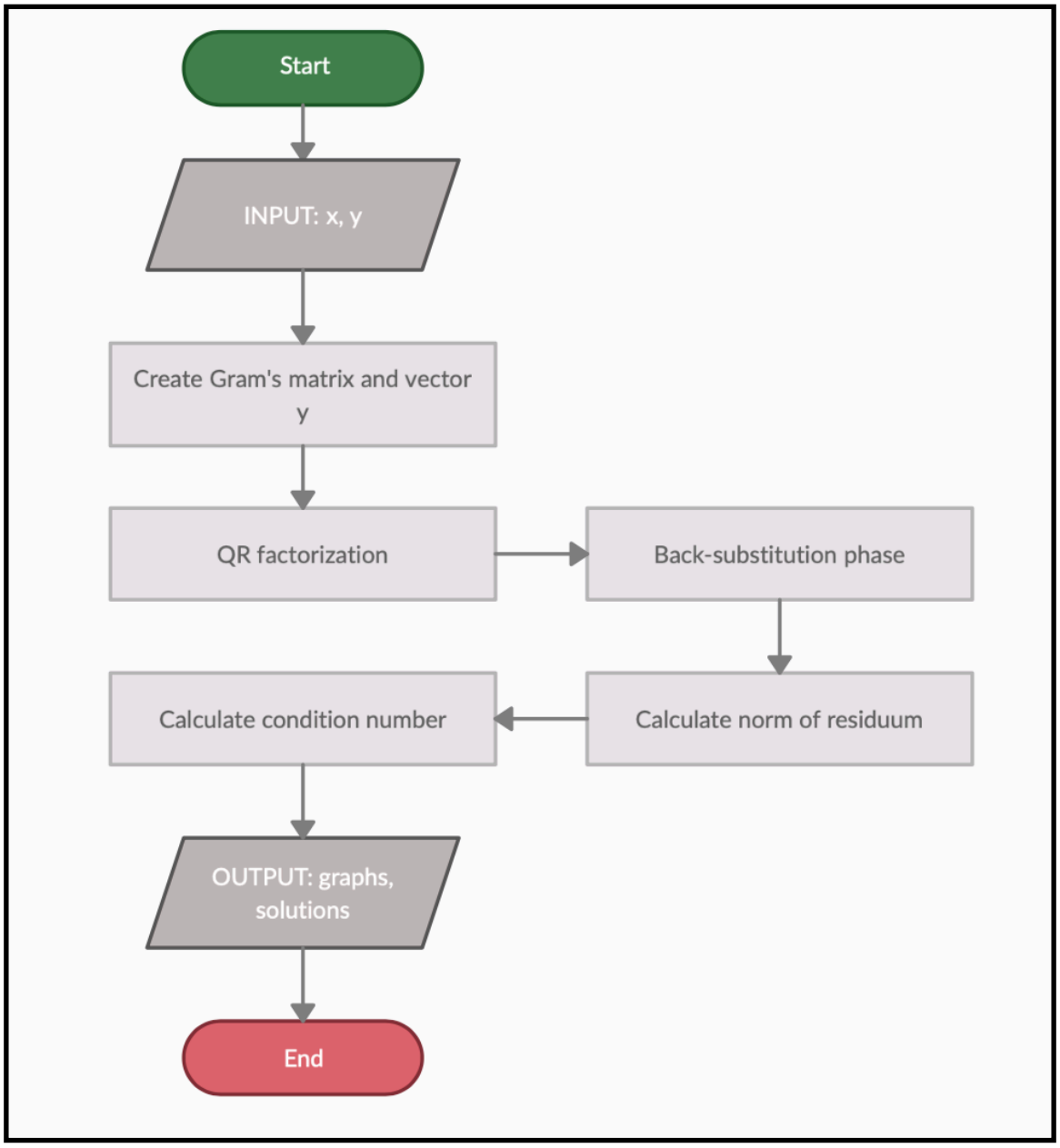
Problem analysis

In this task we are able to use knowledge gained during Assignment A, as we have already acquainted with QR factorisation and solving equations.

The first step is to create Gram's matrix and vector having x and y. As mentioned in theoretical grounds it will quickly become ill-conditioned if we use approximation with natural basis so QR factorisation needs to be applied. To create polynomial, parameters must be found. To obtain them back-substitution will be practiced. At the end Euclidean norm of the vector of residuum and condition number are going to be determined.

The best way to test polynomials of various degrees is to use functions that will take degree as parameter, this way code will be simpler and more resistant to errors.

Algorithm applied



Solution

Polynomials of degrees 2,4,6,8 and 10 were plotted. Below presented are figures with marked experimental data.

After figures, comparison of results for all degrees is presented in table.

Figure for $y = -0.3149x^2 + 4.2587x + 0.1492$

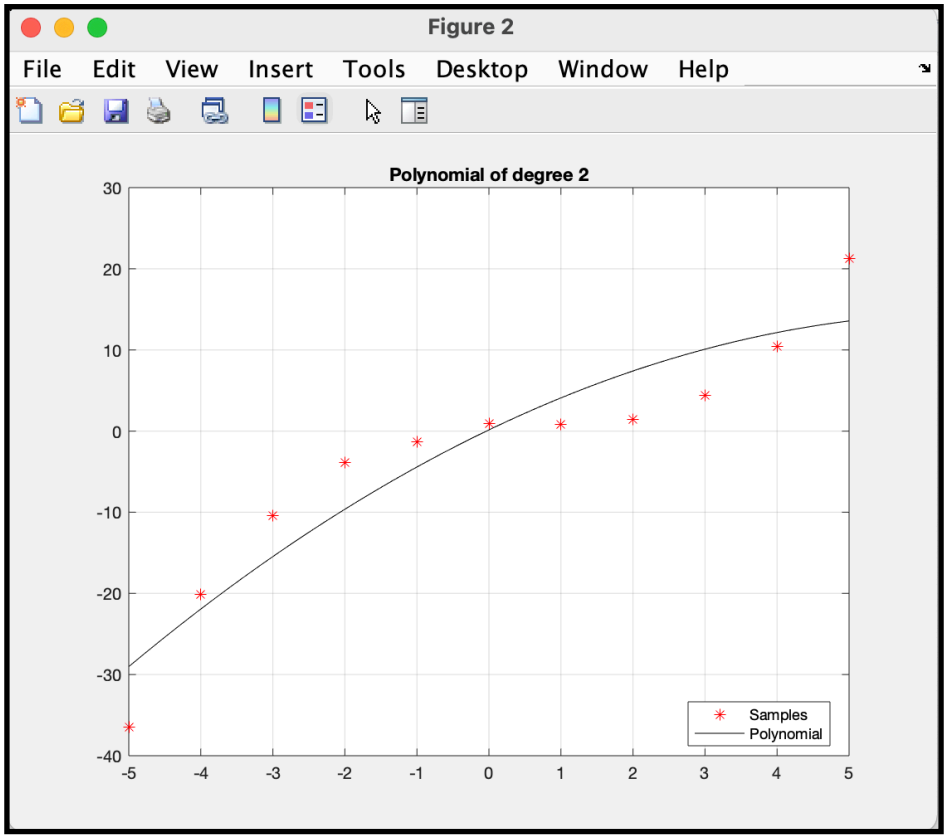


Figure for $y = 0.0026x^4 + 0.2081x^3 - 0.3807x^2 + 0.5554x + 0.3386$

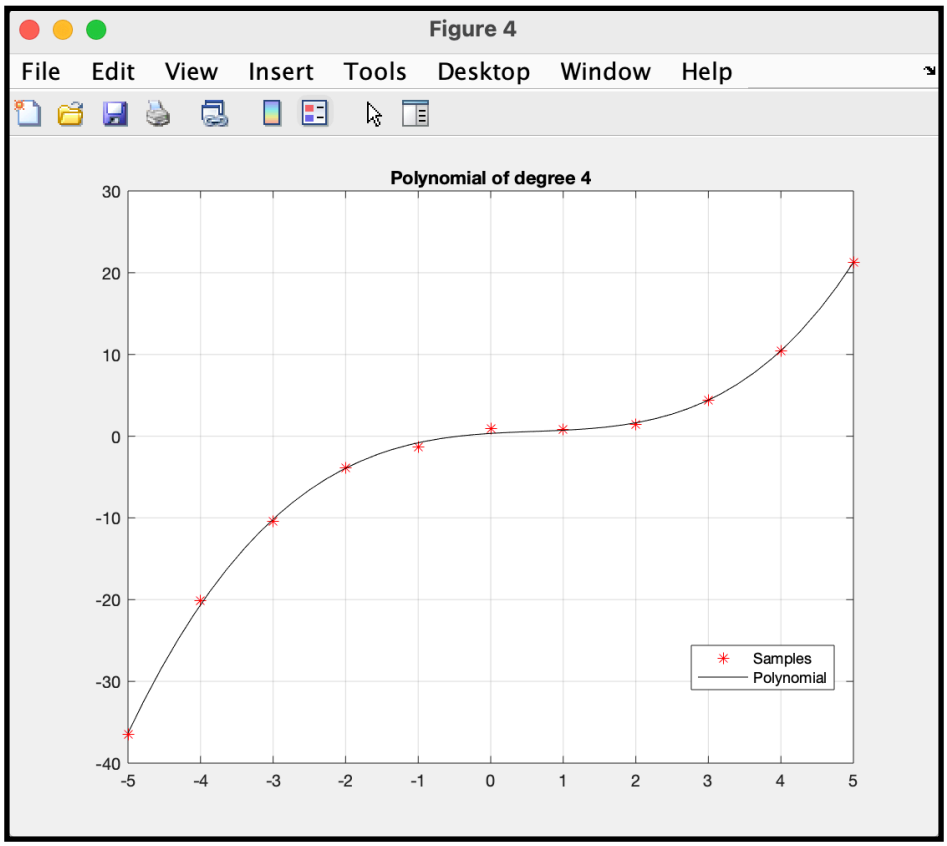


Figure for

$$y = -0.0004x^6 + 0.0009x^5 + 0.0181x^4 + 0.1802x^3 - 0.5214x^2 + 0.7231x + 0.5180$$

