Mussin Daniyar

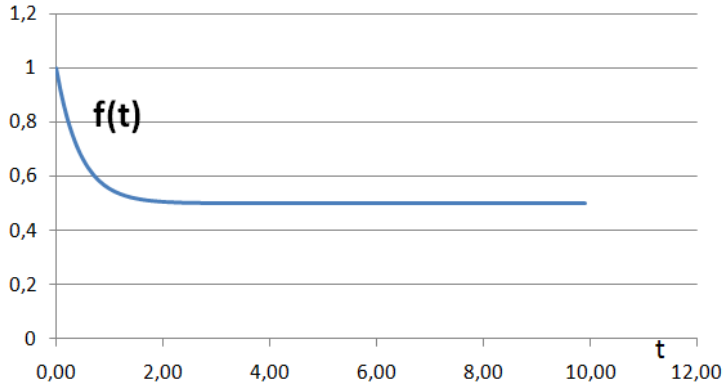
Modelle für virtuelle Realitäten

Grundlagen der numerischen Integration

Exercise 1.1

In file "Euler_Expl" I have realized the Explicit Integration for all functions. (in my example it is made for f=f' example);

a) The results from my program I put to the excel and draw the graph:



The **Wolfram's** result

Input interpretation:

solve	$f' + 2f = 1$	for	f

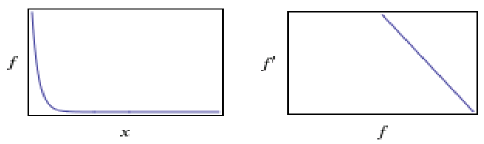
ODE classification:

first-order linear ordinary differential equation

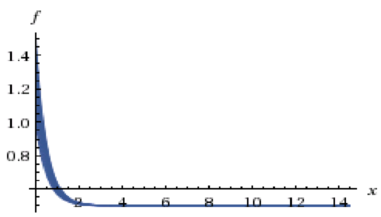
Differential equation solution:

$$f(x) = c_1 e^{-2x} + \frac{1}{2}$$

Plots of sample individual solution:



Sample solution family:



(sampling $f(0)$)

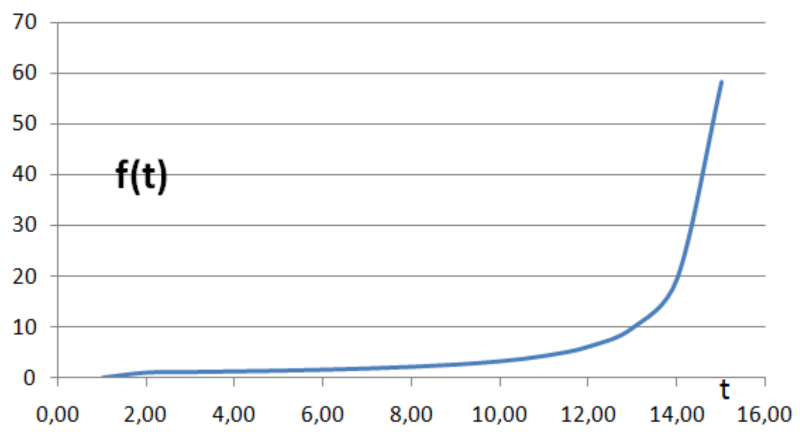
The table with results: first 10 points

t	f(t)
0.00	1
0.1	0.9
0.2	0.82
0.3	0.756
0.4	0.7048
0.5	0.66384
0.6	0.631072
0.70000005	0.604858
0.8000001	0.583886
0.9000001	0.567109

Code in processing(Task_1_a):

```
float x0,y0,x,y,h,n;//real numbers for all variables
x0 = 0;//start point of x
y0 = 1;//start point of y
n = 100;//number of points
h = 0.1;//step
x=x0;
y=y0;
println("t; f(t) ");//collumn with time t and function f(t)
println(x +"; " +y);//first point with start coordinates
for (int i=1;i<n;i++)//Loop for all number of points
{
y=y+h*func(x,y);//Explicit formula of Euler's Method
x=x+h;//step for x
println(x +"; " +y);//Printing the aproximation's result, x is time t, y is function f
}
}
float func(float x,float y)//this is for all functions
{
float f;
f=1-2*y;//Function that is f' or (df/dt)
return f;
}
```

b) The same procedure for function **y'=y^2**: Firstly the plot from my program:



The **wolfram's** result:

