

Numerical Methods II

Project 2

The Runge-Kutta method of 4th order (the Gill formula) for a system of 2 ordinary differential equations.

December 7, 2018

Rafał Ziomek

Group: EA2

Contents

Method description	2
Runge-Kutta-Gill method	2
Algorithm description	3
Matlab functions code	4
solveODE	4
test	6
Numerical examples and analysis	14
Tests 1-4	14
Tests 5-6	17
Tests 7-10	19

METHOD DESCRIPTION

Runge-Kutta-Gill method

Method I will use in my calculations is one of the 4th order Runge-Kutta methods - the Gill's formula. For ordinary differential equations it is stated as:

$$x(t+h) = x(t) + \frac{1}{6}[F_1 + (2 - \sqrt{2})F_2 + (2 + \sqrt{2})F_3 + F_4] \quad (1)$$

where:

$$\left\{ \begin{array}{l} h - \text{step size of the method} \\ F_1 = hf(t, x) \\ F_2 = hf(t + \frac{1}{2}h, x + \frac{1}{2}F_1) \\ F_3 = hf(t + \frac{1}{2}h, x + \frac{1}{2}(\sqrt{2} - 1)F_1 + \frac{1}{2}(2 - \sqrt{2})F_2) \\ F_4 = hf(t + h, x - \frac{1}{2}\sqrt{2}F_2 + \frac{1}{2}(2 + \sqrt{2})F_3) \end{array} \right.$$

Presented method can be easily extended to system of two ordinary differential equations. Using vector notation, let:

$$X' = \begin{pmatrix} x_1' = f_1(t, x_1, x_2) \\ x_2' = f_2(t, x_1, x_2) \end{pmatrix} = F(t, X)$$

Now, formula (1) can be rewritten as:

$$X(t+h) = X(t) + \frac{1}{6}[F_1 + (2 - \sqrt{2})F_2 + (2 + \sqrt{2})F_3 + F_4] \quad (2)$$

where:

$$\left\{ \begin{array}{l} h - \text{step size of the method} \\ F_1 = hF(t, X) \\ F_2 = hF(t + \frac{1}{2}h, X + \frac{1}{2}F_1) \\ F_3 = hF(t + \frac{1}{2}h, X + \frac{1}{2}(\sqrt{2} - 1)F_1 + \frac{1}{2}(2 - \sqrt{2})F_2) \\ F_4 = hF(t + h, X - \frac{1}{2}\sqrt{2}F_2 + \frac{1}{2}(2 + \sqrt{2})F_3) \end{array} \right.$$

Algorithm description

In order to implement Runge-Kutta-Gill method for system of two ordinary differential equations, formula (2) is changed into following:

$$\left\{ \begin{array}{l} x_1(t + h) = x_1(t) + \frac{1}{6}[F_1 + (2 - \sqrt{2})F_2 + (2 + \sqrt{2})F_3 + F_4] \\ x_2(t + h) = x_2(t) + \frac{1}{6}[G_1 + (2 - \sqrt{2})G_2 + (2 + \sqrt{2})G_3 + G_4] \end{array} \right.$$

where:

$$\left\{ \begin{array}{l} h - \text{step size of the method} \\ F_1 = hf_1(t, x_1, x_2) \\ G_1 = hf_2(t, x_1, x_2) \\ F_2 = hf_1(t + \frac{1}{2}h, x_1 + \frac{1}{2}F_1, x_2 + \frac{1}{2}G_1) \\ G_2 = hf_2(t + \frac{1}{2}h, x_1 + \frac{1}{2}F_1, x_2 + \frac{1}{2}G_1) \\ F_3 = hf_1(t + \frac{1}{2}h, x_1 + \frac{1}{2}(\sqrt{2} - 1)F_1 + \frac{1}{2}(2 - \sqrt{2})F_2, x_2 + \frac{1}{2}(\sqrt{2} - 1)G_1 + \frac{1}{2}(2 - \sqrt{2})G_2) \\ F_G = hf_2(t + \frac{1}{2}h, x_1 + \frac{1}{2}(\sqrt{2} - 1)F_1 + \frac{1}{2}(2 - \sqrt{2})F_2, x_2 + \frac{1}{2}(\sqrt{2} - 1)G_1 + \frac{1}{2}(2 - \sqrt{2})G_2) \\ F_4 = hf_1(t + h, x_1 - \frac{1}{2}\sqrt{2}F_2 + \frac{1}{2}(2 + \sqrt{2})F_3, x_2 - \frac{1}{2}\sqrt{2}G_2 + \frac{1}{2}(2 + \sqrt{2})G_3) \\ G_4 = hf_2(t + h, x_1 - \frac{1}{2}\sqrt{2}F_2 + \frac{1}{2}(2 + \sqrt{2})F_3, x_2 - \frac{1}{2}\sqrt{2}G_2 + \frac{1}{2}(2 + \sqrt{2})G_3) \end{array} \right.$$

My algorithm also returns error of calculated $x_{1,i}, x_{2,i}$, by subtracting it from known exact solution.