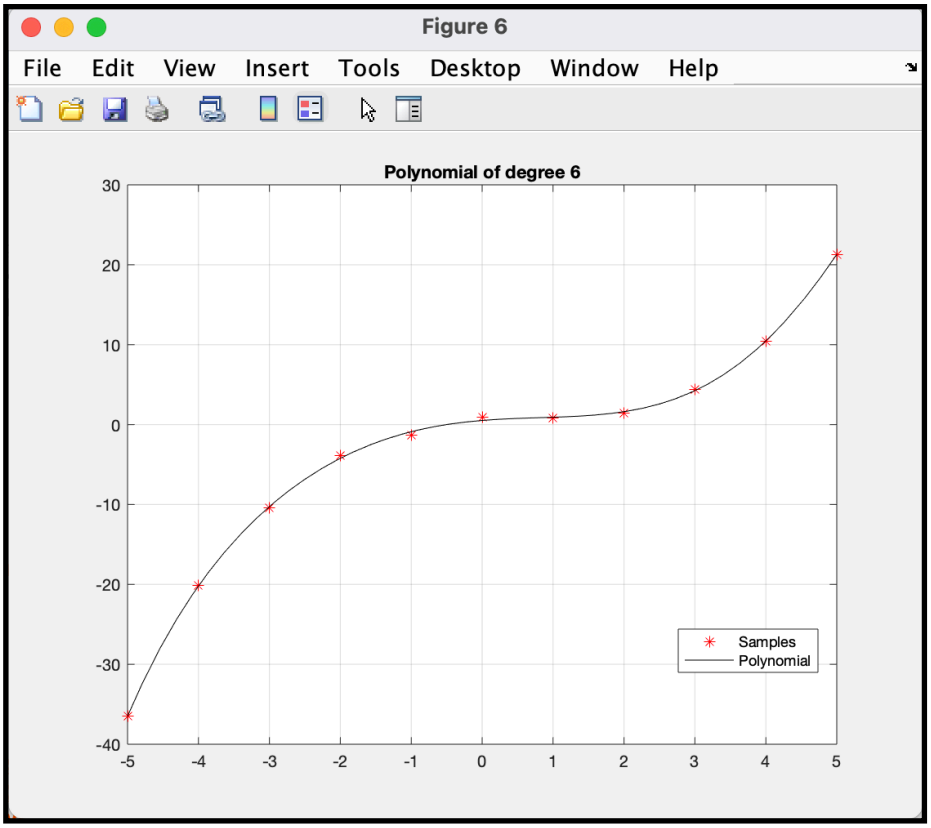


Figure for

$$y = -0.0011x^6 - 0.0005x^5 + 0.0275x^4 + 0.1961x^3 - 0.5579x^2 + 0.6777x + 0.5299$$

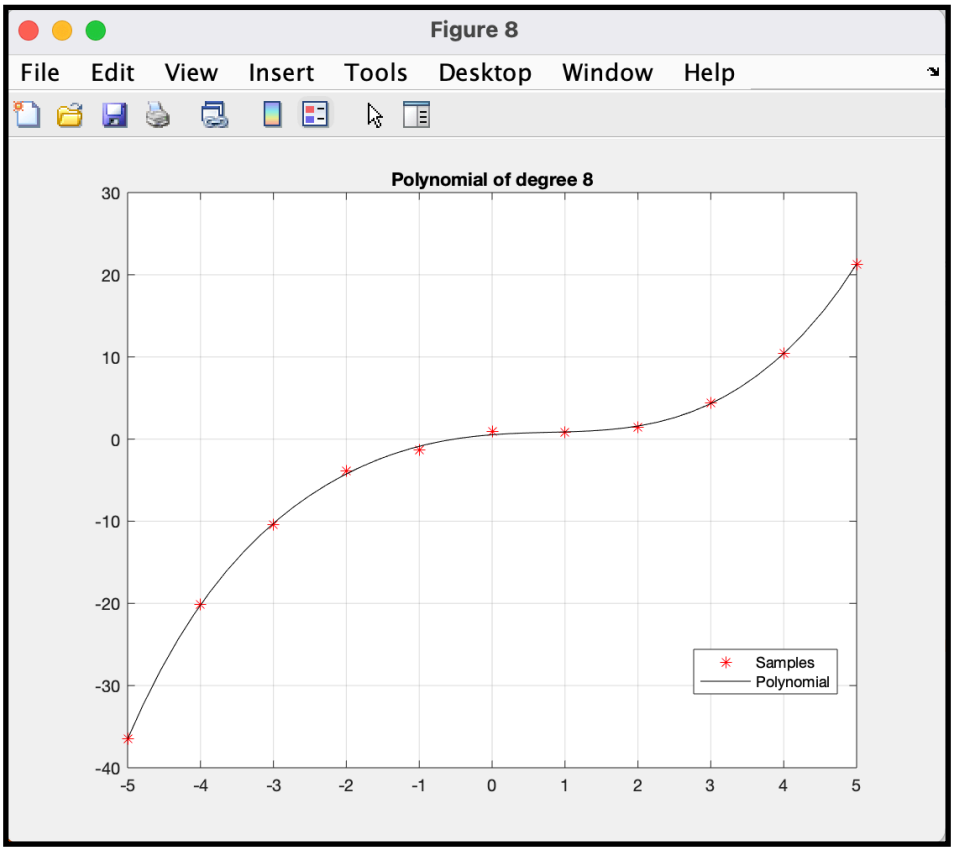
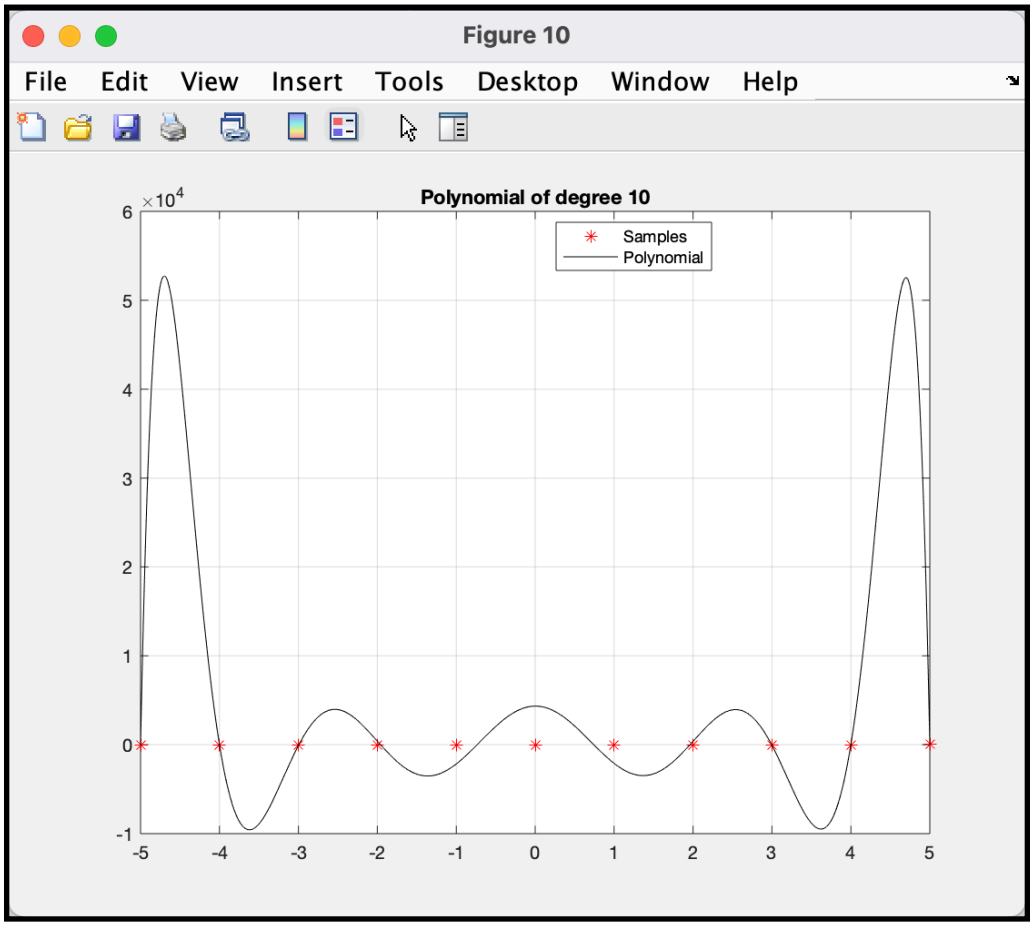


Figure for
$y = -0.0001x^{10} + 0.0031x^{8} - 0.0570x^{6} + 0.0005x^{5} + 0.4071x^{4} - 0.0032x^{3} - 1.0008x^{2} + 0.0060x + 0.4343$



Comparison of errors and condition numbers

Polynomial degree	Error	Condition number
1	2433.1368	9.2929e14
2	34.4547	1.5167e13
3	0.0441	3.3054e11
4	0.0003	7.6462e9
5	1.2723e-05	2.8315e8
6	4.0142e-07	7.4674e6
7	1.4396e-09	3.1798e5
8	4.0459e-11	8558.44
9	4.6534e-13	408.78
10	5.6843e-14	10

Conclusions

From the plotted figures we could say that polynomials of orders 2 and 10 are worse then others. Length between samples and polynomial function is easily noticeable, especially when degree is 10. Polynomials of orders 4,6 and 8 fit much better, as the distances are small. Without precise data it would be difficult to choose the best order.

However, the comparison of errors and condition numbers for all polynomial degrees can clear up any doubts. Bigger polynomial degree results in smaller error and smaller condition number. This of course make sense, polynomials of bigger degree have more parameters to adapt, therefore the function plot can better fit our 11 samples.

Problem 2

Description of the problem

A motion of a point is given by the following equations:

$$\frac{dx_1}{dt} = x_2 + x_1\left(0.5 - x_1^2 - x_2^2\right)$$

$$\frac{dx_2}{dt} = -x_1 + x_2\left(0.5 - x_1^2 - x_2^2\right)$$

Determine the trajectory of the motion on the interval [0,20] fo initial conditions:

$$x_1(0) = -0.4$$

$$x_2(0) = -0.3$$

Solutions needs to be evaluated in two methods:

a) Runge-Kutta method of 4^{th} order (RK4) and Adams PC $\left(P_5EC_5E\right)$ - 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 bit 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

Results will be compared with ones obtained using an ODE solver, e.g. ode45.