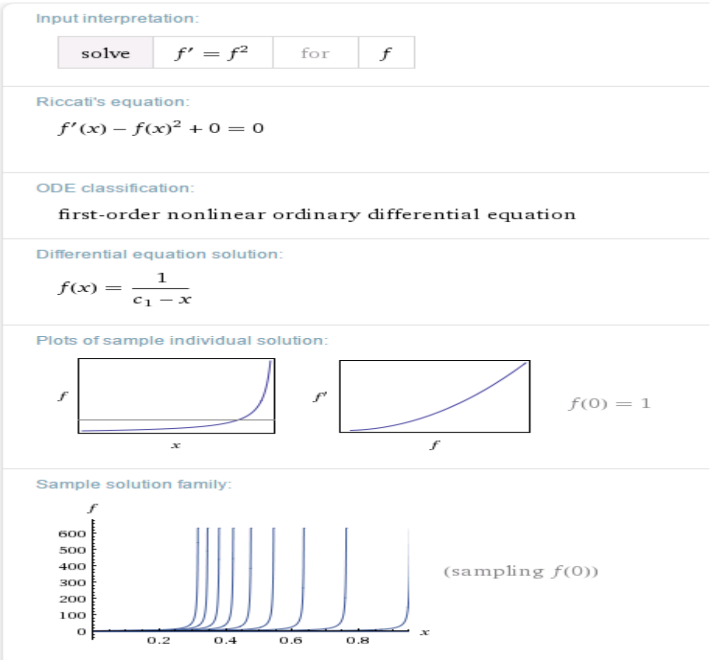


Table with first points:

t	f(t)
0	1
0.1	1.1
0.2	1.221
0.3	1.370084
0.4	1.557797
0.5	1.80047
0.6	2.12464
0.7	2.576049
0.8	3.239652

Approximation is close. The program is called "**Task_1_b**": The procedure is the same, just changing the function:

```
float func(float x,float y)//this is for all functions
{
float f;
f=y*y;//Function that is f' or (df/dt)
return f;
}
```

Exercise 1.2

a) Firstly I found the f(t1) by using Implicit **Euler's Method**:

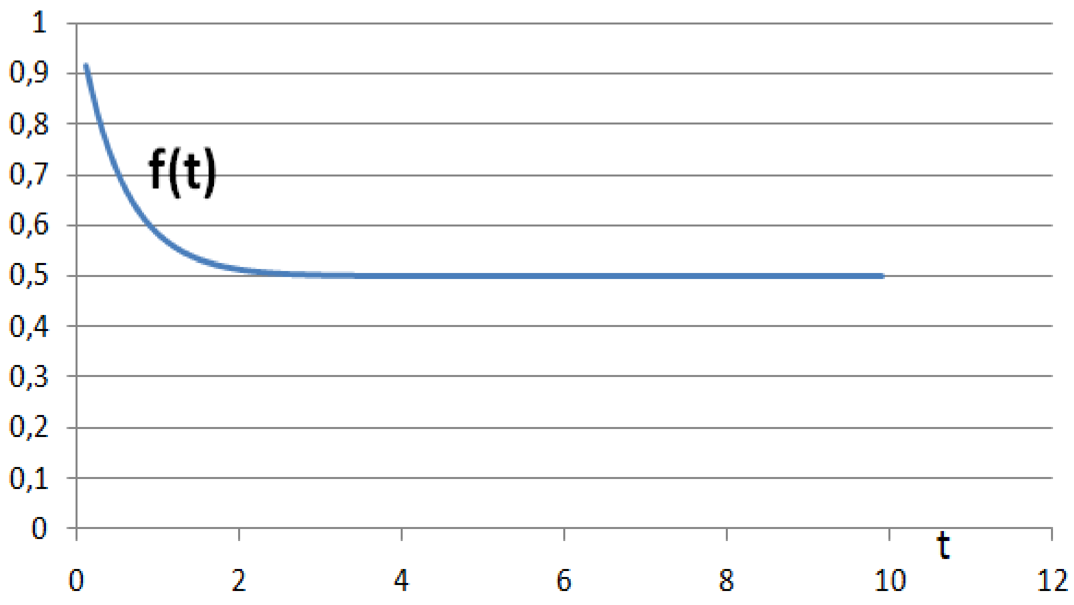
f(t1) =(1-2*f(t1))*h+f(t0);

f(t1) = (h+f(t0))/(1+2*h);

this result I put in my formula. Table of first 10 points:

t	f(t)
0.00	1
0.1	0.916667
0.2	0.847222
0.3	0.789352
0.4	0.741127
0.5	0.700939
0.6	0.667449
0.70000005	0.639541
0.8000001	0.616284
0.9000001	0.596903