MATLAB FUNCTIONS CODE

solveODE

```
1 %Function takes as arguments:
2 % ODE1 – first ordinary differential equation in the system as
   anonymous function
3 % ODE2 – second ordinary differential equation in the system as
   anonymous function
4 % a – start of the calculations interval
5 % b – end of the calculations interval
6 % n – number of steps
7 % initODE1 – initial value for ODE1
8 % initODE2 – initial value for ODE2
9 % exactODE1 – exact solution function of first ordinary differential as
   anonymous function
10 % exactODE2 – exact solution function of second ordinary differential
    equation in the system as anonymous function
11
12 %Function returns:
13 %t – vector of used variables
14 %x – calculated values of x(t) – from ODE1
15 %y – calculated values of y(t) – from ODE2
16 %errODE1 – error of calculated values of x(t)
17 %errODE2 – error of calculated values of y(t)
18
19 function [t, x, y, errODE1, errODE2]=solveODE(ODE1, ODE2, a, b, n,
    initODE1, initODE2, exactODE1, exactODE2)
20 h = (b-a)/n;           %h – step size
21 t = a:h:b;             %initialization of needed vectors
22 x = zeros(1,n+1);
23 y = zeros(1,n+1);
24 errODE1 = zeros(1,n+1);
25 errODE2 = zeros(1,n+1);
26 x(1) = initODE1;       %assigning initial values in the vector
```

```

27 y(1) = initODE2;
28 sqrt2 = sqrt(2);           %calculation of square root of 2 so it is
    not calculated every time when needed
29 for i=1:n                  %loop of calculations, described in report
30     F1 = h*ODE1(t(i), x(i), y(i));
31     G1 = h*ODE2(t(i), x(i), y(i));
32     F2 = h*ODE1(t(i)+0.5*h, x(i)+0.5*F1, y(i)+0.5*G1);
33     G2 = h*ODE2(t(i)+0.5*h, x(i)+0.5*F1, y(i)+0.5*G1);
34     F3 = h*ODE1(t(i)+0.5*h, x(i)+0.5*(sqrt2-1)*F1 + 0.5*(2-sqrt2)*
        F2, y(i) + 0.5*(sqrt2-1)*G1 + 0.5*(2-sqrt2)*G2);
35     G3 = h*ODE2(t(i)+0.5*h, x(i)+0.5*(sqrt2-1)*F1 + 0.5*(2-sqrt2)*
        F2, y(i) + 0.5*(sqrt2-1)*G1 + 0.5*(2-sqrt2)*G2);
36     F4 = h*ODE1(t(i)+h, x(i)-0.5*sqrt2*F2+0.5*(2+sqrt2)*F3, y(i)
        -0.5*sqrt2*G2+0.5*(2+sqrt2)*G3);
37     G4 = h*ODE2(t(i)+h, x(i)-0.5*sqrt2*F2+0.5*(2+sqrt2)*F3, y(i)
        -0.5*sqrt2*G2+0.5*(2+sqrt2)*G3);
38
39     x(i+1) = x(i) + (1/6)*(F1 + (2-sqrt2)*F2 + (2+sqrt2)*F3 + F4);
40     y(i+1) = y(i) + (1/6)*(G1 + (2-sqrt2)*G2 + (2+sqrt2)*G3 + G4);
41     errODE1(i+1) = exactODE1(t(i+1), x(i+1), y(i+1)) - x(i+1);
42     errODE2(i+1) = exactODE2(t(i+1), x(i+1), y(i+1)) - y(i+1);
43 end
44 end

```

test

```

1 %TEST FILE , AFTER EACH PLOT PLEASE PRESS ANY BUTTON TO SEE PLOT FROM
  NEXT TEST
2 %SYSTEM OF ODE FOR TESTS 1-4
3
4 %ODE1:  $x' = x + 4y - e^t$ 
5 %ODE2:  $y' = x + y + 2e^t$ 
6 % $x(0) = 4$ 
7 % $y(0) = 5/4$ 
8 %exactX:  $4e^{3t} + 2e^{-t} - 2e^t$ 
9 %exactY:  $2e^{3t} - e^{-t} + 1/4e^t$ 
10
11 %TEST 1 # of steps:5
12
13 n=5;
14 [t,x,y,errx,erry] = solveODE(@(t,x,y)x+4*y-exp(t), @(t,x,y) x + y + 2*
    exp(t), 0, 1, n, 4, 1.25, @(t,x,y) 4*exp(3*t) + 2*exp(-t) - 2*exp(t)
    , @(t,x,y)2*exp(3*t) - exp(-t) + 0.25*exp(t));
15
16 ax1 = subplot(2,1,1);
17 plot(ax1,t,x, t,y);
18 title(ax1,'Calculated values')
19 ylabel(ax1,'f(t)')
20 xlabel(ax1,'t')
21 legend('x(t)','y(t)')
22 ax1 = subplot(2,1,2);
23 plot(ax1,t,errx,t,erry);
24 title(ax1,'Errors')
25 ylabel(ax1,'exact(t) - f(t)')
26 xlabel(ax1,'t')
27 legend('error x(t)','error(y(t))')
28
29 waitforbuttonpress
30

