Exercise 1

Task 1

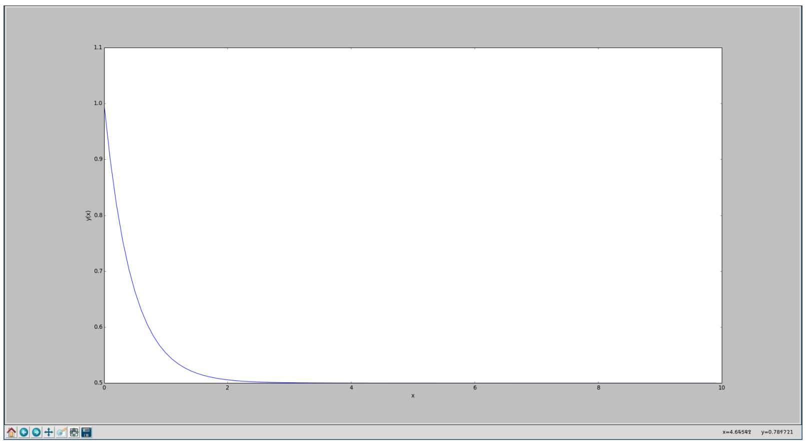
Explicit Euler method: for $y' = f(x,y)$ it holds that $y_1 = y_0 + \Delta x \cdot f(x_0, y_0)$

The program for this task has the name *explicit_euler.py*. But this script uses a library for symbolic computation (*sympy*) which you might not have installed on your computer. So there are also two scripts *explicit_euler_a.py* and *explicit_euler_b.py* which solve only the related subtasks with hardcoded formulae (They are providing also quality checks for my solutions compared to the exact solutions). Each script has a highlighted block for input parameters at the beginning.

a) $y' = 1 - 2 \cdot y$

$$\Rightarrow y_1 = y_0 + \Delta x \cdot (1 - 2 \cdot y_0)$$

My program outputs the following solution for the initial values $x_0 = 0$, $y_0 = 1$ and a step size $\Delta x = 0.1$:

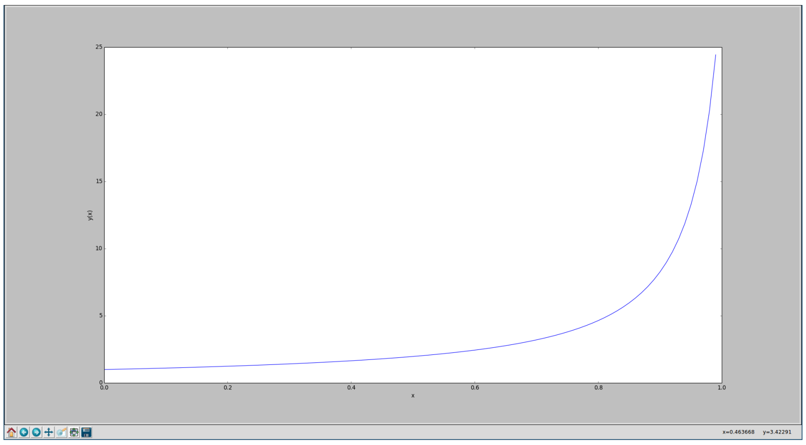


This already seems to be quite similar to the solution given by WolframAlpha, that is $y(x) = c \cdot e^{-2 \cdot x} + \frac{1}{2}$. The program *explicit_euler_a.py* is also able to evaluate the quality of our approximation and outputs an average residual for the above example of: 0.00258. This seems pretty good for me for such a simple approximation but it depends strongly on the chosen step size.