And the graph:



The program for a) part is called **"Task_2_a".**

Code:

```
void setup()
{
float x0,y0,x,y,h,n;//variables

x0 = 0;//start point of x
y0 = 1;//start point of y
n = 100;//number of points
h = 0.1;//step

x=x0;
y=y0;

println("t; f(t) ");//collumns t is a time, f(t) is a function
println(x +"; " +y);//printing the first point

for (int i=1;i<n;i++)
{
  x=x+h;//step for x
  y=(y+h)/(1+2*h);//Euler's Implicit method

println(x +"; " +y);//results
}


}
```

b) The same procedure for function **y'=y^2**;

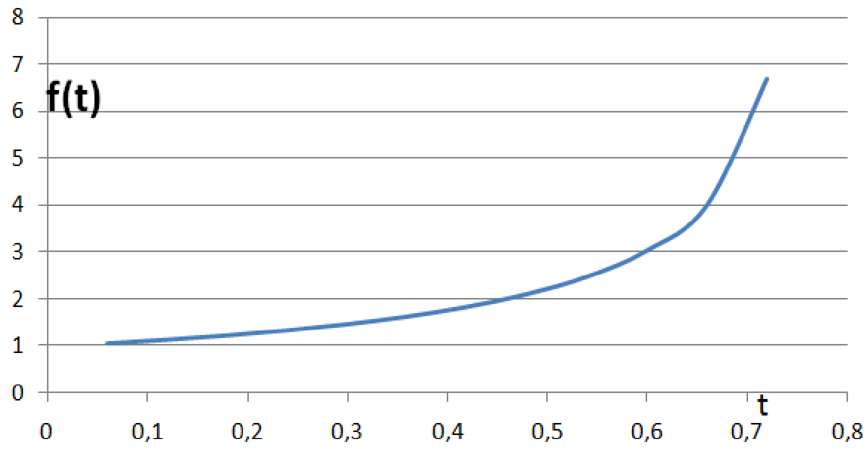
f(t1) = f^2(t1)*h+f(t0);

f^2(t1)*h-f(t1)+f(t0)=0;

After solving this equation I got the result for **f(t1) = (1-sqrt(1-4*h*f(t0)))/(2*h);**

And the table and graph result:

T	f(t)
0.00	1
0.06	1.068502
0.12	1.147508
0.17999999	1.239723
0.24	1.348894
0.29999998	1.480387
0.35999998	1.642195
0.42	1.846845
0.48	2.11532
0.53999996	2.486187
0.59999996	3.041076
0.65999997	4.002069
0.71999997	6.677043



As we see from the results, Implicit method is closer than explicit.

Code is called Task_2_b:

```
void setup()
{
float x0,y0,x,y,h,n;//variables

x0 = 0;//start of x
y0 = 1;//start of y
n = 14;//number of points
h = 0.06;//step

x=x0;
y=y0;

println("t; f(t) ");//collumn of time and function
println(x +"; " +y);//printing the first point

for (int i=1;i<n;i++)
{
  x=x+h;//step for x

  y=(1-sqrt(1-4*y*h))/(2*h);//the Euler's Implicit method formula

println(x +"; " +y);//printing results
}


}
```

Exercise 1.3

For this exercise I just modified my first code and add the function that will be valid also for second condition.

The table below shows the result with f(t) for the **first** system of functions in vector form:

t	vector f(t)
0	(1.0, 1.0)
0.1	(1.0, 0.9)
0.2	(0.99, 0.79999995)
0.3	(0.9702, 0.701)
0.4	(0.941094, 0.60397995)
0.5	(0.90345025, 0.5098705)
0.6	(0.85827774, 0.4195255)
0.7	(0.80678105, 0.33369774)
0.8	(0.75030637, 0.25301963)
0.9	(0.69028187, 0.17798899)