Theoretical grounds

Ordinary differential equations

They are commonly used for mathematical modelling of dynamic systems. Describing real-life problems are usually nonlinear and the only way to calculate it's solutions is using numerical methods.

If for function f two following conditions are satisfied:

- is continuous over the set

$$D = \left\{(x, y) : \quad a \leq x \leq b, \quad y \in R^m\right\}$$

- satisfies the Lipschitz condition with respect to y

$$\exists L > 0 \quad \forall x \in [a, b], \quad \forall y, \bar{y} \quad \left|\left|f(x, y) - f(x, \bar{y})\right|\right| \leq L\left|\left|y - \bar{y}\right|\right|$$

Then for any initial condition y_a there is a unique and continuously differentiable function $y(x)$ such that

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

Moreover, it can bee shown that under the assumptions of the above theorem, solution $y(x)$ depends in a continuous way on problem perturbations.

A numerical method for a system of differential equations is convergent, if for any system having a unique solution $y(x)$ and for the step length h_n converging to zero for all steps, the following condition holds

$$\lim_{h_n \to 0} y(x_n; h_n) \to y(x)$$

where $y(x_n; h_n)$ denotes the approximate the solution found by the method.

The difference between single-step and multi-step methods are that single-step methods refer to only one previous point and its derivative to determine the current value. Multi-step methods however, refers to informations gained from previous steps.

Runge-Kutta (RK) methods

Family of Runge-Kutta methods can be defined by following formula:

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

where

$$k_1 = f(x_n, y_n),$$
$$k_i = f(x_n + c_i h, y_n + h \cdot \sum_{j=1}^{i-1} a_{ij} k_j), \quad i = 2,3,...,m,$$

and also

$$\sum_{j=1}^{i-1} a_{ij} = c_i, \quad i = 2,3,...,m$$

To perform a single step of the method the values of the right-hand sides of the equation must be calculated precisely m times, therefore we can describe the method as m stage one. Usually, coefficients are chosen in a way assuring a high order of the method, for a given m.

Methods with $m = 4$ and of the order $p = 4$ are most important in practice as they constitute a good tradeoff between approximation accuracy and number of arithmetic operations performed at one iteration.

The RK method of order 4 (RK4)

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

We have four approximate values of the solution derivative over the one step interval: one at initial point, two at midpoint and one at endpoint. Final approximation of the solution derivative for the final step of method is calculated as a weighted mean value of these derivatives, with weight 1 for initial and end points and weight 2 for the midpoint.

Predictor-corrector methods

For the Adams methods the $P_k EC_k E$ algorithm has the following form:

$$P: \quad y_n^{[0]} = y_{n-1} + h\sum_{j=1}^{k} \beta_j f_{n-j} \qquad P \text{ - prediction}$$

$$E: \quad f_n^{[0]} = f(x_n, y_n^{[0]}) \qquad E \text{ - evaluation}$$

$$C: \quad y_n = y_{n-1} + h\sum_{j=1}^{k} \beta_j^* f_{n-j} + h\beta_0^* f_n^{[0]} \qquad C \text{ - correction}$$

$$E: \quad f_n = f(x_n, y_n) \qquad E \text{ - evaluation}$$

Step-size selection

Appropriate procedure for a selection/correction of the step-size h_n is a fundamental problem for a practical implementation of numerical methods. There are two counteracting phenomena here:

- if the step h_n becomes smaller, then the approximation error of the method becomes smaller
- if the step h_n becomes smaller, then also the number of steps needed to find a solution on a given interval increases, so more arithmetic solutions are needed to find a solution

Calculations with too small steps are not recommended as the step should be sufficiently small to perform calculations with a desired accuracy, not smaller then necessary.

It would be better when an initial step-size given by the user was adjusted automatically by the method. But, for problems having solutions with a rate of change differing significantly over the interval, step-size should be a variable that can be adjusted automatically at each step by the method.

Fundamental information necessary to perform automatic step-size adjustment is an estimate of approximation error occurring at each step of method.

The step-doubling approach

To find the approximation error for every step of the size h two additional steps of size $\frac{h}{2}$ are performed additionally in parallel.

We need to note 4 variables:

$y_n^{(1)}$ - new point obtained using step size $\frac{h}{2}$

$y_n^{(2)}$ - new point obtained using two consecutive steps of size $\frac{h}{2}$

$r^{(1)}$ - approximation error after the single step

$r^{(2)}$ - summed approximation errors after two smaller steps

After several manipulations we have:

$$y(x_n + h) = y_n^{(2)} + \frac{y_n^{(2)} - y_n^{(1)}}{2^p - 1} + O(h^{p+2})$$

$$y(x_n + h) = y_n^{(1)} + 2^p \frac{y_n^{(2)} - y_n^{(1)}}{2^p - 1} + O(h^{p+2})$$

Error estimate for a step-size h follows:

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

Error estimate of the two consecutive steps of size $\frac{h}{2}$:

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

Problem analysis

As this is probably the most difficult task of all the problems, it is the best to do divide it into two parts (a and b) as in description of the problem.

First part is simpler, as we manually adjust step-size to find the best one. Two functions are needed here, one to calculate RK4 and one for Adam's method P5EC5E. Both needs to take 3 parameters: step-size value, beginning of interval and ending of interval. As we take into account problem solutions versus time, functions will return not only solutions but also the time interval. Each method will be performed few times, until the best optimal step-size will be found.

Second part will be directly connected to RK4 from part one. This time however, it will be automatised, so there will be no need to perform it several times. New function will be using moving initial point and adjusting step-size, it's structured will be based on the block diagram from the page 174. Knowledge of error estimation and correction of the step size is essential.

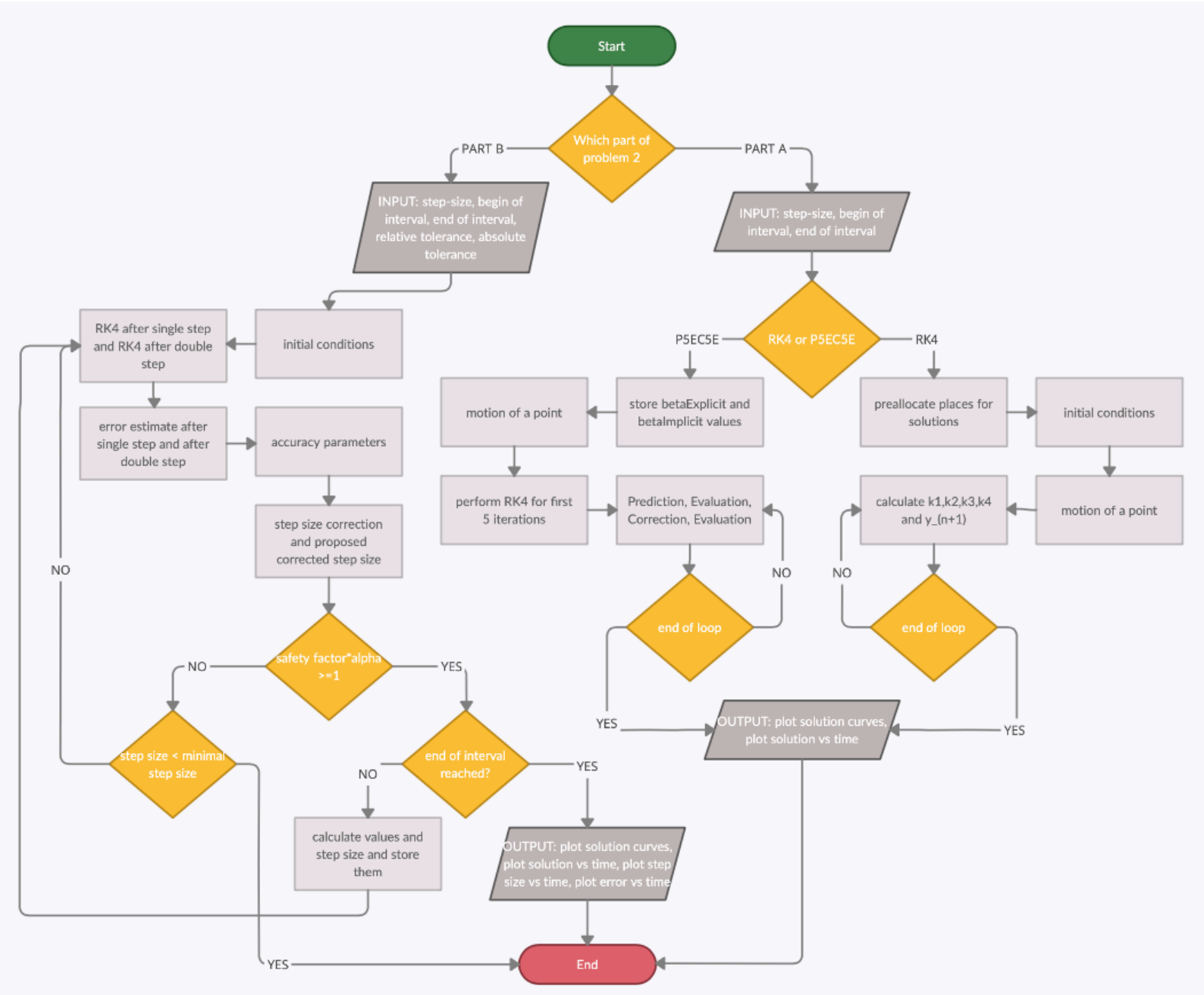
To keep the code simple plotting figures will be done in functions, divided accordingly to what we want to show.

Values of tolerance are set to 1e-9, as in previous assignments A and B.

All results will be compared with MATLAB build in ODE solver.

Algorithm applied

Right part of the block diagram corresponds to the part a) of the problem 2, left part corresponds to the part b). Diagram has been simplified in order to present used algorithms and its structure in more user friendly form, MATLAB code is much more complicated.