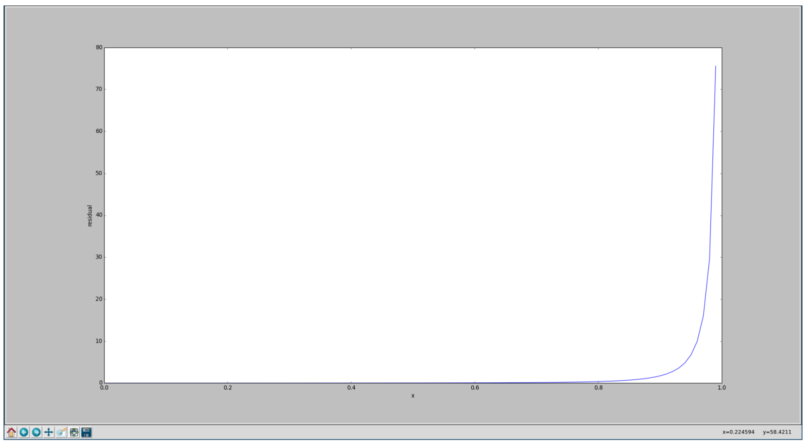
b) $y' = y^2$

$$\Rightarrow y_1 = y_0 + \Delta x \cdot y_0^2$$

My program outputs the following solution for the initial values $x_0 = 0$, $y_0 = 1$ and a step size $\Delta x = 0.01$:



This curve already has the same shape as the solution given by WolframAlpha, that is $y(x) = \frac{1}{c-x}$. The program *explicit_euler_b.py* is able to evaluate the quality of our approximation and outputs an average residual for the above example of: 1.63876. The development of the residual is depicted below. At first our approximation seems to be pretty good. But as the slope gets extremely large our approximation gets quickly worse.



Task 2

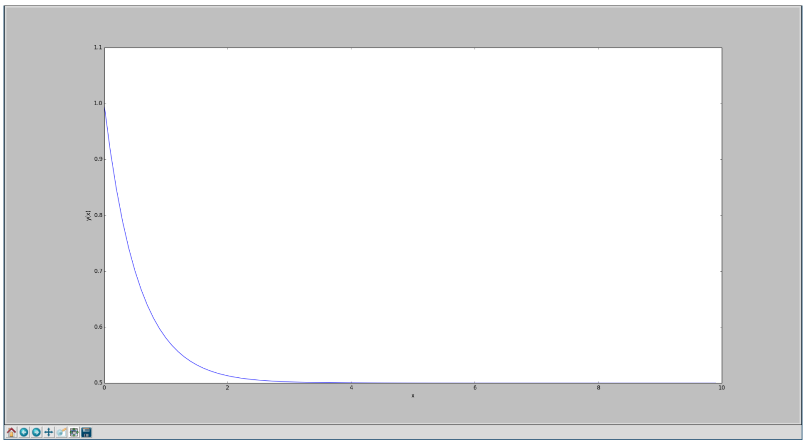
Implicit Euler method: for $y' = f(x, y)$ it holds that $y_1 = y_0 + \Delta x \cdot f(x_1, y_1)$

Again, the *implicit_euler.py* script is a generic solution and there are two hardcoded solutions if you are not able to run this script.

a) $y' = 1 - 2 \cdot y$

$$\begin{aligned} & y_1 = y_0 + \Delta x \cdot (1 - 2 \cdot y_1) \\ \Leftrightarrow\ & y_1 + 2 \cdot \Delta x \cdot y_1 = y_0 + \Delta x \\ \Leftrightarrow\ & y_1 = \frac{y_0 + \Delta x}{1 + 2 \cdot \Delta x} \end{aligned}$$

My program outputs the following solution for the initial values $x_0 = 0$, $y_0 = 1$ and a step size $\Delta x = 0.1$:



The program *implicit_euler_a.py* can measure the quality of our approximation and has output an average residual for the above example of: 0.00242. So the solution is nearly the same as previously when using the explicit method.