Code - "Task_3_a":

```
float x0,y0,x,y,h,n, yx,ynew ;//real numbers for all variables
x0 = 0;//start point of x
y0 = 1;//start point of y
n = 10;//number of points
h = 0.1;//step
x=x0;
y=y0;
yx = y0;
println("t; vector f(t) ");//collumn with time t and function f(x,y)
println(x +"; (" +y+", "+yx+")");//first point with start coordinates
for (int i=1;i<n;i++)//Loop for all number of points
{ynew=y;
y=y+h*funcx(x,y);//Explicit formula of x coordinate
yx = yx + h*funcy(x,ynew);//Explicit formula of y coordinate
x=x+h;//step for x
println(x +"; ("+y+", " + yx+")");//Printing the aproximation's result, x is time t, y is function f
}
}
float funcx(float x,float y)//this is for x
{
float[] f = {0,0};
float[] c1 = {1,0};
float[] c2 = {0,1};
f[0]=-(x*y);//Function for dx/dt
//f[1]=c1[1]*x-c2[1]*y;
//Function that is f' or (df/dt)
return f[0];
}
float funcy(float x,float y)//this is for y
{
float[] f = {0,0};
float[] c1 = {1,0};
float[] c2 = {0,1};
//f[0]=c1[0]*x-c2[0]*y;
f[1]=(-y);//Function for dy/dt
//Function that is f' or (df/dt)
return f[1];
}
```

Next table is for the **second** equation:

t	vector c
0	(1.0, 1.0)
0.1	(0.9, 1.0)
0.2	(0.81, 1.01)
0.3	(0.729, 1.03)
0.4	(0.6561, 1.06)
0.5	(0.59049, 1.0999999)
0.6	(0.531441, 1.1499999)
0.7	(0.47829688, 1.2099998)
0.8	(0.4304672, 1.2799999)
0.9	(0.38742048, 1.3599999)

code -"Task_3_b":

```
void setup() {
float x0,y0,x,y,h,n, yx,ynew ;//real numbers for all variables
x0 = 0;//start point of x
y0 = 1;//start point of y
n = 10;//number of points
h = 0.1;//step
x=x0;
y=y0;
yx = y0;
println("t; vector c ");//collumn with time t and function f(x,y)
println(x +"; (" +y+", "+yx+")");//first point with start coordinates
for (int i=1;i<n;i++)//Loop for all number of points
{ynew=y;
y=y+h*funcx(x,y);//Explicit formula of x coordinate
yx = yx + h*funcy(x,ynew);//Explicit formula of y coordinate
x=x+h;//step for x
println(x +"; ("+y+", " + yx+")");//Printing the aproximation's result, x is time t, y is function f
}
}
float funcx(float x,float y)//this is for x
{
float[] f = {0,0};
float[] c1 = {1,0};
float[] c2 = {0,1};
f[0]=x*c2[0]-y*c1[0];//Function for dx/dt
//Function that is f' or (df/dt)
return f[0];
}
float funcy(float x,float y)//this is for y
{
float[] f = {0,0};
float[] c1 = {1,0};
float[] c2 = {0,1};

f[1]=x*c2[1]-y*c1[1];//Function for dy/dt
//Function that is f' or (df/dt)
return f[1];
}
```

Exercise 1.4

In **mathematics** and **computational science**, **Heun's method** may refer to the improved or modified **Euler's method.**

$$\tilde{y}_{i+1} = y_i + hf(t_i, y_i)$$
$$y_{i+1} = y_i + \frac{h}{2}[f(t_i, y_i) + f(t_{i+1}, \tilde{y}_{i+1})],$$

where h is the step size and $t_{i+1} = t_i + h$.

(C) **Wikipedia**

1) **Code Task_4_a** :

```
void setup() {
float x0,y0,x,y,h,n,yx,ynext,yxnext,xprev;//real numbers for all variables
x0 = 0;//start point of x
y0 = 1;//start point of y
n = 10;//number of points
h = 0.1;//step
x=x0;
y=y0;
yx = y0;
println("t; vector f(t) ");//collumn with time t and function f(t)
println(x +"; (" +y+ ", " +yx+")");//first point with start coordinates
for (int i=1;i<n;i++)//Loop for all number of points
{
xprev = x;
x=x+h;//step for x
yxnext = yx + h*funcy(xprev,yx);//Heuns formula for y komponent
yx = yx + (h/2)*(funcy(xprev,yx)+funcy(x,yxnext));//Heuns formula for y

ynext=y+h*funcx(xprev,y);//Heuns formula for x komponent
 y=y+(h/2)*(funcx(xprev,y)+funcx(x,ynext));//Heuns method for x

println(x +"; (" +y+ ", " +yx+")");//Printing the aproximation's result, x is time t, y is vector from of
function f
}
}
float funcx(float x,float y)//this is for all functions
{
float[] f = {0,0};
float[] c1 = {1,0};
float[] c2 = {0,1};
f[0]=-x*y;//Function that is f' or (df/dt)
return f[0];
}
float funcy(float x,float y)//this is for all functions
{
float[] f = {0,0};
float[] c1 = {1,0};
float[] c2 = {0,1};
f[1]=(-y);//Function that is f' or (df/dt)
return f[1];
}
```