Solution

PART A: Results achieved for the RK method of order 4 (RK4)

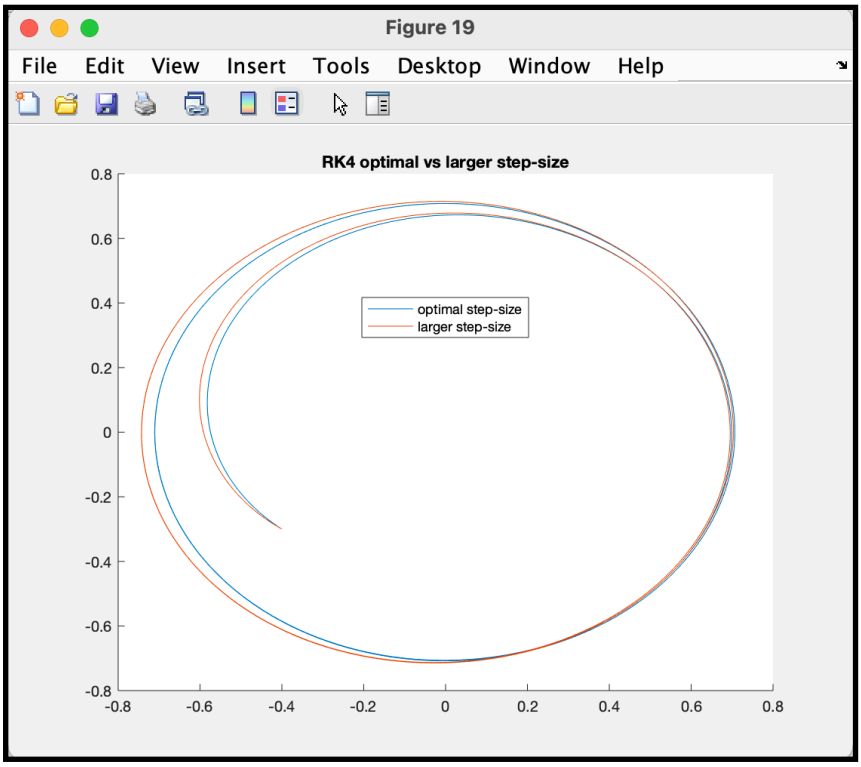


Fig. Step size comparison. Optimal step-size=0.004, Larger step-size=0.04

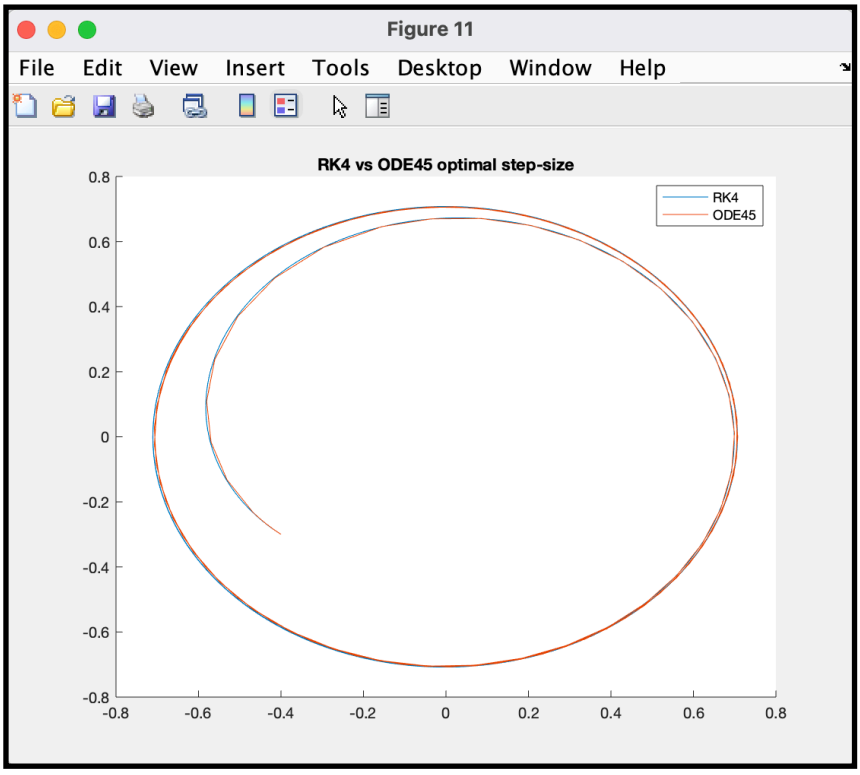


Fig. Optimal step-size=0.004 vs ODE45 solver.

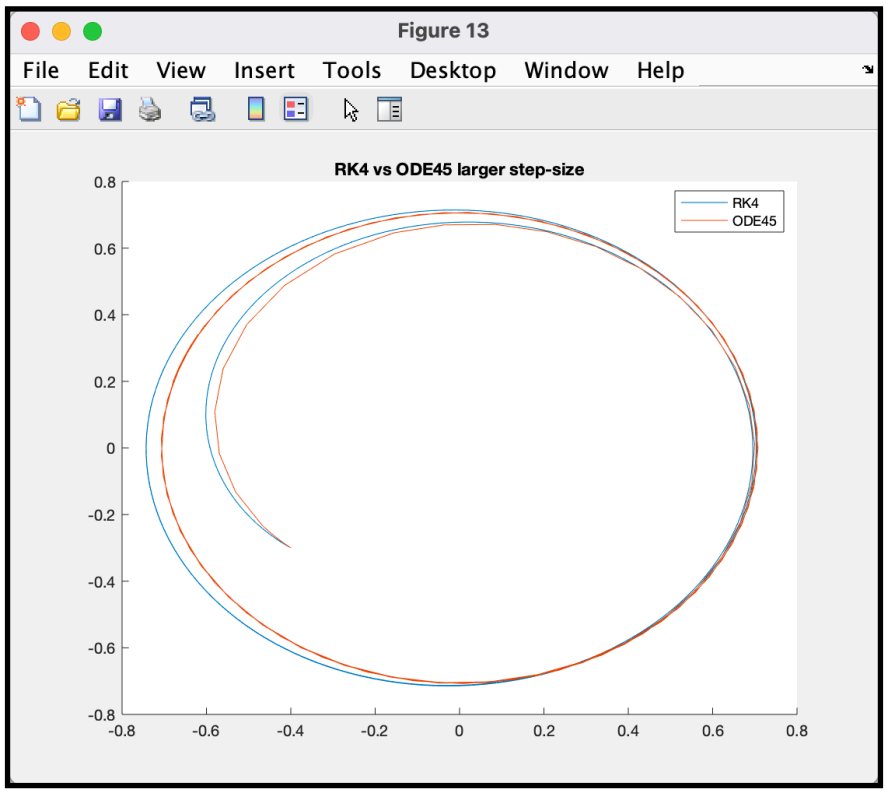


Fig. Larger step-size=0.04 vs ODE45 solver.

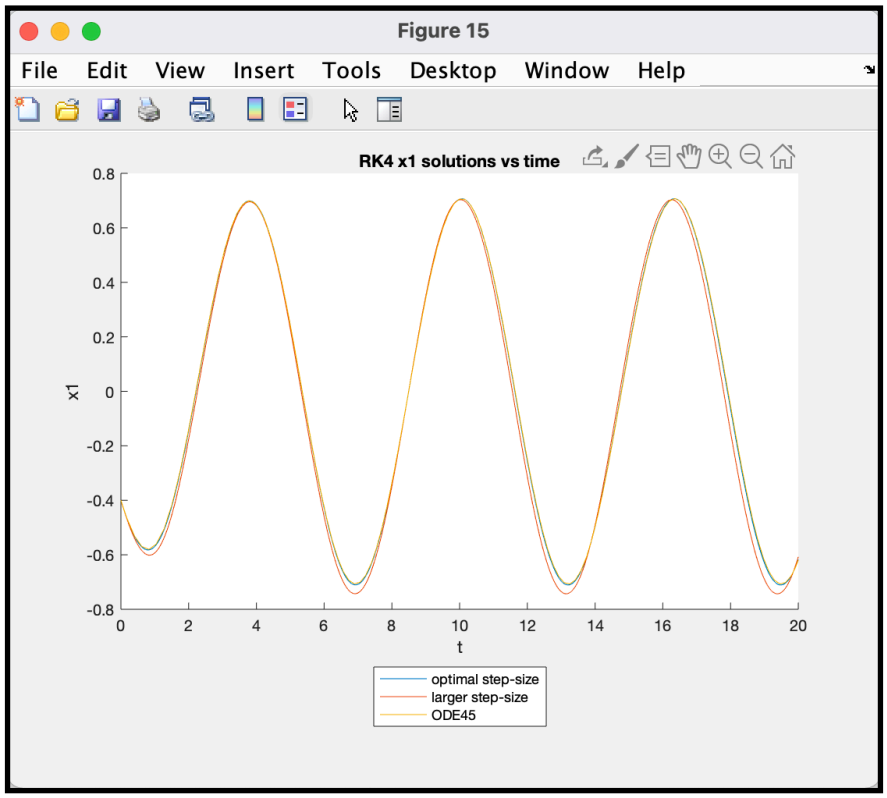


Fig. Solutions vs time comparison. Optimal step-size=0.004, Larger step-size=0.04

