Exercise No. 3

David Bubeck, Pascal Becht, Patrick Nisblè

May 12, 2017

1 2 - Three Body System

1.1 a)

Problem: Simulate a three-body system with given initial positions and velocities (y_1 to y_{12}) using the Runge-Kutta-Scheme of 4th order

Using the following implementation of the RK4 method:

```python
def rKN(x: np.ndarray, fx: List[Callable], hs: float):
    k1 = np.array(x)
    k2 = np.array(x)
    k3 = np.array(x)
    k4 = np.array(x)

    l = len(fx)

    for i in range(l):
        k1[i] = fx[i](x)*hs

    for i in range(l):
        k2[i] = fx[i](x + k1*.5)*hs

    for i in range(l):
        k3[i] = fx[i](x + k2*.5)*hs

    for i in range(l):
        k4[i] = fx[i](x + k3)*hs

    return x + (k1 + 2*(k2 + k3) + k4)/6
```

Figure 1: rKN-function for iterative use on previous calculated values

and the following set of equations:

$$\overrightarrow{\dot{x}_{n+1}} = \vec{v}_n \tag{1}$$

$$\overrightarrow{\dot{v}_{n+1}} = \frac{Gm_2}{r_{12,n}^2}\frac{\vec{r}_{12,n}}{r_{12,n}} + \frac{Gm_3}{r_{13,n}^2}\frac{\vec{r}_{13,n}}{r_{13,n}} \tag{2}$$

where $r_n = \|\vec{r}\|$, for m_1

```python
from rk4 import rKN
import numpy as np
import matplotlib.pyplot as plt

# calculating gravitational acceleration
def grav_acc(
    m1: float,
    x1: float,
    y1: float,
    m2: float,
    x2: float,
    y2: float) -> (float,float):

    G = 1 # m^3/kg/s^2

    r = np.array([x2-x1,y2-y1])
    a = G*m2/np.sqrt(r[1]**2+r[0]**2)**3*r
    return a

def three_body_sim(
        nsteps: int = 100,
        tmax: float = 1,
        init_val = np.ndarray(shape=(1,12))
    ) -> np.ndarray:

    # where (y_1+4i,y_2+4i) initial x,y coords of m_i and
    # (y_3+4i,y_4+4i) the initial x,y velocity of m_i
    # taken from exercise sheet no3

    xlist = np.ndarray(shape=(nsteps+1,12), dtype=float)
    xlist[0] = init_val


    # only dist dependent accelerations
    a12 = lambda x: grav_acc(1,x[0],x[1],1,x[4],x[5])
    a13 = lambda x: grav_acc(1,x[0],x[1],1,x[8],x[9])
    a23 = lambda x: grav_acc(1,x[4],x[5],1,x[8],x[9])

    # system of linear equations for three body system
    fsys = [
        lambda x: x[2],             # dx_1
        lambda x: x[3],             # dy_1
        lambda x: a12(x)[0] + a13(x)[0],  # dv_x1
        lambda x: a12(x)[1] + a13(x)[1],  # dv_y1

        lambda x: x[6],             # dx_2
        lambda x: x[7],             # dy_2
        lambda x: -a12(x)[0] + a23(x)[0], # dv_x2
        lambda x: -a12(x)[1] + a23(x)[1], # dv_y2

        lambda x: x[10],            # dx_3
        lambda x: x[11],            # dy_3
        lambda x: -a23(x)[0] - a13(x)[0], # dv_x3
        lambda x: -a23(x)[1] - a13(x)[1]  # dv_y3
    ]
```

```
    # runge-kutta using prev calculated values
    for i in range(nsteps):
        xlist[i+1] = rKN(xlist[i],fsys,tmax/nsteps)

    return xlist


if __name__=="__main__":

    xinit = np.array([[
        -.97000436,
        .24308753,
        -.46620368,
        -.43236573,
        .97000436,
        -.24308753,
        -.46620368,
        -.43236573,
        .0,
        .0,
        .93240737,
        .86473146
    ]])
    tmax = 2 # s
    plt.figure(figsize=(10.8,7.2))
    plt.xlabel('x')
    plt.ylabel('y')
    for stepsize in np.linspace(.01,.001, 3):
        xlist = three_body_sim(int(tmax/stepsize),tmax,xinit)
        print(xlist)
        plt.plot(xlist[:,0],xlist[:,1],label = '$m_1, \Delta t={:.3f}$'.format(
            stepsize))
        plt.plot(xlist[:,4],xlist[:,5],label = '$m_2, \Delta t={:.3f}$'.format(
            stepsize))
        plt.plot(xlist[:,8],xlist[:,9],label = '$m_3, \Delta t={:.3f}$'.format(
            stepsize))

    plt.legend()
    plt.savefig('3-21-plot.png', bbox_inches='tight')
    plt.show()
```

Figure 2: implementation of a three-body system using the RK4 method

where when done for stepsizes between 0.01 and 0.001 we can only see a difference in calulated length over multiple steps, when accounted for stepsize the calculated paths look alike. An animation was also created.