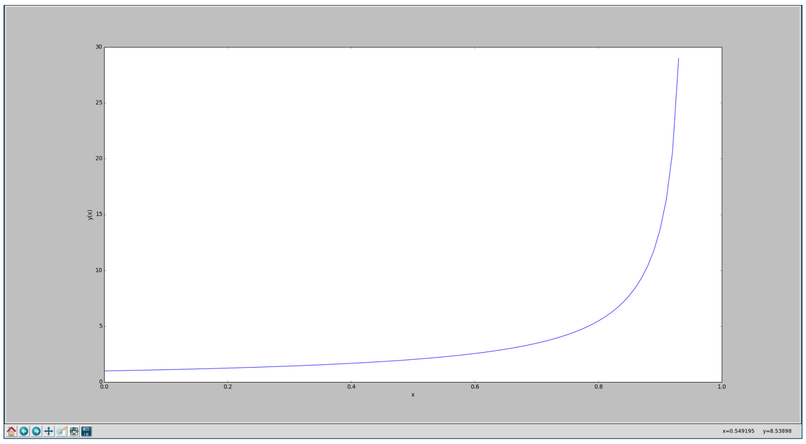
b) $y' = y^2$

$$\begin{aligned} y_1 &= y_0 + \Delta x \cdot y_1^2 \\ \Leftrightarrow y_1^2 &- \frac{1}{\Delta x} y_1 + \frac{y_0}{\Delta x} = 0 \\ \Leftrightarrow y_1 &= \frac{1}{2 \cdot \Delta x} \pm \sqrt{\frac{1}{4 \cdot \Delta x^2} - \frac{y_0}{\Delta x}} \end{aligned}$$

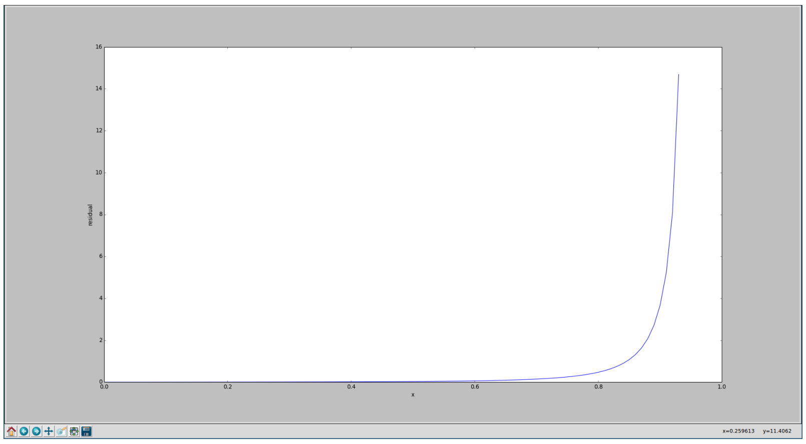
Here, we have the special case that there are two possible solutions for y_1, since we had to solve quadratic equation. If we again consider the calculation rule for the implicit Euler method, $y_1 = y_0 + \Delta x \cdot f(x_1, y_1)$, it tells us that y_1 converges to y_0 if $\Delta x \to 0$. We can rewrite the above equation to:

$$y_1 = \frac{1 \pm \sqrt{1 - 4 \cdot \Delta x \cdot y_0}}{2 \cdot \Delta x}$$

Now, we can simply see that for $\Delta x \to 0$ the solution with a + wouldn't make much sense. I have to admit that I do not know a generic approach for this problem with several possible solutions and therefore my program uses just one of the solutions without a subtle logic behind.
My program outputs the following solution for the initial values $x_0 = 0$, $y_0 = 1$ and a step size $\Delta x = 0.01$:



The average residual provided by *implicit_euler_b.py* is: 0.51072. This seems to be quite better than the solution provided by the explicit method but this time there arose complex numbers due to the square root, thus the last samples were omitted which were those with the largest residual in the explicit approach. The development of the residual is depicted below. As we can see, the development of the residual has worsened.



Task 3

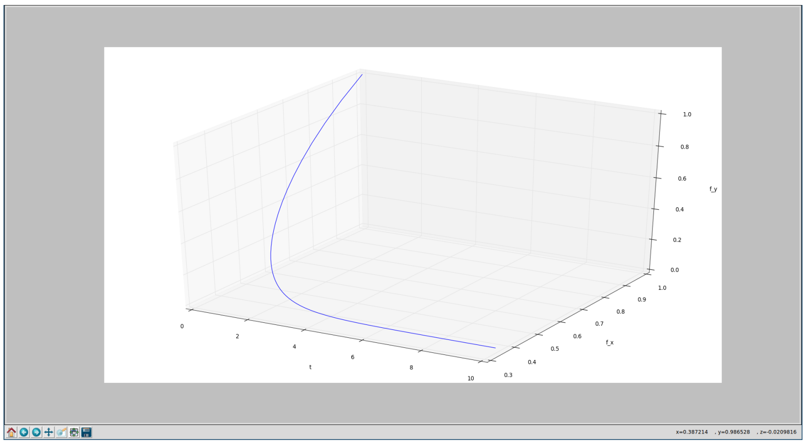
Explicit Euler method: for $f' = g(f,t)$ it holds that $\begin{pmatrix} f_{x,1} \\ f_{y,1} \end{pmatrix} = \begin{pmatrix} f_{x,0} \\ f_{y,0} \end{pmatrix} + \Delta t \cdot g(f_0, t_0)$