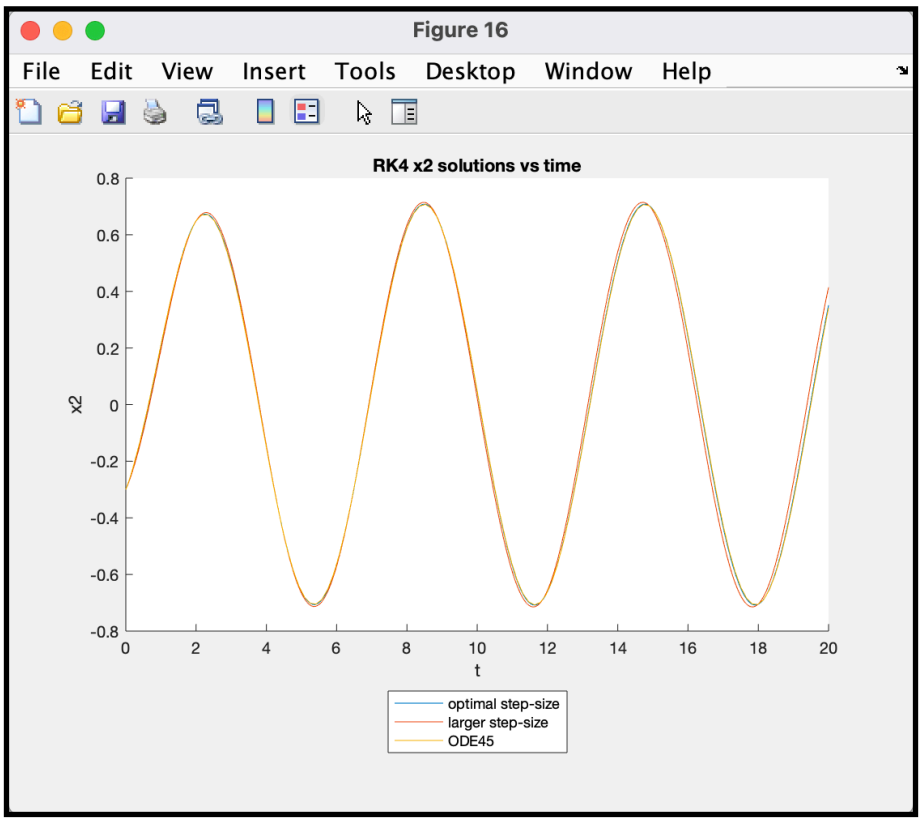


Fig. Solutions vs time comparison. Optimal step-size=0.004, Larger step-size=0.04

PART A: Results achieved for Adams method P5EC5E

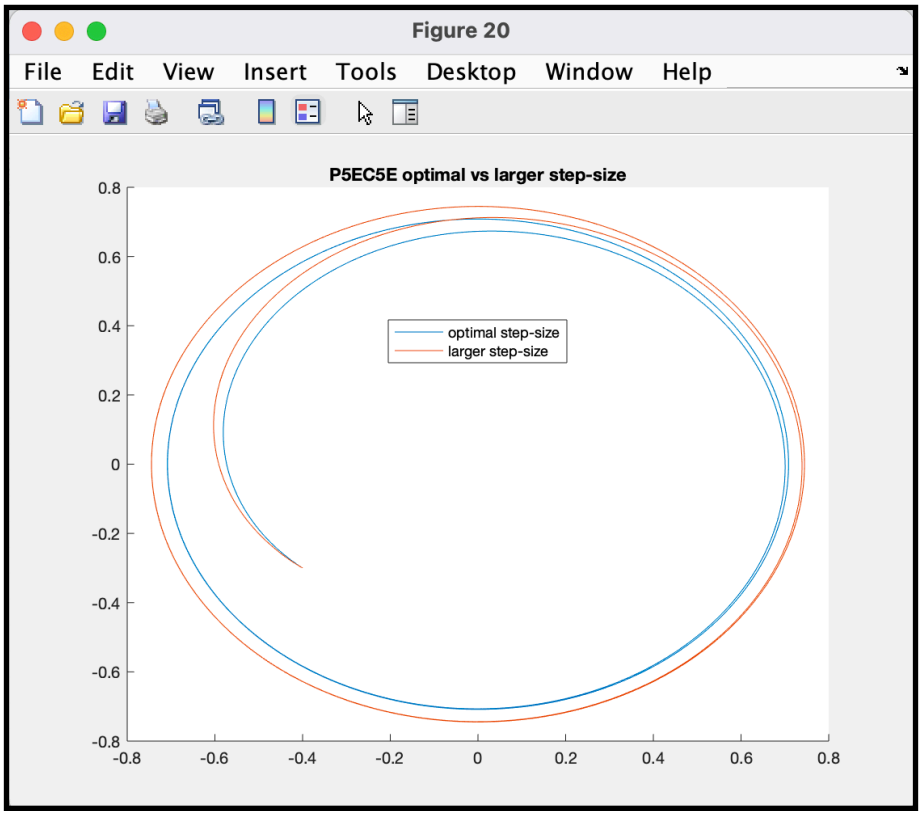


Fig. Step size comparison. Optimal step-size=0.001, Larger step-size=0.03

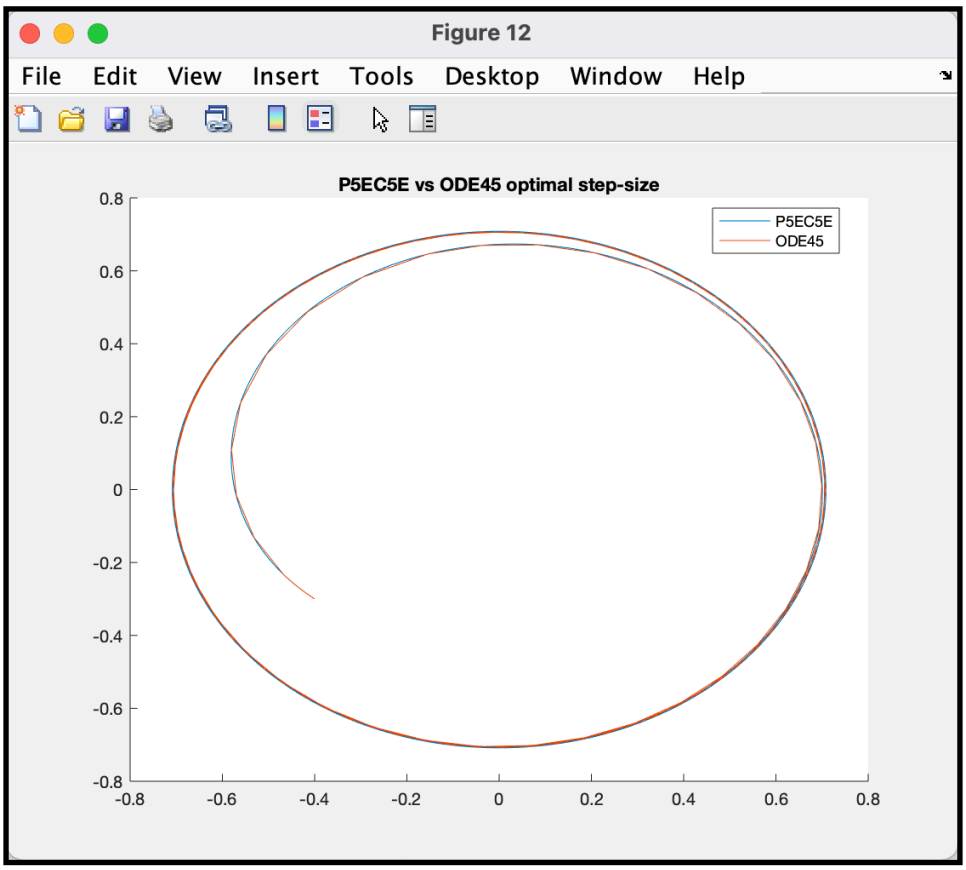


Fig. Optimal step-size=0.001 vs ODE45 solver.

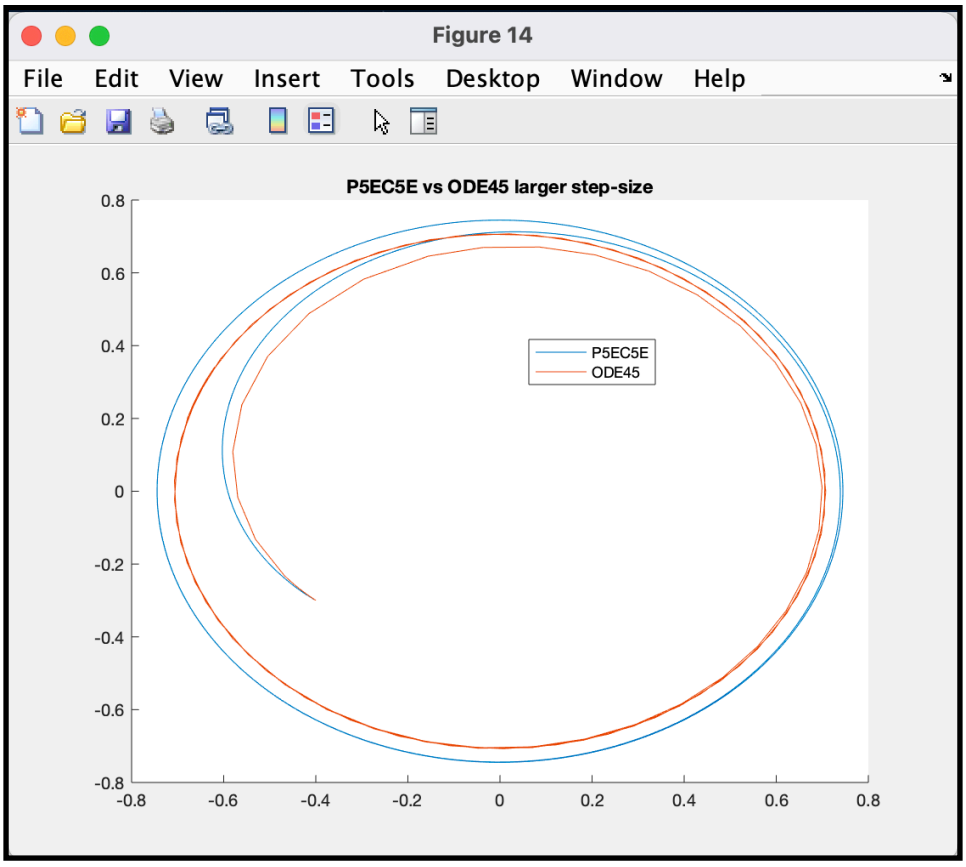


Fig. Larger step-size=0.03 vs ODE45 solver.