```

```
31 %TEST 2 # of steps:20
32
33 n=20;
34 [t,x,y,errx,erry] = solveODE(@(t,x,y)x+4*y-exp(t), @(t,x,y) x + y + 2*
    exp(t), 0, 1, n, 4, 1.25, @(t,x,y) 4*exp(3*t) + 2*exp(-t) - 2*exp(t)
    , @(t,x,y)2*exp(3*t) - exp(-t) + 0.25*exp(t));
35
36 ax1 = subplot(2,1,1);
37 plot(ax1,t,x, t,y);
38 title(ax1, 'Calculated values')
39 ylabel(ax1, 'f(t)')
40 xlabel(ax1, 't')
41 legend('x(t)', 'y(t)')
42 ax1 = subplot(2,1,2);
43 plot(ax1,t,errx,t,erry);
44 title(ax1, 'Errors')
45 ylabel(ax1, 'exact(t) - f(t)')
46 xlabel(ax1, 't')
47 legend('error x(t)', 'error(y(t))')
48
49 waitforbuttonpress
50
51 %TEST 3 # of steps:100
52
53 n=100;
54 [t,x,y,errx,erry] = solveODE(@(t,x,y)x+4*y-exp(t), @(t,x,y) x + y + 2*
    exp(t), 0, 1, n, 4, 1.25, @(t,x,y) 4*exp(3*t) + 2*exp(-t) - 2*exp(t)
    , @(t,x,y)2*exp(3*t) - exp(-t) + 0.25*exp(t));
55
56 ax1 = subplot(2,1,1);
57 plot(ax1,t,x, t,y);
58 title(ax1, 'Calculated values')
59 ylabel(ax1, 'f(t)')
60 xlabel(ax1, 't')
61 legend('x(t)', 'y(t)')
```



```

62 ax1 = subplot(2,1,2);
63 plot(ax1,t,errx,t,err);
64 title(ax1,'Errors')
65 ylabel(ax1,'exact(t) - f(t)')
66 xlabel(ax1,'t')
67 legend('error x(t)', 'error(y(t))')
68
69 waitforbuttonpress
70
71 %TEST 4 # of steps:1000
72
73 n=1000;
74 [t,x,y,errx,err] = solveODE(@(t,x,y)x+4*y-exp(t), @(t,x,y) x + y + 2*
    exp(t), 0, 1, n, 4, 1.25, @(t,x,y) 4*exp(3*t) + 2*exp(-t) - 2*exp(t)
    , @(t,x,y)2*exp(3*t) - exp(-t) + 0.25*exp(t));
75
76 ax1 = subplot(2,1,1);
77 plot(ax1,t,x,t,y);
78 title(ax1,'Calculated values')
79 ylabel(ax1,'f(t)')
80 xlabel(ax1,'t')
81 legend('x(t)', 'y(t)')
82 ax1 = subplot(2,1,2);
83 plot(ax1,t,errx,t,err);
84 title(ax1,'Errors')
85 ylabel(ax1,'exact(t) - f(t)')
86 xlabel(ax1,'t')
87 legend('error x(t)', 'error(y(t))')
88
89 waitforbuttonpress
90
91 %SYSTEM OF ODE FOR TEST 5-6
92
93 %ODE1: x' = x - y
94 %ODE2: y' = x + 3y

```

```
95 %exactX: e^2t
96 %exactY: -e^2t
97
98 %TEST 5, n=10
99
100 n=10;
101 a=0;
102 b=1;
103 initx = exp(2*a);
104 inity = -exp(2*a);
105
106 [t,x,y,errx,erry] = solveODE(@(t,x,y) x - y, @(t,x,y) x + 3*y, a, b, n,
    initx, inity, @(t,x,y)exp(2*t), @(t,x,y)-exp(2*t));
107
108 ax1 = subplot(2,1,1);
109 plot(ax1,t,x,t,y);
110 title(ax1,'Calculated values')
111 ylabel(ax1,'f(t)')
112 xlabel(ax1,'t')
113 legend('x(t)','y(t)')
114 ax1 = subplot(2,1,2);
115 plot(ax1,t,errx,t,erry);
116 title(ax1,'Errors')
117 ylabel(ax1,'exact(t) - f(t)')
118 xlabel(ax1,'t')
119 legend('error x(t)','error(y(t))')
120
121 waitforbuttonpress
122
123 %TEST 6, n=1000
124
125 n=1000;
126 a=0;
127 b=1;
128 initx = exp(2*a);
```

```

129 inity = -exp(2*a);
130
131 [t,x,y,errx,erry] = solveODE(@(t,x,y) x - y, @(t,x,y) x + 3*y, a, b, n,
    initx, inity, @(t,x,y)exp(2*t), @(t,x,y)-exp(2*t));
132
133 ax1 = subplot(2,1,1);
134 plot(ax1,t,x, t,y);
135 title(ax1, 'Calculated values')
136 ylabel(ax1, 'f(t)')
137 xlabel(ax1, 't')
138 legend('x(t)', 'y(t)')
139 ax1 = subplot(2,1,2);
140 plot(ax1,t,errx,t,erry);
141 title(ax1, 'Errors')
142 ylabel(ax1, 'exact(t) - f(t)')
143 xlabel(ax1, 't')
144 legend('error x(t)', 'error(y(t))')
145
146 waitforbuttonpress
147
148 %STIFF SYSTEM OF ORDINARY DIFFERENTIAL EQUATIONS TESTS 8-10
149 %ODE1: x' = -80.6x + 119.4y
150 %ODE2: y' = 79.6x - 120.4y
151 %exactX: c1*3*e^-t + (-c2) * e^-200t
152 %exactY: c1*2*e^-t + c2 * e^-200t
153 %c1 and c2 depend on initial conditions, if c2==0, then solution should
    be
154 %smooth, otherwise it should produce rapid changes while approaching to
155 %solution
156
157 %TEST 7: c1=5, c2=0, hence x(0) = 15 y(0)= 10
158 c1 = 5;
159 c2 = 0;
160 a = 0;
161 b = 1;