The program related to this task is *explicit_euler_2D.py*

a) For the first differential equation our calculation rule is:

$$\begin{pmatrix} f_{x,1} \\ f_{y,1} \end{pmatrix} = \begin{pmatrix} f_{x,0} \\ f_{y,0} \end{pmatrix} + \Delta t \cdot \begin{pmatrix} -f_{x,0} \cdot f_{y,0} \\ -f_{y,0} \end{pmatrix}$$

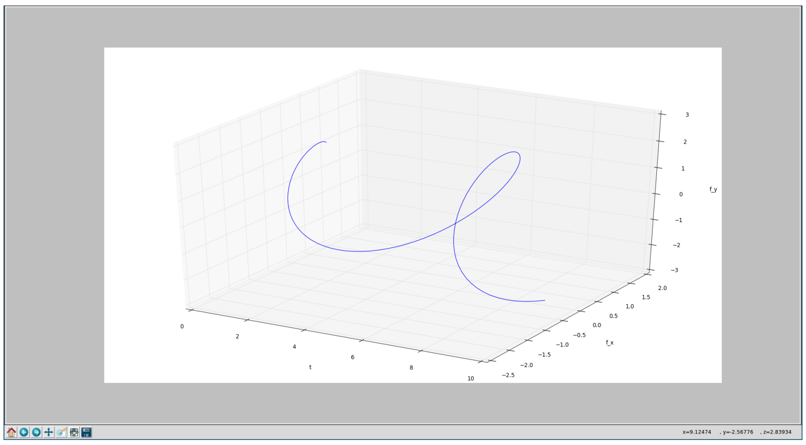
The solution for the initial values $t_0 = 0,\ f_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and a step size $\Delta x = 0.1$ looks as follows:



For the second differential equation our calculation rule is:

$$\begin{pmatrix} f_{x,1} \\ f_{y,1} \end{pmatrix} = \begin{pmatrix} f_{x,0} \\ f_{y,0} \end{pmatrix} + \Delta t \cdot \left[\begin{pmatrix} 0 \\ 1 \end{pmatrix} \cdot f_{x,0} - \begin{pmatrix} 1 \\ 0 \end{pmatrix} \cdot f_{y,0} \right] = \begin{pmatrix} f_{x,0} \\ f_{y,0} \end{pmatrix} + \Delta t \cdot \begin{pmatrix} -f_{y,0} \\ f_{x,0} \end{pmatrix}$$

The solution for the initial values $t_0 = 0,\ f_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and a step size $\Delta x = 0.1$ looks as follows:



b) If we consider the differential equation $f' = \begin{pmatrix} -f_y & f_x \end{pmatrix}$ we can simply deduce that our solution has to include cos and sin functions. We just have to think of different constants that might vanish or stay unchanged during the derivation. We can try the following ansatz:

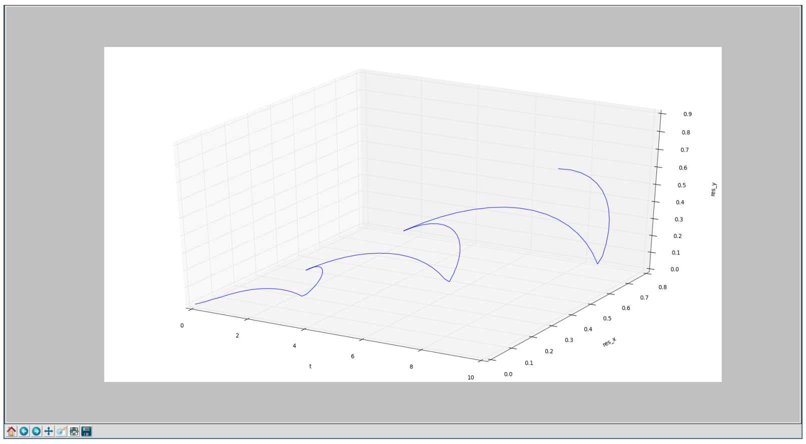
$$\begin{pmatrix} f_x \\ f_y \end{pmatrix} = c \cdot \begin{pmatrix} \cos(t+d) \\ \sin(t+d) \end{pmatrix}$$