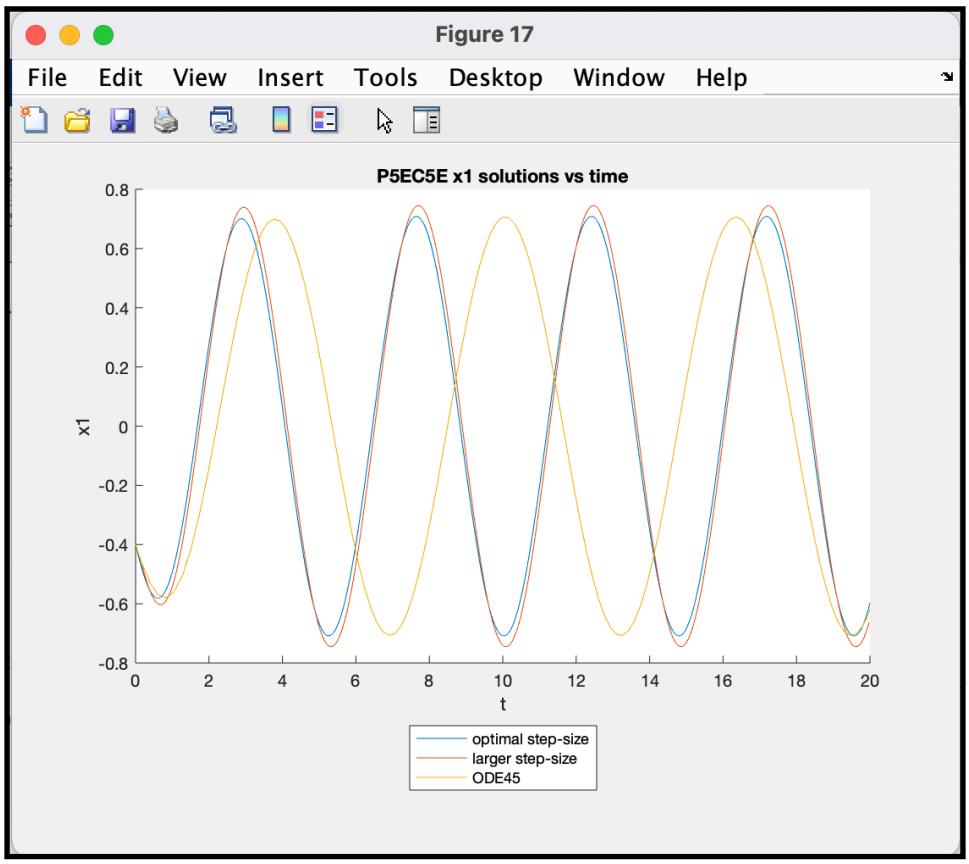


Fig. Solutions vs time comparison. Optimal step-size=0.001, Larger step-size=0.03

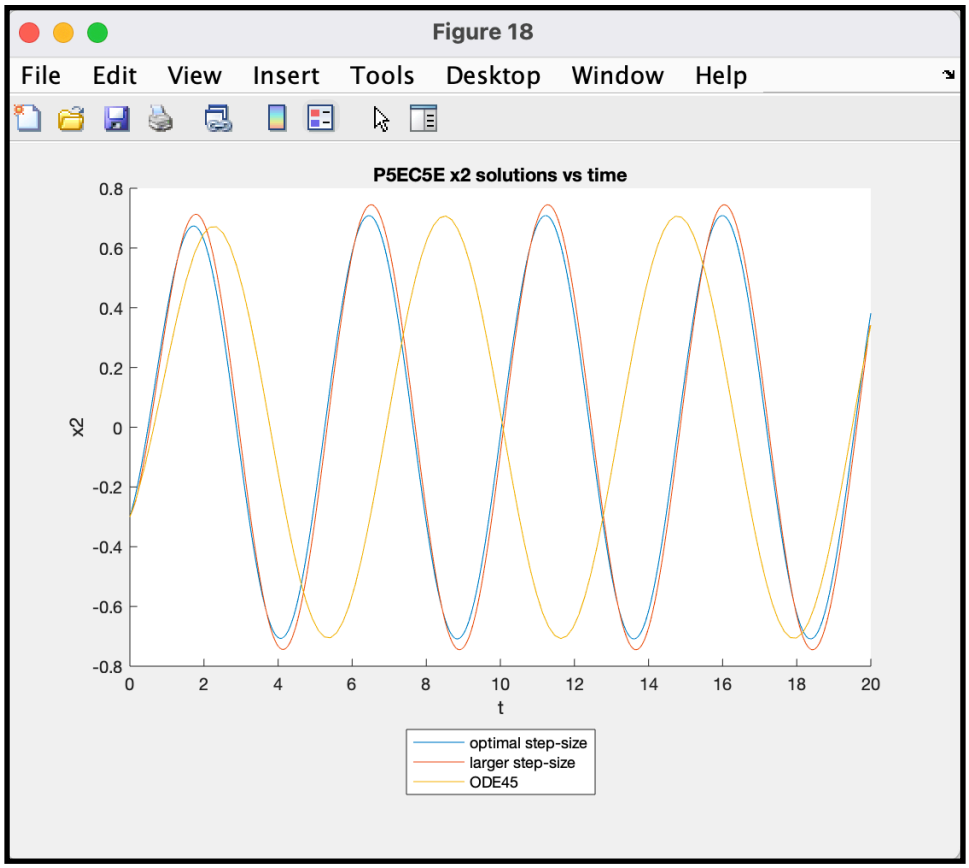


Fig. Solutions vs time comparison. Optimal step-size=0.001, Larger step-size=0.03