```

```

162 initx = c1*3*exp(-a) + (-c2) * exp(-200*a);
163 inity = c1*2*exp(-a) + c2 * exp(-200*a);
164 [t,x,y,errx,erry] = solveODE(@(t,x,y)-80.6*x + 119.4*y, @(t,x,y) 79.6*x
    - 120.4*y, a, b, 200, initx, inity, @(t,x,y)c1*3*exp(-t) + (-c2) *
    exp(-200*t), @(t,x,y)c1*2*exp(-t) + c2 * exp(-200*t));
165
166 ax1 = subplot(2,1,1);
167 plot(ax1,t,x, t,y);
168 title(ax1, 'Calculated values')
169 ylabel(ax1, 'f(t)')
170 xlabel(ax1, 't')
171 legend('x(t)', 'y(t)')
172 ax1 = subplot(2,1,2);
173 plot(ax1,t,errx,t,erry);
174 title(ax1, 'Errors')
175 ylabel(ax1, 'exact(t) - f(t)')
176 xlabel(ax1, 't')
177 legend('error x(t)', 'error(y(t))')
178
179 waitforbuttonpress
180
181 %TEST 8: c1=5, c2=1, hence x(0) = 14 y(0)= 12
182 c1 = 5;
183 c2 = 1;
184 a = 0;
185 b = 1;
186 initx = c1*3*exp(-a) + (-c2) * exp(-200*a);
187 inity = c1*2*exp(-a) + c2 * exp(-200*a);
188 [t,x,y,errx,erry] = solveODE(@(t,x,y)-80.6*x + 119.4*y, @(t,x,y) 79.6*x
    - 120.4*y, a, b, 200, initx, inity, @(t,x,y)c1*3*exp(-t) + (-c2) *
    exp(-200*t), @(t,x,y)c1*2*exp(-t) + c2 * exp(-200*t));
189
190 ax1 = subplot(2,1,1);
191 plot(ax1,t,x, t,y);
192 title(ax1, 'Calculated values')

```

```

193 ylabel(ax1, 'f(t)')
194 xlabel(ax1, 't')
195 legend('x(t)', 'y(t)')
196 ax1 = subplot(2,1,2);
197 plot(ax1, t, errx, t, erry);
198 title(ax1, 'Errors')
199 ylabel(ax1, 'exact(t) - f(t)')
200 xlabel(ax1, 't')
201 legend('error x(t)', 'error(y(t))')
202
203 waitforbuttonpress
204
205 %TEST 9: c1=1, c2=0, hence x(0) = 3 y(0)= 2
206 c1 = 1;
207 c2 = 0;
208 a = 0;
209 b = 1;
210 initx = c1*3*exp(-a) + (-c2) * exp(-200*a);
211 inity = c1*2*exp(-a) + c2 * exp(-200*a);
212 [t,x,y,errx, erry] = solveODE(@(t,x,y)-80.6*x + 119.4*y, @(t,x,y) 79.6*x
    - 120.4*y, a, b, 200, initx, inity, @(t,x,y)c1*3*exp(-t) + (-c2) *
    exp(-200*t), @(t,x,y)c1*2*exp(-t) + c2 * exp(-200*t));
213
214 ax1 = subplot(2,1,1);
215 p = plot(ax1, t, x, t, y);
216 title(ax1, 'Calculated values')
217 ylabel(ax1, 'f(t)')
218 xlabel(ax1, 't')
219 legend('x(t)', 'y(t)')
220 ax1 = subplot(2,1,2);
221 plot(ax1, t, errx, t, erry);
222 title(ax1, 'Errors')
223 ylabel(ax1, 'exact(t) - f(t)')
224 xlabel(ax1, 't')
225 legend('error x(t)', 'error(y(t))')

```

```
226
227 waitforbuttonpress
228
229 %TEST 10: c1=1, c2=5, hence  $x(0) = -2$   $y(0) = 7$ 
230 c1 = 1;
231 c2 = 5;
232 a = 0;
233 b = 1;
234 initx = c1*3*exp(-a) + (-c2) * exp(-200*a);
235 inity = c1*2*exp(-a) + c2 * exp(-200*a);
236 [t,x,y,errx,erry] = solveODE(@(t,x,y)-80.6*x + 119.4*y, @(t,x,y) 79.6*x
    - 120.4*y, a, b, 200, initx, inity, @(t,x,y)c1*3*exp(-t) + (-c2) *
    exp(-200*t), @(t,x,y)c1*2*exp(-t) + c2 * exp(-200*t));
237
238 ax1 = subplot(2,1,1);
239 plot(ax1,t,x,t,y);
240 title(ax1,'Calculated values')
241 ylabel(ax1,'f(t)')
242 xlabel(ax1,'t')
243 legend('x(t)','y(t)')
244 ax1 = subplot(2,1,2);
245 plot(ax1,t,errx,t,erry);
246 title(ax1,'Errors')
247 ylabel(ax1,'exact(t) - f(t)')
248 xlabel(ax1,'t')
249 legend('error x(t)','error(y(t))')
```