$$\Rightarrow \frac{\partial}{\partial t}\begin{pmatrix} f_x \\ f_y \end{pmatrix} = c \cdot \begin{pmatrix} -\sin(t+d) \\ \cos(t+d) \end{pmatrix} = \begin{pmatrix} -f_y \\ f_x \end{pmatrix}$$

Now, we can calculate the constants c and d for the given initial values t_0 and $f_0 = \begin{pmatrix} f_{x,0} \\ f_{y,0} \end{pmatrix}$

$$\begin{aligned}
&\begin{pmatrix} f_{x,0} \\ f_{y,0} \end{pmatrix} = c \cdot \begin{pmatrix} \cos(t_0+d) \\ \sin(t_0+d) \end{pmatrix}, \qquad e = t_0 + d \\
&\Leftrightarrow f_{x,0} = c \cdot \cos(e) \text{ and } f_{y,0} = c \cdot \sin(e) \\
&\Leftrightarrow \frac{f_{x,0}}{\cos(e)} = \frac{f_{y,0}}{\sin(e)} \\
&\Leftrightarrow \frac{f_{x,0}}{f_{y,0}} \cdot \sin(e) = \cos(e), \qquad f = \frac{f_{x,0}}{f_{y,0}} \\
&\Leftrightarrow f \cdot \sin(e) = \sqrt{1 - \sin^2(e)} \\
&\Leftrightarrow f^2 \cdot \sin^2(e) = 1 - \sin^2(e) \\
&\Leftrightarrow \sin^2(e) = \frac{1}{1+f^2} \\
&\Leftrightarrow e = \arcsin\left(\sqrt{\frac{1}{1+f^2}}\right) \\
&\Leftrightarrow d = \arcsin\left(\sqrt{\frac{1}{1+\left(\frac{f_{x,0}}{f_{y,0}}\right)^2}}\right) - t_0 \\
&\Rightarrow c = \frac{f_{x,0}}{\cos(t_0+d)} = \frac{f_{x,0}}{\cos\left[t_0 + \arcsin\left(\sqrt{\frac{1}{1+\left(\frac{f_{x,0}}{f_{y,0}}\right)^2}}\right) - t_0\right]}
\end{aligned}$$

If my calculations are right then the explicit Euler method does not provide a good approximation for this function. As we can see below, the residual is growing over time.



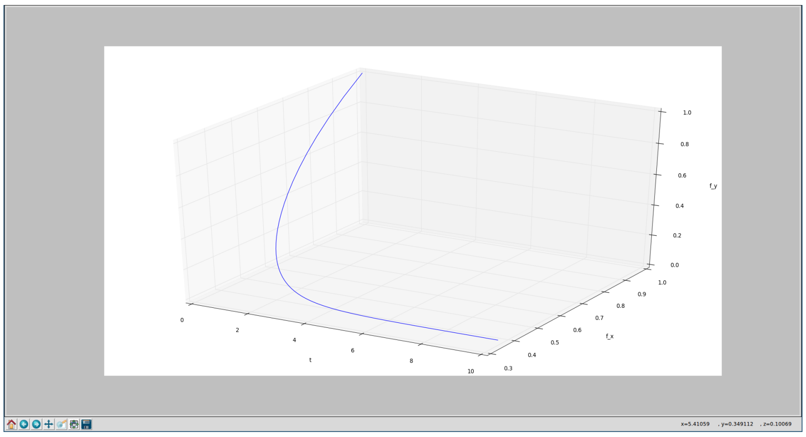
Task 4

Heun method: for $f' = g(f,t)$ and $\begin{pmatrix} f^*_{x,1} \\ f^*_{y,1} \end{pmatrix} = \begin{pmatrix} f_{x,0} \\ f_{y,0} \end{pmatrix} + \Delta t \cdot g(f_0, t_0)$ it holds that