2) **Code of Task_4_b:**

```
void setup() {
float x0,y0,x,y,h,n,yx,ynext,yxnext,xprev;//real numbers for all variables

x0 = 0;//start point of x
y0 = 1;//start point of y
n = 10;//number of points
h = 0.1;//step

x=x0;
y=y0;
yx = y0;

println("t; f(t) ");//collumn with time t and function f(t)
println(x +"; (" +y+ "; " +yx+")");//first point with start coordinates

for (int i=1;i<n;i++)//Loop for all number of points
{
xprev = x;
x=x+h;//step for x
yxnext = yx + h*funcy(xprev,yx);//Heuns formula for y komponent
yx = yx + (h/2)*(funcy(xprev,y)+funcy(x,yxnext));

ynext=y+h*funcx(xprev,y);//Heuns formula for x komponent
y=y+(h/2)*(funcx(xprev,y)+funcx(x,ynext));



println(x +"; (" +y+ "; " +yx+")");//Printing the aproximation's result, x is time t, y is function f
}

}

float funcx(float x,float y)//this is for all functions
{
float[] f = {0,0};
float[] c1 = {1,0};
float[] c2 = {0,1};
f[0]=c2[0]*x-c1[0]*y;

//Function that is f' or (df/dt)
return f[0];
}

float funcy(float x,float y)//this is for all functions
{
float[] f = {0,0};
float[] c1 = {1,0};
float[] c2 = {0,1};
f[1]=c2[1]*x-c1[1]*y;

//Function that is f' or (df/dt)
return f[1];
}
```

Result table for the C vector:

t	vector C Heun's method	vector C Euler's method
0	(1.0, 1.0)	(1.0, 1.0)
0.1	(0.905, 1.005)	(0.9, 1.0)
0.2	(0.819025, 1.02)	(0.81, 1.01)
0.3	(0.7412176, 1.045)	(0.729, 1.03)
0.4	(0.67080194, 1.0799999)	(0.6561, 1.06)
0.5	(0.60707575, 1.1249999)	(0.59049, 1.0999999)
0.6	(0.54940355, 1.1799998)	(0.531441, 1.1499999)
0.7	(0.4972102, 1.2449999)	(0.47829688, 1.2099998)
0.8	(0.44997522, 1.3199999)	(0.4304672, 1.2799999)
0.9	(0.40722758, 1.405)	(0.38742048, 1.3599999)

The results are **approximately** equal**.**

Exercise 1.5

a) I used the **Euler's explicit method** for **A - stability.**

Code Task_5_a:

```
float k=1;//k in function
void setup()
{
double x0,y0,x,y,h,n,ynew,count;//real numbers for all variables

x0 = 0;//start point of x
y0 = 1;//start point of y
n = 300;//number of points(maximum for -k*y was 464)
h = 0.8;//step
x=x0;
y=y0;
count=1;
for (int i=1;i<n;i++)//Loop for all number of points
{ynew = y;

y=y+h*func(x,y);//Explicit formula of Euler's Method
if (y/ynew<1)//Condition for convergence
{
  count++;
}
x=x+h;//step for x
}
if (count == n)
{
  println("Integrator is A - stable");
}else
 println("Integrator is NOT A - stable");
}
double func(double x,double y)//this is for all functions
{
double f;
f=-k*y;//Function that is f' or (df/dt)
return f;
}
```

b) **A- stability** shows the numerical method's behavior is convergence to zero:

$$\lim_{n \to \infty} A(tn) = 0$$

A-stability does not show the instability problems.(For example for the big number of points)

(I will try to find the integrator....but I am thinking that it is Implicit, because it does not solve the equation)

Testing all 3 methods:

The comparison between all 3 methods

For the function **y'=2-e^(-4t)-2y**, the test for all 3 methods

t(time)	Explicit Euler's method	Implicit Euler's method	Heun's method	Exact value	Error Expl	Error Impl	Error Heun
0	1	1	1	1	0	0	0
0.1	0.9	0.9441399	0.926484	0.925795	0.025795	0.018345	0.000689
0.2	0.852968	0.91600585	0.8904376	0.889504	0.036536	0.026502	0.000934
0.3	0.8374415	0.9049054	0.877126	0.876191	0.03875	0.028714	0.000935
0.4	0.8398338	0.9039297	0.8771007	0.876191	0.036357	0.027739	0.00091
0.5	0.85167736	0.90866345	0.88438	0.883728	0.032051	0.024935	0.000652

As we see the minimal error is in **Heun's Method**.