FYS-MEK1110 Oblig 2

Samuel Bigirimana

February 12, 2021

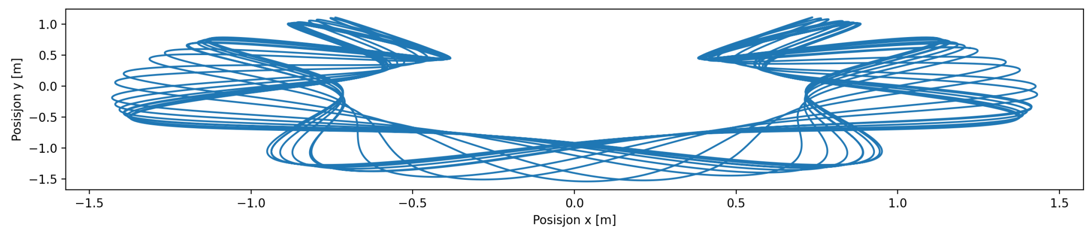
Euler-Cromer Method for the ball that hangs in a springified string

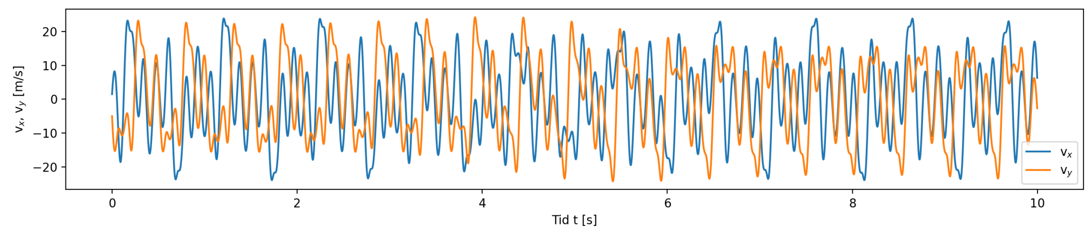
i) The program:

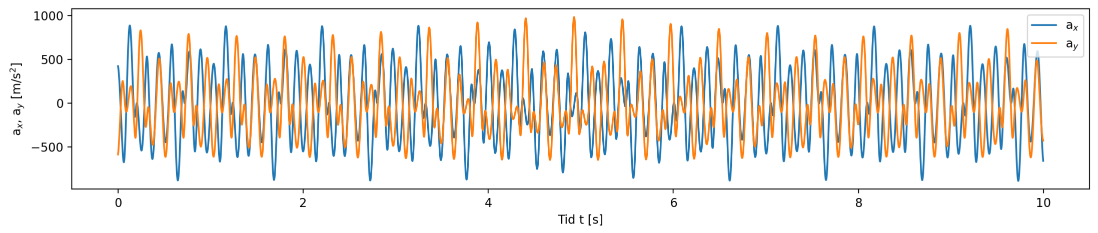
```
import numpy as np
import matplotlib.pyplot as plt
from math import sqrt, pi, sin, cos

# Defining the constants and variables:
g = 9.81         # m/s^2 gravitational accelaration
m = 0.1          # Weight of the ball
k = 200.0        # Spring constant

# Our initial values
y0 = 1.0                    # height in (y-direction), in m
x0 =  -y0*np.cos(np.pi/6) # posisjon in x-direction, defined to be
    zero in m
r0 = np.array([x0, y0])     # r-vektor, with sin x- og y-komponent
v0 = np.array([1.5, -5])    # We build the start velocity by t0 = 0
     s

time = 10 # maximum time we look at in seconds
dt = 0.001  # timesteps in seconds

# We use np.ceil function to decide the amount of elements we want
    to have in our vectors and matrises
n = int(np.ceil(time/dt))

# Defining a vector for the time t, and matrises for the position r
     (x- and y-component by a time t),
# the speed v (x - and y-component by the time t), and acceleration
     a (x- and y-component by a time t)
# n rows, two colums:
t = np.zeros(n)
r = np.zeros((n,2))
v = np.zeros((n,2))
a = np.zeros((n,2))

# Using our initial values:
r[0, :] = r0    # Start position for the r-vector
```

```
t[0] = 0      # Already desided in the definition above
v[0, :] = v0  # Initial velocity
a[0, :] = 0     # Initial acceleration

# Calculating with Euler-cromers method in a for-loop:
for i in range (0, n-1):
    Rx = r[i, 0]                # r-vector in x-direction
    Ry = r[i, 1]                # r-vector in y-direction
    R = np.sqrt(Rx**2 + Ry**2)  # r-vector using Rx and Ry
    theta = np.arctan(Ry/Rx)    # defining theta (the angle)
    Fnet = -g*np.array([0, 1]) - (k*(R - y0)) * (r[i, :]/R)     #
    Netforce calculation with all given initieal constants and
    variables

    a[i, :] = Fnet/m                        # acceleration in m/s^2
    v[i + 1, :] = v[i, :] + a[i, :]*dt      # velocity in m/s
    r[i + 1, :] = r[i, :] + v[i+1, :]*dt    # position in m
    t[i + 1] = t[i] + dt                    # time in seconds

print(r) # checking how r(t) looks like

# Ploting three graphs for position in x- and y-direction,
    acceleration in time and velocity in time:
fig, ax = plt.subplots(nrows=3, figsize=(10, 7))

# the ball position in x- and y-direction:
ax[0].plot(r[:i, 0], r[:i, 1])
ax[0].set_xlabel("Posisjon x [m]")
ax[0].set_ylabel("Posisjon y [m]")

# Building the two velocity components in time:
ax[1].plot(t[:i], v[:i, 0])
ax[1].plot(t[:i], v[:i, 1])
ax[1].set_xlabel('Tid t [s]')
ax[1].set_ylabel('v$_x$, v$_y$ [m/s]')
ax[1].legend(["v$_x$", "v$_y$"])    # explanation box

# Building the two acceleration components in time:
ax[2].plot(t[:i], a[:i, 0])
ax[2].plot(t[:i], a[:i, 1]) #,"." for markers
ax[2].set_xlabel('Tid t [s]')
ax[2].set_ylabel('a$_x$, a$_y$ [m/s$^2$]')
ax[2].legend(["a$_x$", "a$_y$"]) # explanation box

# Now that the plotting is done:
fig.tight_layout() # so things wont overwrite eachother
plt.show() # showing the plot
```

The graphs for the balls position, velocity and accelaration:







j) What we can see from figure 1 is the fluctuation in the motion in x- and y direction as the ball swings back and forward in 10 seconds. The wave motion describes how the string pulls the ball and how the acceleration and velocity work on the ball to pull it away from R = 0. Also to add: because of the stretch and pull on the rope in both directions, the fluctuation looks much bigger than it actually is. As L0 = 1m means the natural swing should be 2m, but because of the elasticity of the rope we can add 0.5m stretch. Witch will show that the ball can swing between a stretch of 3m. (hope I make my thought clear here :D)

The velocity and acceleration graphs shows how the forces change in direction. The numbers might look big on the graphs, but as you can see, they are very tight. This will be observed as vibrations on both the ball and the rope.

k) By changing the time intervals the program cant seem to have any good read of the motion before the 9th second. But when that happens the charts shoot right up and down with big fluctuations. I belive this happens because the R values comes too close to zero.

Also by observing the printed R-value by time-step 0.1 in x- and y directions; we can see that the the fluctuations start very small, but quickly grow very big.

By using the Euler-method I get a very clean read on the velocity and acceleration values, and how they grow in time. But the never stop growing and the y values are much bigger than the x-values.