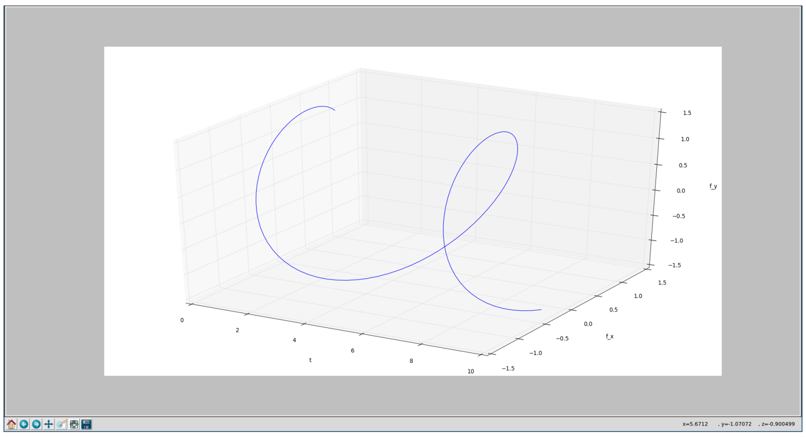
$$\begin{aligned}
\begin{pmatrix} f_{x,1} \\ f_{y,1} \end{pmatrix} &= \begin{pmatrix} f_{x,0} \\ f_{y,0} \end{pmatrix} + \frac{1}{2} \cdot \Delta t \cdot \left(g(f_0, t_0) + g(f^*_1, t_1)\right) \\
&= \frac{1}{2} \cdot \begin{pmatrix} f_{x,0} \\ f_{y,0} \end{pmatrix} + \frac{1}{2} \cdot \left(\begin{pmatrix} f^*_{x,1} \\ f^*_{y,1} \end{pmatrix} + \Delta t \cdot g(f^*_1, t_1)\right)
\end{aligned}$$

The program related to this task is *heun_2D.py*.

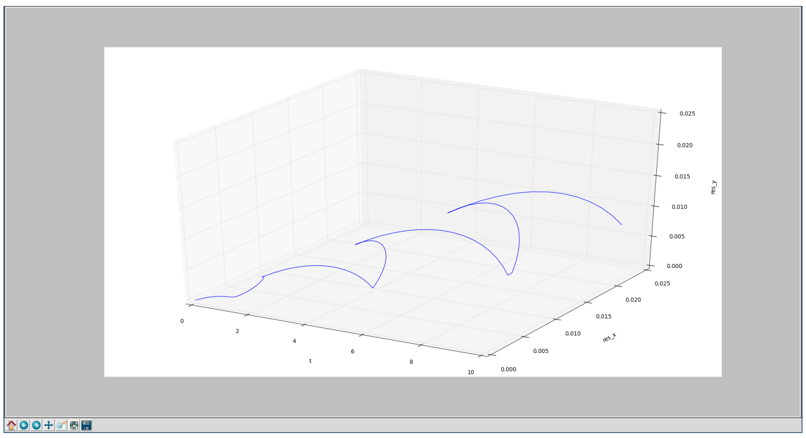
a) For the first differential equation the solution for the initial values $t_0 = 0,\ f_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and a step size $\Delta x = 0.1$ looks as follows:



For the second differential equation the solution for the initial values $t_0 = 0,\ f_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and a step size $\Delta x = 0.1$ looks as follows:



The approximation for the second equation is much better than in task 3. The development of the residual is depicted below.



Task 5

a) The A stability is just criteria to assure that integrators show the expected behaviour for a certain type of differential equations, i.e. the equation $f' = -kf$. This equation obviously has the solution $f(t) = c \cdot e^{-k \cdot t}$. Thus it will converge to 0 for all $k > 0$ if $t \to \infty$. To become more familiar with this concept, we will consider a few example.