PART B: Results achieved for the RK4 with automatically adjusted step size

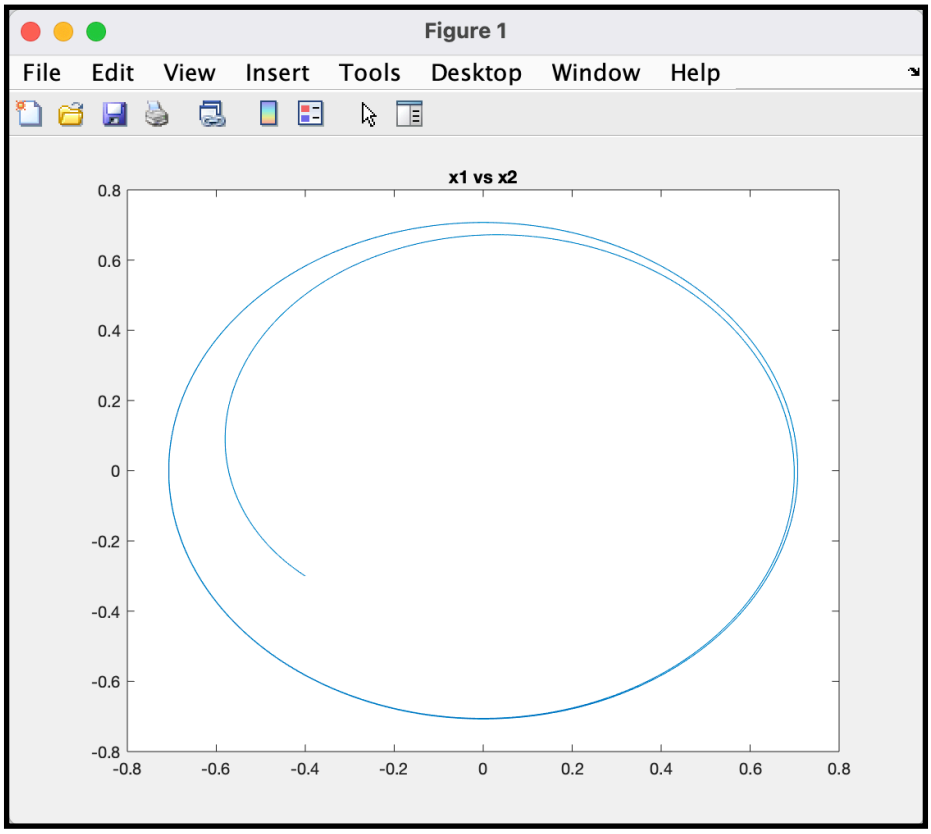


Fig. x_2 versus x_1

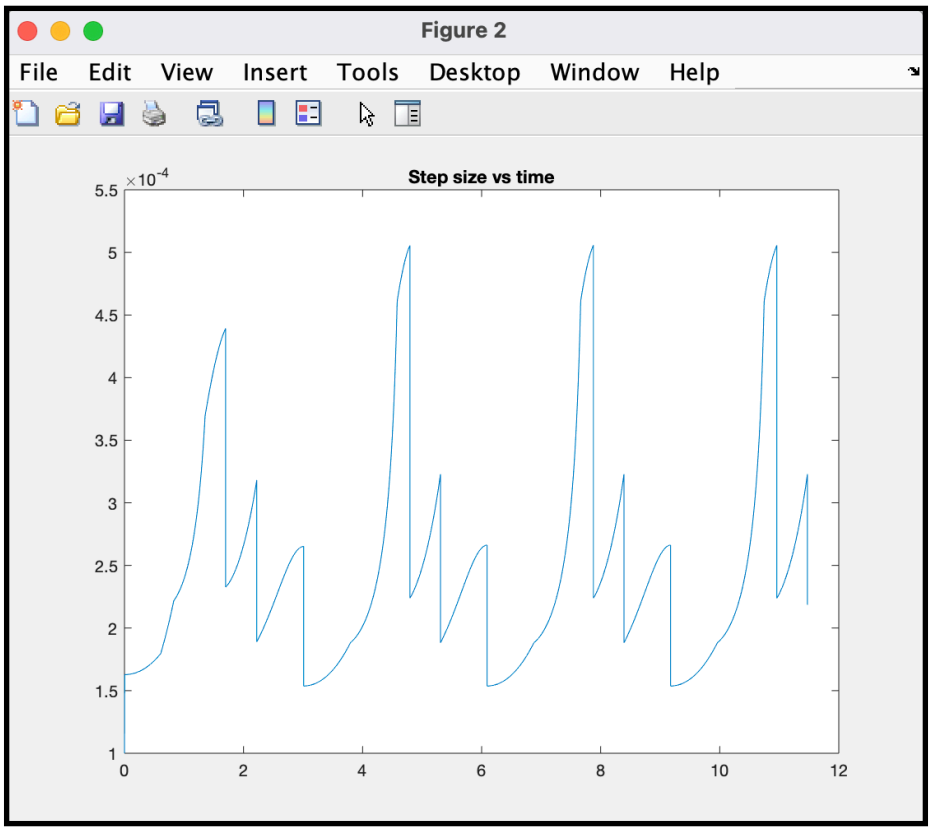


Fig. Step size versus time

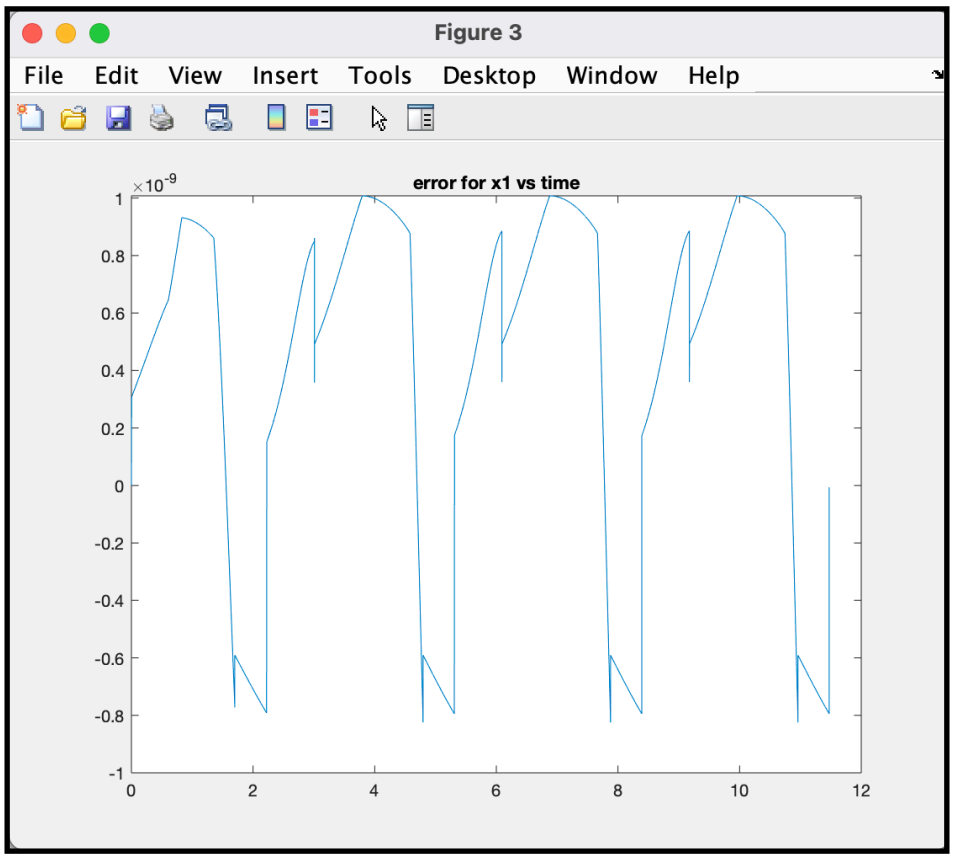


Fig. Error estimate for x_1 versus time

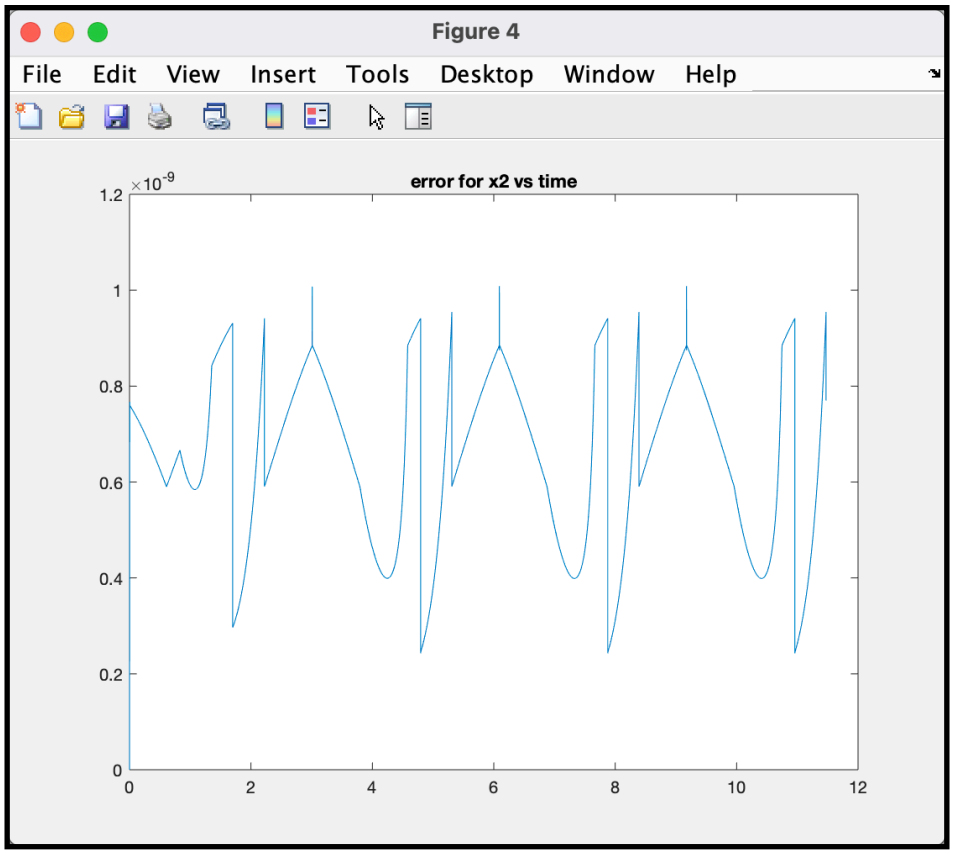


Fig. Error estimate for x_2 versus time

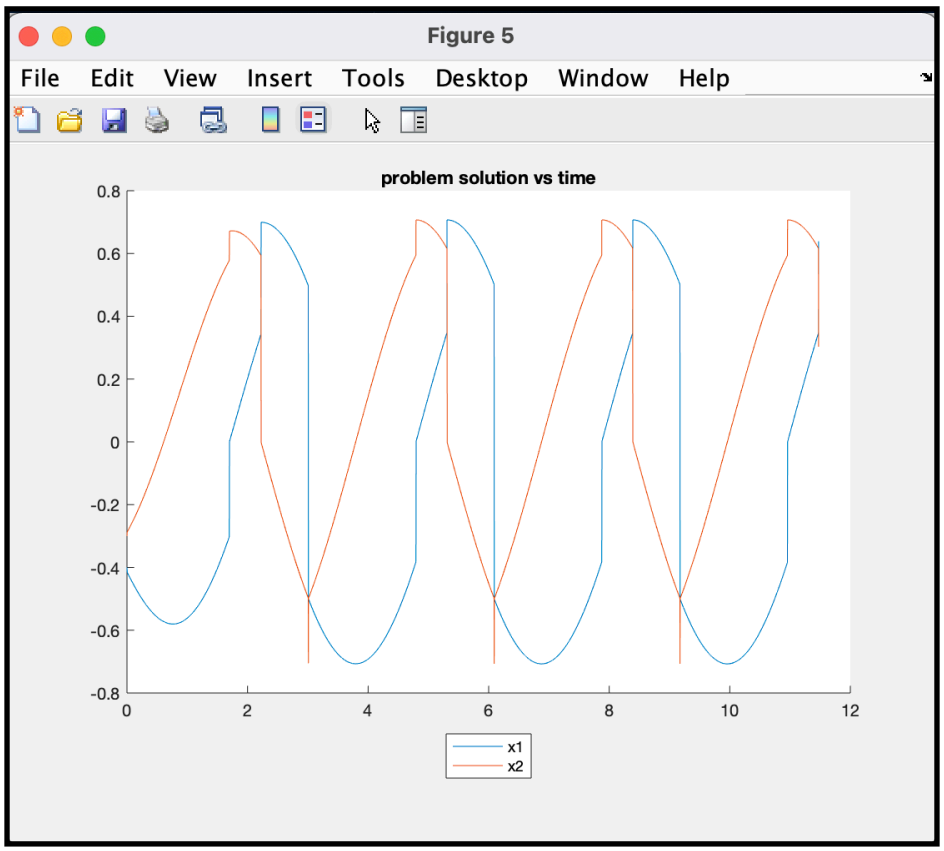


Fig. Problem solution versus time

Conclusions

Part A:

Finding the right step size is difficult and not always accurate. Very small changes can sometimes make big differences that affects the results. The best approach for this problem is to slowly and carefully investigate the effect of changes on the result.

The best optimal values were 0.004 for RK4 and 0.001 for P5EC5E. Decrease of these values does not influence the solution significantly, but its increase does. Comparing them with built in ODE45 shows that the values overlap, they are almost equal.

To show the effect of increased values, figures were plotted for step size 0.04 for RK4 and 0.03 for P5EC5E. Here the comparison with ODE45 shows great differences, values vary considerably and two colours can be easily distinguishable.

Part B:

Knowledge gained in previous ENUME projects were useful to select the appropriate parameters for automatised RK4 algorithm. Minimal step size is equal to machine epsilon, minimal positive floating point number, which is qual to 2.22e-16. Approaching this value already results in measurement errors, going below it is basically pointless to continue, as the function would not display the result. Absolute and relative tolerances are equal to 1e-9.

The initial step size was set to 0.0001, which turned to be too small and algorithm increased the value. From the figure showing the dependancy between step size and time, pattern may be drawn. The size increases until it suddenly drops, then it increases slightly and then falls again, to finally rise again and fall to the value at which it started. The highest value is slightly above 5e-4.

Analysing the error estimate versus time figures shows that errors for both x_1 and x_2 were nearly always smaller then set 1e-9, for x_1 partly below 0. Error figures, as the step size versus time, shows repeating patterns 3 or 4 times over the interval.