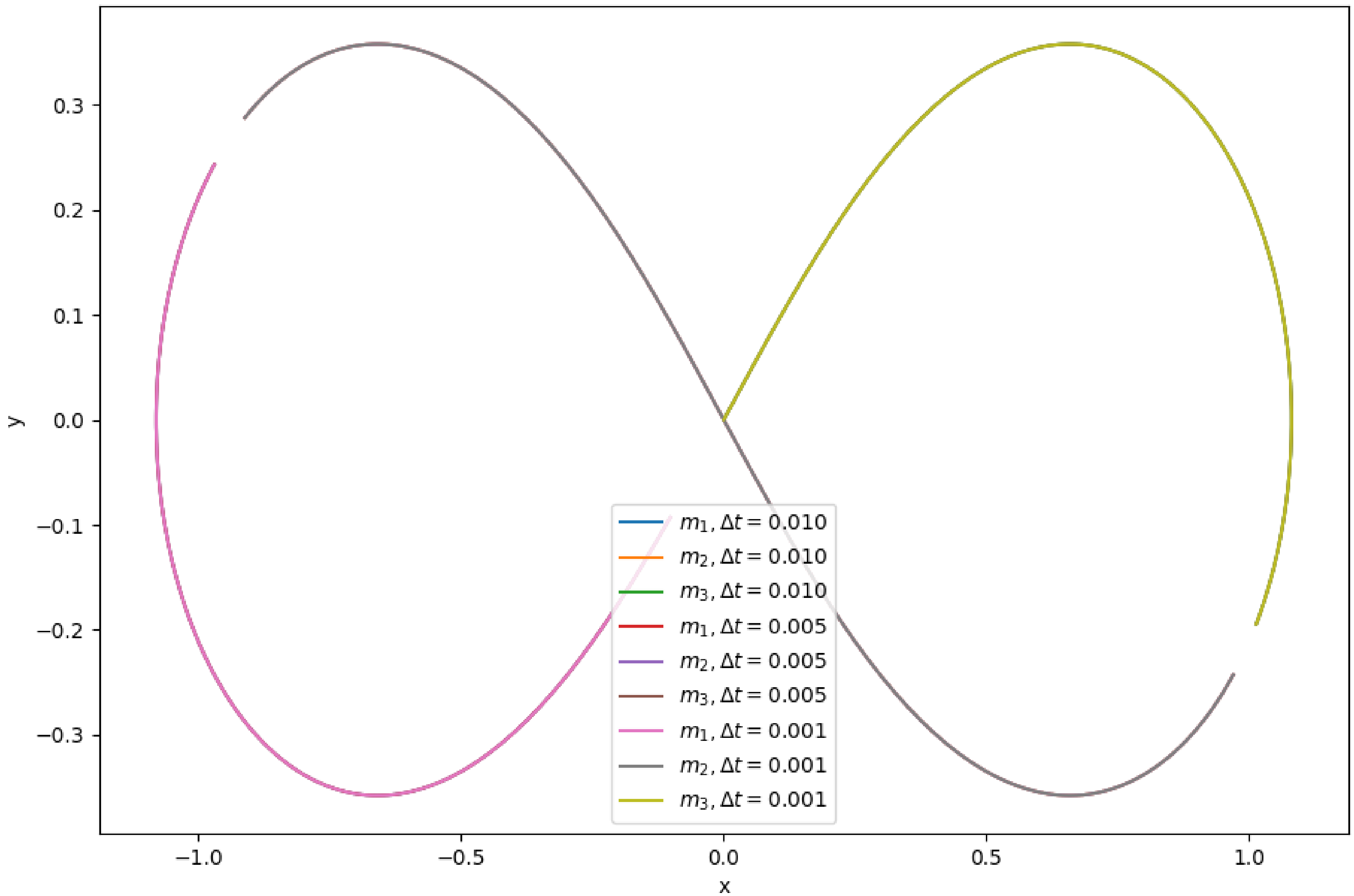

Figure 3: plot of all 3 masses and 3 different stepsizes