Explicit Euler method

$$\begin{aligned}
f_1 &= f_0 + \Delta t \cdot (-k \cdot f_0) \\
&= f_0 \cdot (1 - \Delta t \cdot k) \\
f_2 &= f_1 \cdot (1 - \Delta t \cdot k) \\
&= f_0 \cdot (1 - \Delta t \cdot k)^2 \\
&\dots \\
f_i &= f_0 \cdot (1 - \Delta t \cdot k)^i
\end{aligned}$$

As we can see, $\lim\limits_{i \to \infty} f_i \to 0$ if and only if $|1 - \Delta t \cdot k| < 1$. So a good A stability tester would probe a k such that $|1 - \Delta t \cdot k| \geq 1$ for a fixed step size.

Implicit Euler method

$$\begin{aligned}
f_1 &= f_0 + \Delta t \cdot (-k \cdot f_1) \\
\Leftrightarrow f_1 &= \frac{f_0}{1 + \Delta t \cdot k} \\
f_2 &= \frac{f_1}{1 + \Delta t \cdot k} \\
&= \frac{f_0}{(1 + \Delta t \cdot k)^2} \\
&\dots \\
f_i &= \frac{f_0}{(1 + \Delta t \cdot k)^i}
\end{aligned}$$

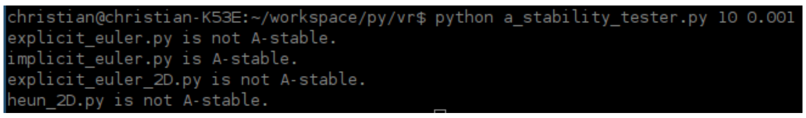
Since $\Delta t, k > 0$ this method is A stable for all k.

Heun method

$$\begin{aligned}
f_1 &= f_0 + \frac{1}{2} \cdot \Delta t \Big((-k \cdot f_0) + (-k \cdot f_1^*) \Big) \\
&= f_0 + \frac{1}{2} \cdot \Delta t \Big((-k \cdot f_0) + (-k \cdot f_0 \cdot (1 - \Delta t \cdot k)) \Big) \\
&= f_0 \cdot \Big(1 + \frac{1}{2} \cdot \Delta t (-k - k + \Delta t \cdot k^2) \Big) \\
&= f_0 \cdot \Big(1 - k \cdot \Delta t + \frac{1}{2} \cdot \Delta t^2 \cdot k^2 \Big) \\
f_2 &= f_1 \cdot \Big(1 - k \cdot \Delta t + \frac{1}{2} \cdot \Delta t^2 \cdot k^2 \Big) \\
&= f_0 \cdot \Big(1 - k \cdot \Delta t + \frac{1}{2} \cdot \Delta t^2 \cdot k^2 \Big)^2 \\
&\dots \\
f_i &= f_0 \cdot \Big(1 - k \cdot \Delta t + \frac{1}{2} \cdot \Delta t^2 \cdot k^2 \Big)^i
\end{aligned}$$

Again, this method fulfils $\lim_{i\to\infty} f_i \to 0$ if and only if $\left|1 - k \cdot \Delta t + \frac{1}{2} \cdot \Delta t^2 \cdot k^2\right| < 1$.

We can sum up, that our A stability tester has to cover a wide range of values for k to be able to deliver good results. Additionally, we can take advantage of the property that $c \cdot e^{-k \cdot t}$ is strictly monotonic decreasing.
My A stability test program is *a_stability_test.py*. As parameters you have to specify a minimum number of trials (how many different k-values should be probed) and an epsilon value that defines when a approximated function value will be small enough to be treated as zero. The program is probing random k-values between 0 and the maximum float number. The output is depicted below:



```
christian@christian-K53E:~/workspace/py/vr$ python a_stability_tester.py 10 0.001
explicit_euler.py is not A-stable.
implicit_euler.py is A-stable.
explicit_euler_2D.py is not A-stable.
heun_2D.py is not A-stable.
```

b) As previously mentioned, the A stability is just a certain criteria to assure that integrators show the expected behaviour for a certain type of differential equations. It does not state anything about the quality of the approximations given by an integrator. Thus an integrator which approximates every function value by the constant 0 is A stable but does not provide a meaningful approximation for function in general.