RK4 with automatically adjusting step size is for sure much more useful and more reliable then manually setting and looking for values, even very small changes in step size can already make a big difference. What is worse, watching the graphs shows that set initial value of the step size is not always the same as the initial value, it changes during the algorithm.

However, the simplicity of the normal RK4 cannot be questioned. RK4 with auto adjustment is much more complicated MATLAB code that needs good and detailed background knowledge to implement.

MATLAB Code

```
% close all windows and figures, clear command window
close all;
clear all;
clc;

%%%%%%%%%% ENUME %%%%%%%%%%
%%%%% PROJECT C NO.61 %%%%%
% JAKUB TOMKIEWICZ 300183 %

%%%%%%%% PROBLEM 1 %%%%%%%%

% data
x = [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5];
y = [-36.4986, -20.1300, -10.4370, -3.8635, -1.3503, 0.8879, 0.8176, 1.4830,
4.4024, 10.3910, 21.3003];

% plot graphs of polynomials
solveAndCreateFigure(x,y,2);
solveAndCreateFigure(x,y,4);
solveAndCreateFigure(x,y,6);
solveAndCreateFigure(x,y,8);
solveAndCreateFigure(x,y,10);

% show errors and condition nr for all
showDataForAllDegrees(x,y);
```

```
%%%%%%%% PROBLEM 2 %%%%%%%%

%%%%%% PART A %%%%%%

% Solution curves %

plotSolutionCurves(1,1); % optimal RK4
plotSolutionCurves(1,0); % optimal P5EC5E

plotSolutionCurves(0,1); % larger RK4
plotSolutionCurves(0,0); % larger P5EC5E

plotOptimalVsLarger(1); % RK4
plotOptimalVsLarger(0); % P5EC5E

% Solution versus time %

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

%P5EC5E
plotSolutionTime(0,1); % x1
plotSolutionTime(0,0); % x2

%%%%%% PART B %%%%%%

[x1,x2,t,h,error1,error2] = RK4autoAdjust(0.0001,0,20,1e-9,1e-9);

figure(1);
plot(x1,x2);
title("x1 vs x2");

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

figure(3);
plot(t,error1);
title("error for x1 vs time");

figure(4);
plot(t,error2);
title("error for x2 vs time");

figure(5);
hold on;
plot(t,x1);
plot(t,x2);
hold off;
title("problem solution vs time");
legend("x1","x2",'Location','southoutside');
```

```
%%%%%%%% FUNCTIONS %%%%%%%%

% PROBLEM 1 algorithm for the QR factorization, same as in Project A
function [Q,R] = qrmgs(A)

    [m n] = size(A);

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

    % factorization with orthogonal/not orthogonoal columns of Q
    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

    % column 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

% PROBLEM 1 create matrix known as Grams matrix
function [A,b] = createGramsMatrix(x,y,degree)

    degree = degree+1;

    % matrix A and vector b filled with zeros
    A = zeros(degree, degree);
    b = zeros(degree, 1);

    n = length(x);

    % i and j cant start from 0 as array indices has to be positive

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

    % vector b
    for i=1:degree
        for k=1:n % sum
```

```
            b(i, 1) = b(i, 1) + (y(k)*(x(k)^(i-1))); % auxiliary definition
        end
    end
end

% PROBLEM 1 solve equations, most code taken from Project A
function [error, conditionNr, C] = solveEquation(A,b)

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

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

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

    % starting from end until 1
    for i=n:-1:1 % back-substitution phase
        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);

    % conditionNr
    conditionNr = cond(A);
end

% PROBLEM 1 solve equation and print figure
function solveAndCreateFigure(x,y,degree)

    % create Gram's matrix and solve equation
    [A,b] = createGramsMatrix(x,y,degree);

    % we need only C
    [~,~,C] = solveEquation(A,b);

    disp(C);
    %C2 = A\b; % make sure if answer is same as buildin function
    %disp(C2);

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

    % functions various from degree
    if(degree == 2)
        f = @(x)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 == 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 == 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

    if(degree == 10)
        f = @(x)C(11)*x^10 + C(10)*x^9 + 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'); % from -5 to 5
    grid on;
    hold off;
    title("Polynomial of degree " + degree);
    % put legend in less conflict area
    legend("Samples","Polynomial",'Location','Best');

end

% PROBLEM 1 show error and condition nr for all degrees
function showDataForAllDegrees(x,y)

    % from degree 1 to degree 10
    for degree=1:1:10
        % create
        [A,b] = createGramsMatrix(x,y,degree);
        %disp(A);
        %disp(b);
        % solve
        [error,conditionNr,~] = solveEquation(A,b);
        disp("Degree "+ degree + " Error " + error + " ConditionNr " +
conditionNr);
    end

end

% PROBLEM 2 calculate Runge Kutta 4th order
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) = -0.4;
```

```
    y2(1) = -0.3;

    % 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

% PROBLEM 2 calculate Adams method P5EC5E
function [y1,y2,x] = P5EC5E(h, beginning, ending)

    %P5EC5E method so k=5
    k = 5;

    % values taken from book, page 177
    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

% PROBLEM 2 plot solution curves
function plotSolutionCurves(optimal,rk4)
    if(optimal == 1) % optimal step size
        if(rk4 == 1) % use RK4
            [x1,x2] = RK4(0.004,0,20);

            % 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)], [0,20], [-0.4,-0.3]);
            % t is a time vector, y is a vector solutions of a function

            figure(11);
            hold on;

            plot(x1,x2);
            plot(y_ode45(:,1),y_ode45(:,2));
            legend("RK4","ODE45",'Location','Best');
            title("RK4 vs ODE45 optimal step-size");
            hold off;
        else % use P5EC5E
            [x1,x2] = P5EC5E(0.001,0,20);

            % 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)], [0,20], [-0.4,-0.3]);
            % t is a time vector, y is a vector solutions of a function

            figure(12);
            hold on;

            plot(x1,x2);
            plot(y_ode45(:,1),y_ode45(:,2));
            legend("P5EC5E","ODE45",'Location','Best');
            title("P5EC5E vs ODE45 optimal step-size");
            hold off;
        end
    else % larger step size
        if(rk4 == 1) % use RK4
            [x1,x2] = RK4(0.04,0,20);

            % 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)], [0,20], [-0.4,-0.3]);
            % t is a time vector, y is a vector solutions of a function

            figure(13);
            hold on;

            plot(x1,x2);
            plot(y_ode45(:,1),y_ode45(:,2));
            legend("RK4","ODE45",'Location','Best');
            title("RK4 vs ODE45 larger step-size");
            hold off;
        else % use P5EC5E
            [x1,x2] = P5EC5E(0.03,0,20);

            % 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)], [0,20], [-0.4,-0.3]);
            % t is a time vector, y is a vector solutions of a function

            figure(14);
            hold on;

            plot(x1,x2);
            plot(y_ode45(:,1),y_ode45(:,2));
            legend("P5EC5E","ODE45",'Location','Best');
            title("P5EC5E vs ODE45 larger step-size");
            hold off;
        end
    end
end

% PROBLEM 2 plot solutions vs time
function plotSolutionTime(rk4,x1)
    if(rk4 == 1) % use RK4
        if(x1 == 1) % for x1
            [x1,~,t1] = RK4(0.004,0,20);
```

```
            [x1_2,~,t1_2] = RK4(0.04,0,20);

            % 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)], [0,20], [-0.4,-0.3]);
            % 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(0.004,0,20);

            [~,x2_2,t1_2] = RK4(0.04,0,20);

            % 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)], [0,20], [-0.4,-0.3]);
            % 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(0.001,0,20);

            [x1_2,~,t1_2] = P5EC5E(0.03,0,20);

            % 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)], [0,20], [-0.4,-0.3]);
            % 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 20]);
            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(0.001,0,20);

            [~,x2_2,t1_2] = P5EC5E(0.03,0,20);

            % 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)], [0,20], [-0.4,-0.3]);
            % 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 20]);
            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

% PROBLEM 2 plot optimal vs larger step-size
function plotOptimalVsLarger(rk4)
    if(rk4 == 1) % RK4
        [x1,x2] = RK4(0.004,0,20); % optimal
        [x1_2,x2_2] = RK4(0.04,0,20); % larger

        figure(19);
        hold on;

        plot(x1,x2);
        plot(x1_2,x2_2);
        legend("optimal step-size","larger step-size",'Location','Best');
        title("RK4 optimal vs larger step-size");
        hold off;
    else % P5EC5E
        [x1,x2] = P5EC5E(0.001,0,20); % optimal
        [x1_2,x2_2] = P5EC5E(0.03,0,20); % larger

        figure(20);
        hold on;
```

```
        plot(x1,x2);
        plot(x1_2,x2_2);
        legend("optimal step-size","larger step-size",'Location','Best');
        title("P5EC5E optimal vs larger step-size");
        hold off;
    end
end

% PROBLEM 2 RK4 with adjustable initial variables for RK4autoAdjust
function [root1,root2,x] = RK4withInitialValues(h, beginning, ending, initial1,
initial2)

    x = beginning:h:ending;

    % dont prealocate with zeros here

    % initial conditions
    y1(1) = initial1;
    y2(1) = initial2;

    % 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
```

```
% PROBLEM 2 calculate RK4 with automatic step-size adjustement
function [y1,y2,x,auto_h,error_1,error_2] =
RK4autoAdjust(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) = -0.4;
    y2(1) = -0.3;

    % 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 % more than enought, answer is achieved at 56756 iterations

        % after single step
        [y1_single,y2_single,time_single] =
RK4withInitialValues(auto_h(i),x(i),auto_h(i)+x(i),y1(i),y2(i));


        % after double step
        [y1_double,y2_double,time_double] =
RK4withInitialValues(auto_h(i)*0.5,x(i),auto_h(i)+x(i),y1(i),y2(i));


        % 7.22 from page 169
        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); % 7.33 page 173
        alpha_2 = (eps2/abs(delta_y2_double))^(1/5);

        % we need to take worst case scenerio, so only smallest alpha needs
        % to be taken into accound (smallest alpha with the biggest
        % denominator)
        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
```

Sources of knowledge

„Numerical methods” Piotr Tatjewski
MathWorks Help Center
Youtube channel „Let’s Code Physics”