NUMERICAL EXAMPLES AND ANALYSIS

Tests 1-4

For tests 1-4 I used following system of differential equations:

$$\begin{cases} x_1'(t) = x_1 + 4x_2 - e^t \\ x_2'(t) = x_1 + x_2 - 2e^t \end{cases}$$

with initial condition:

$$\begin{cases} x_1(0) = 4 \\ x_2(0) = \frac{5}{4} \end{cases}$$

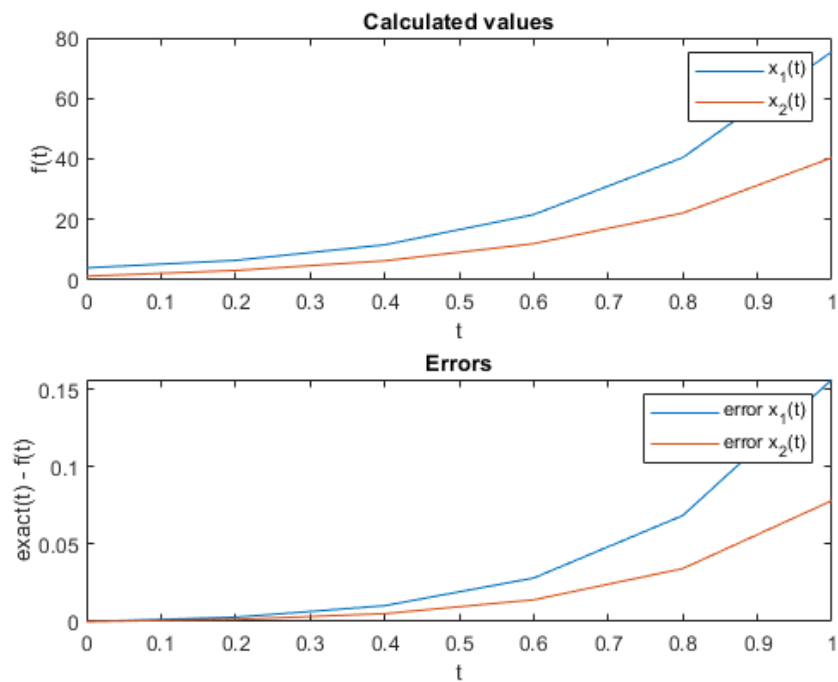
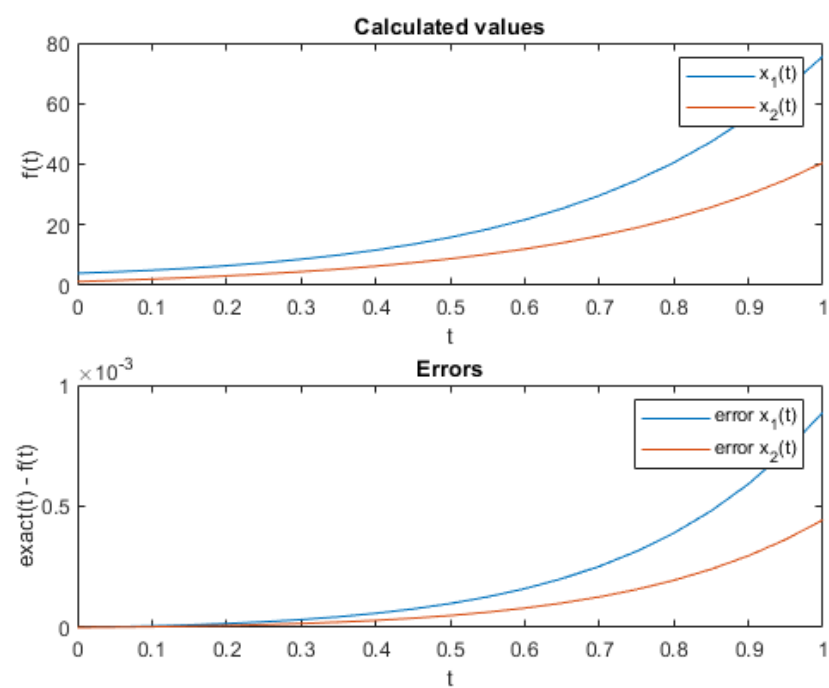
and exact solution:

$$\begin{cases} x_1(t) = 4e^{3t} + 2e^{-t} - 2e^t \\ x_2(t) = 2e^{3t} - e^{-t} + \frac{1}{4}e^t \end{cases}$$

In each test, the number of steps was increasing. For test 1 - 5 steps, test 2 - 20 steps, test 3 - 100 steps and test 4 - 1000 steps.

On the next pages there are plots of calculated solutions and errors.

As the number of steps increase (hence the step size gets smaller), the error of the method changes in magnitude, as expected. What I found quite interesting, is the fact that plots of and error and the function itself are very similar in shape.

