1) Linear system of first ODEs

A: Apparently falls exponentially and independently of K2, hence the approach:
A' = -k1*A
B: Starts converging towards 1 for a low k2, then falls and falls even faster for higher k2:
B’ = A*k1 - B*k2
C: See A-argumentation
C’ = k2*C

with **A(0) = 1, B(0) = C(0) = 0**

2) Implementation of Euler and Heun

I implemented Euler and Heun as functions that take a function with parameters as arguments to iterate over every k2 and pass a lambda function as argument

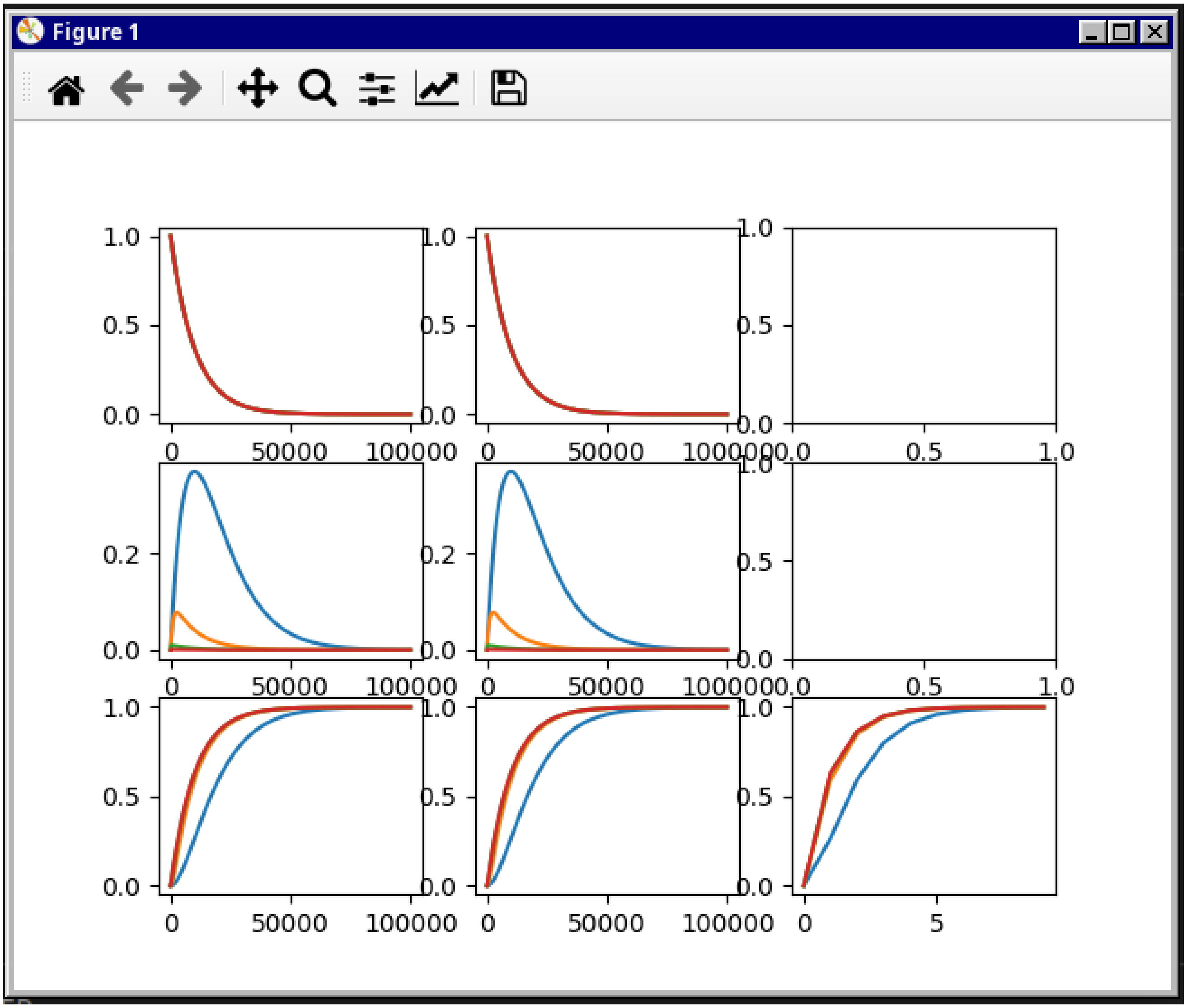
```
def euler(func, k2, startingValue, oldValues = range(iterations)):
    values = [];
    values.append(startingValue);
    for n in np.arange(1,iterations):
        f = values[n-1] + h*func(values[n-1],oldValues[n-1], k2);
        values.append(f);
    return values;

def Heun(func,k2, startingValue, oldValues=range(iterations)):
    values = [];
    values.append(startingValue);
    for n in np.arange(1,iterations):
        ystrich = values[n-1] + h*func(values[n-1],oldValues[n-1], k2);
        f = values[n-1] + h/2*(func(values[n-1],oldValues[n-1], k2) + func(ystrich,oldValues[n], k2));
        values.append(f);
    return values;
```

Note that every lambda function has 3 arguments in order to make this work, but only B actually takes 3 as an input

```
k2s = [1,10,100,1000];
Aprime = lambda A, y, z: -k1*A
Bprime = lambda B,A,k2: k1*A-k2*B;
Cprime = lambda C,B, k2: k2*B;
```

3) Visualization and Comparisation



The first column contains the Euler method, the second column the Heun method. In the third column in the last row, the analytical form of C is shown. Each color is a k2.

I didn't notice any great differences in runtime between Euler and Heun, but I have reason to believe that Heun would have to be a bit more accurate and might be a bit slower for large iterations because you have to calculate more terms.