**Figure 1: Test 1****Figure 2: Test 2**

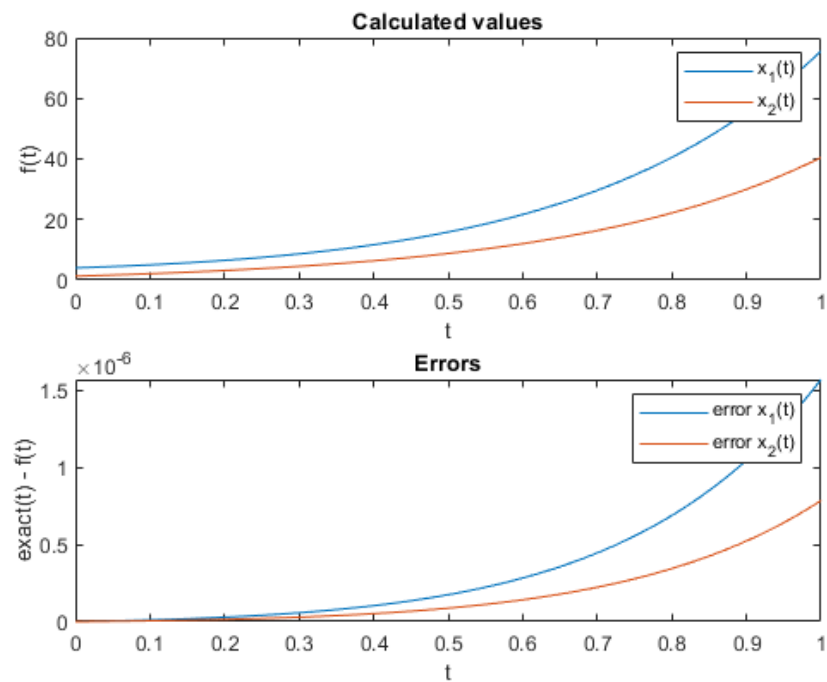


Figure 3: Test 3

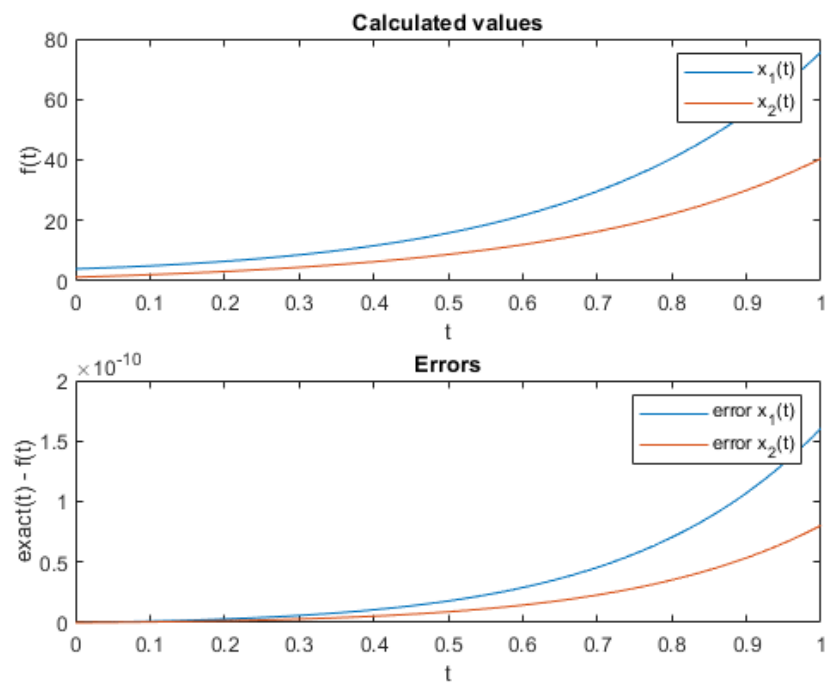


Figure 4: Test 4

Tests 5-6

For tests 5-6 I used following system of differential equations:

$$\begin{cases} x_1'(t) = x_1 - x_2 \\ x_2'(t) = x_1 + 3x_2 \end{cases}$$

with initial condition:

$$\begin{cases} x_1(0) = 2 \\ x_2(0) = -2 \end{cases}$$

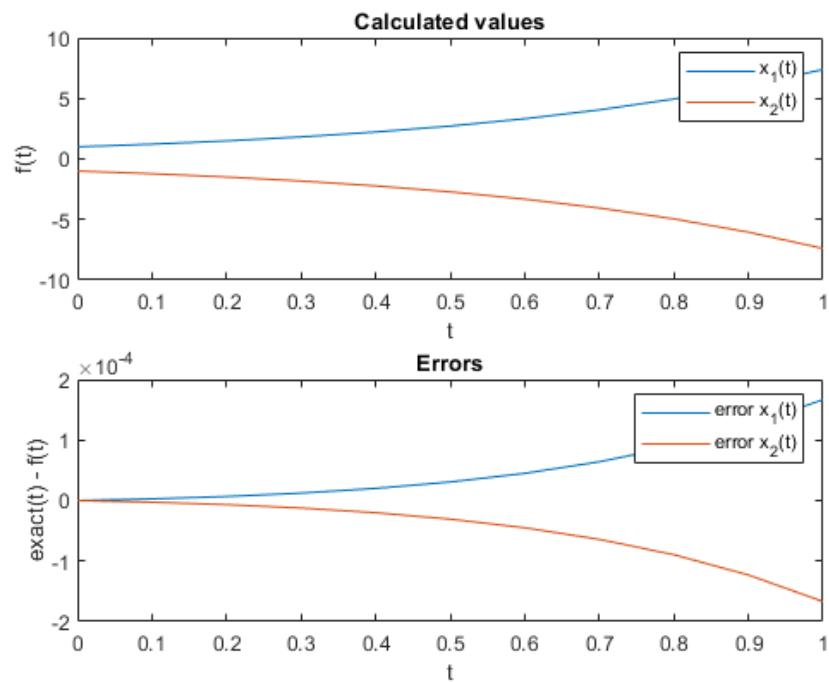
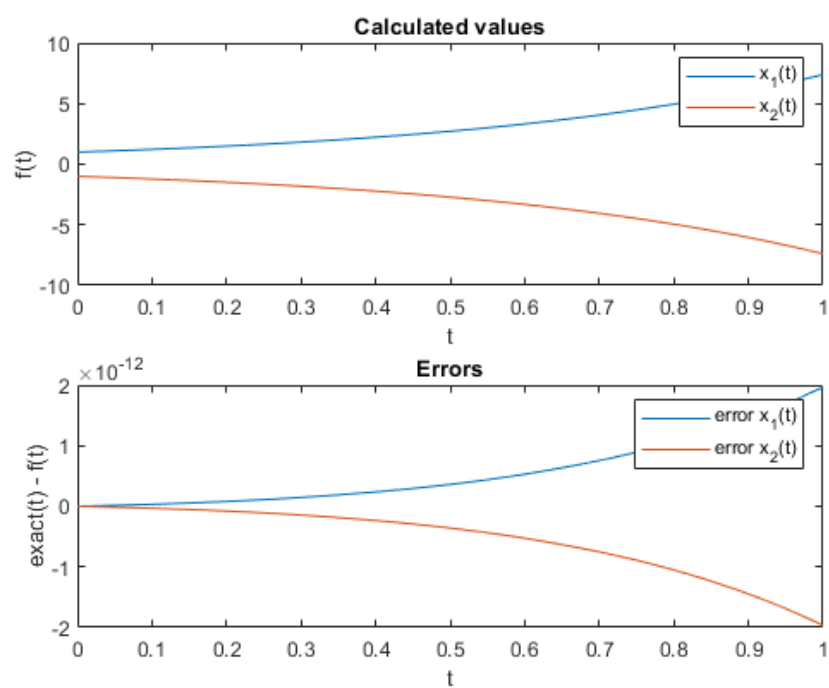
and exact solution:

$$\begin{cases} x_1(t) = e^{2t} \\ x_2(t) = -e^{2t} \end{cases}$$

As in previous tests, the number of steps was increasing. For test 5 - 10 steps and test 6 - 1000 steps.

On the next page there are plots of calculated solutions and errors.

Here I observed the same behaviour as in previous tests.

**Figure 5:** Test 5**Figure 6:** Test 6

Tests 7-10

For tests 7-10 I used following system of differential equations:

$$\begin{cases} x_1'(t) = -80.6x_1 + 119.4x_2 \\ x_2'(t) = 79.6x_1 - 120.4x_2 \end{cases}$$

with initial condition:

$$\begin{cases} x_1(0) = 3c_1 - c_2 \\ x_2(0) = 2c_1 + c_2 \end{cases}$$

and exact solution:

$$\begin{cases} x_1(t) = c_1 * 3e^{-t} - c_2e^{-200t} \\ x_2(t) = c_1 * 2e^{-t} + c_2e^{-200t} \end{cases}$$

This system of ODE is stiff, if $c_2 = 0$, then solution should be smooth, without significant error. However if $c_2 \neq 0$ it should produce significant error. For test 7, $c_1 = 5, c_2 = 0$, test 8 $c_1 = 5, c_2 = 1$, test 9 $c_1 = 1, c_2 = 0$ and test 10 $c_1 = 5, c_2 = 0$.

Plots of these tests are on following pages.

It can be seen, that results are as expected, meaning that Runge-Kutta-Gill method is not suitable for every type of ODE.

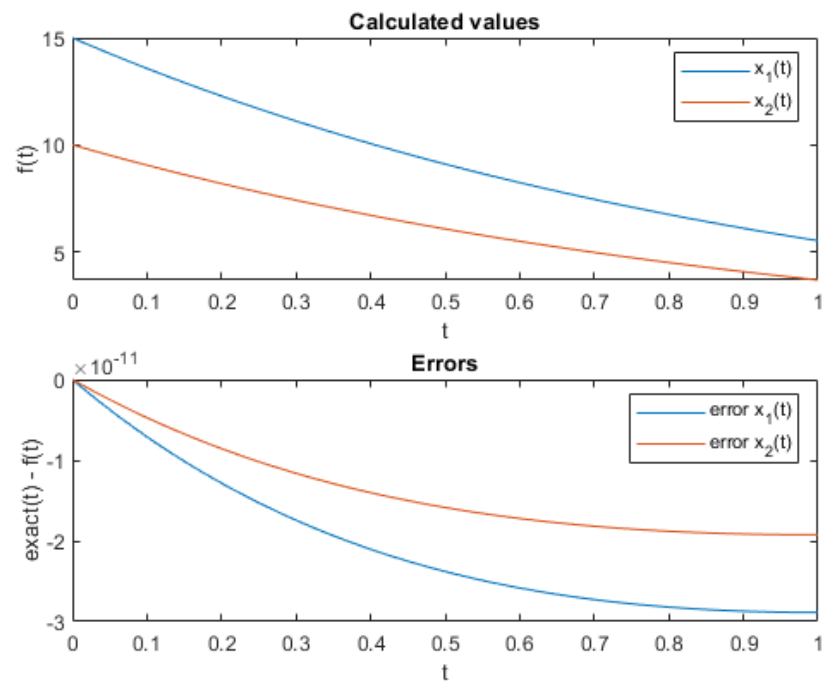


Figure 7: Test 7

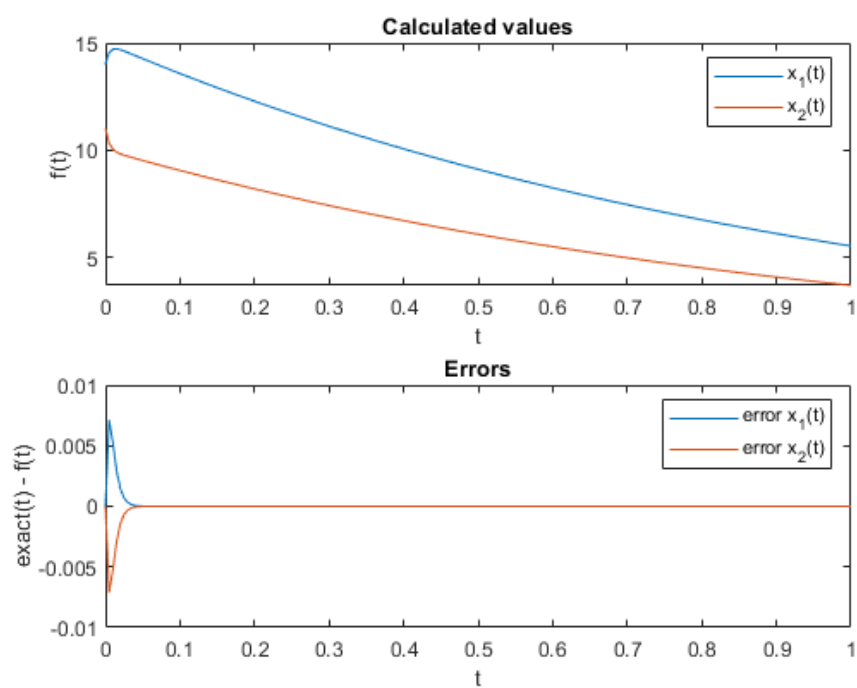


Figure 8: Test 8

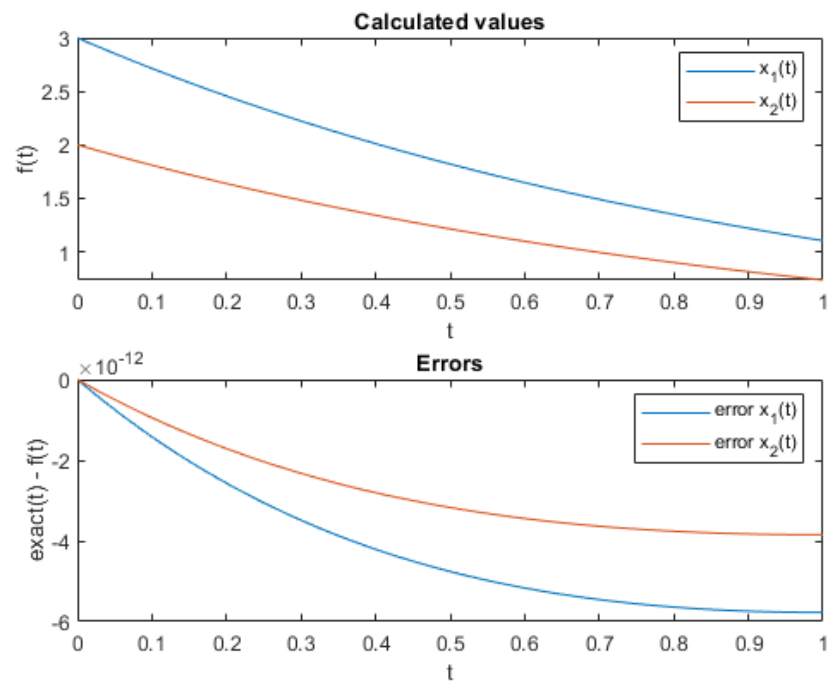


Figure 9: Test 9

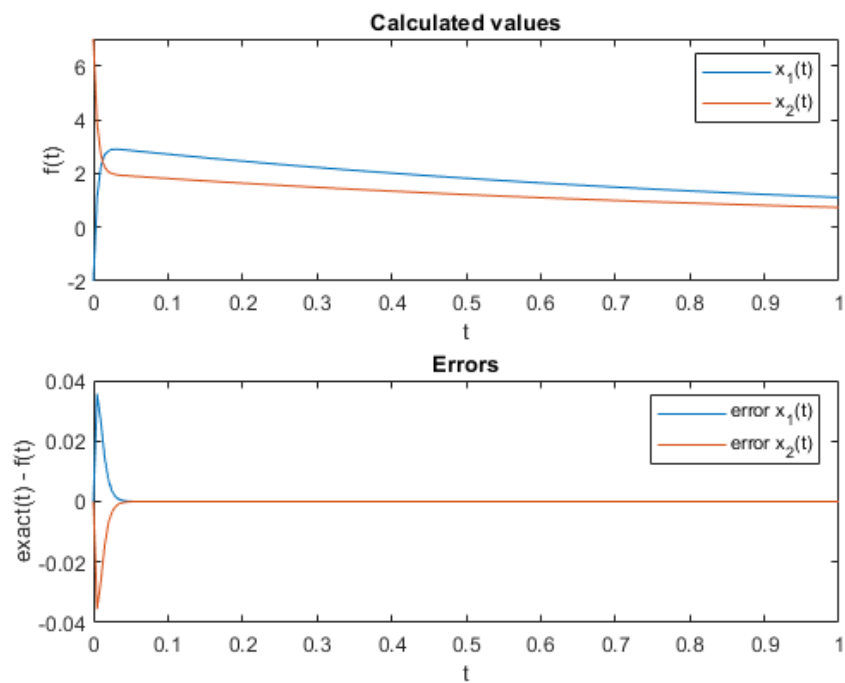


Figure 10: Test 10