PH3501 - 2a Euler Method

Claus-Dieter Ohl

October 13, 2016

1 Euler Method for Pathline Calculations

In the previous lecture we showed how to solve analyticaly for the pathline from a given velocity field $\mathbf{u}(\mathbf{x}, t)$. There we have to solve the ordinary differential equation (ODE)

$$\frac{\mathrm{d}\mathbf{X}}{\mathrm{d}t} = \mathbf{u}(\mathbf{x}, t) \tag{1}$$

.

This may be possible for some simple flow fields, for flow fields which are obtained experimentally or which are more complex (speak non-linear) we are left with an approximate solution of the ODE.

In this chapter we'll have a look into a numerical method to calculate the pathline using the Euler Method. It is based on the idea that differential can be approximated by a finite difference, i.e.

$$\frac{\mathrm{d}\mathbf{X}}{\mathrm{d}t} = \lim_{\Delta t \to 0} \frac{\mathbf{X}(t + \Delta t) - \mathbf{X}(t)}{\Delta t} \tag{2}$$

.

$$\frac{\mathrm{d}\mathbf{X}}{\mathrm{d}t} \simeq \frac{\mathbf{X}(t + \Delta t) - \mathbf{X}(t)}{\Delta t} = \frac{\mathrm{d}\mathbf{X}}{\mathrm{d}t} + O(\Delta t) \tag{3}$$

The very right hand side of above equation states that we make at maximum an error of order Δt by introducing the finite difference approximation. This fact can be derived using a Taylor series approximation.

The Euler method utilizes the observation of Euler that for small enough time steps Δt the differential equation can be approximated by the finite difference. Although we do not obtain the exact solution, we may be sufficiently close to it if the time step is sufficiently small (we keep it here with the vague formulation).

We can now write one forward time step in components

$$\begin{aligned}
X_1(t + \Delta t) &= X_1(t) + \Delta t\, u_1(X_1(t), X_2(t), X_3(t), t) && (1)\\
X_2(t + \Delta t) &= X_2(t) + \Delta t\, u_2(X_1(t), X_2(t), X_3(t), t) && (2)\\
X_3(t + \Delta t) &= X_3(t) + \Delta t\, u_3(X_1(t), X_2(t), X_3(t), t) && (3)
\end{aligned}$$

Before we start with implementing the Euler method we need to provide a function which returns the velocity at the position x. We define this function velocity(x1,x2,t) below.

The velocity field we want to plot is

$$u_1 = gx_1 - \omega x_2 \quad (4)$$
$$u_2 = -gx_1 + \omega x_2 \quad (5)$$

where g and ω are constants.

```
#standard boilerplate for Python
%matplotlib inline
import numpy as np
import matplotlib.pylab as plt
import math

#This is our velocity function
def velocity(x1, x2, t):
  omega = 1
  g = 0.5
  u1 = g*x1-omega*x2
  u2 = -g*x2+omega*x1

  return u1, u2
```

Now we put the Euler Method to work:

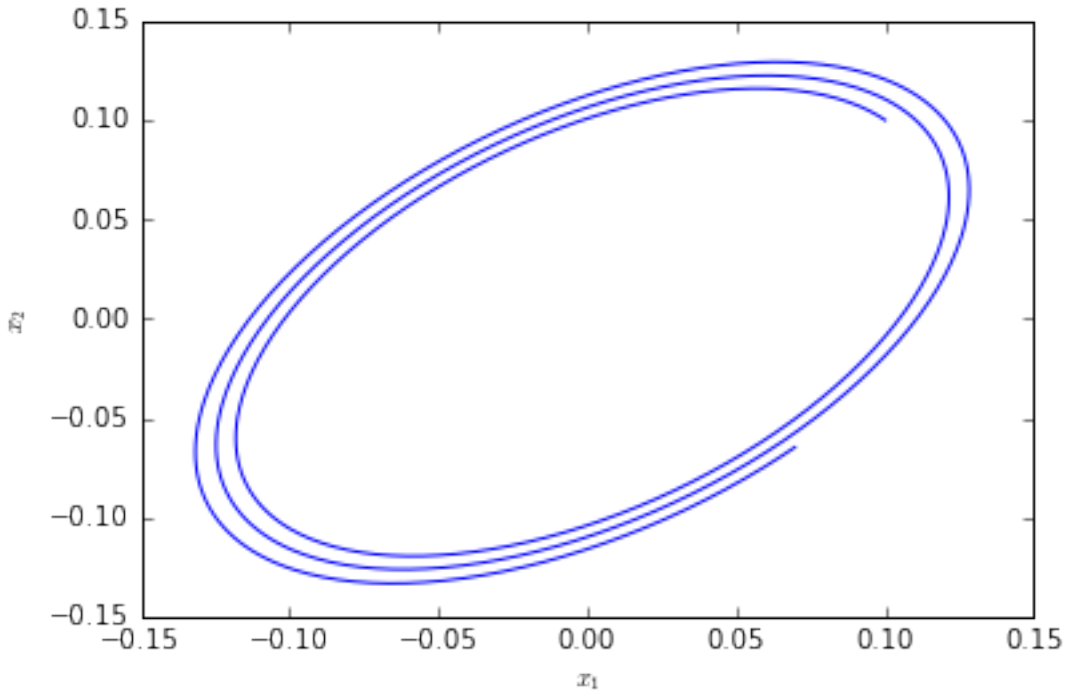
```
nt = 1000 #number of steps
deltat = 0.02 #step size

x10 = 0.1 #initial position
x20 = 0.1
x1 = np.zeros(nt) #thats where we store the result
x2 = np.zeros(nt)
x1[0] = x10  #insert the initial conditions
x2[0] = x20

for n in range(nt-1):
  t = deltat*n
  u1, u2 = velocity(x1[n],x2[n],t) #get velocity at that position

  #This is the Euler step
  x1[n+1] = x1[n]+deltat*u1 #update the position
  x2[n+1] = x2[n]+deltat*u2

plt.plot(x1,x2);
plt.xlabel('$x_1$'),plt.ylabel('$x_2$');
```

The flow field should return closed pathlines, you can check by reducing the time steps that better approximations are achieved. The Euler method assumes that the velocity is not changing during the step, yet the velocity is a continous function of space and time. Therefore the step size needs to be small, to keep also the these changes in the velocity field small. We can now improve the Euler method, by using not the velocity of the initial point in space $\mathbf{x}(t)$, but by using the average of the velocity at point $\mathbf{x}(t + \Delta t)$. Thus we make a step with

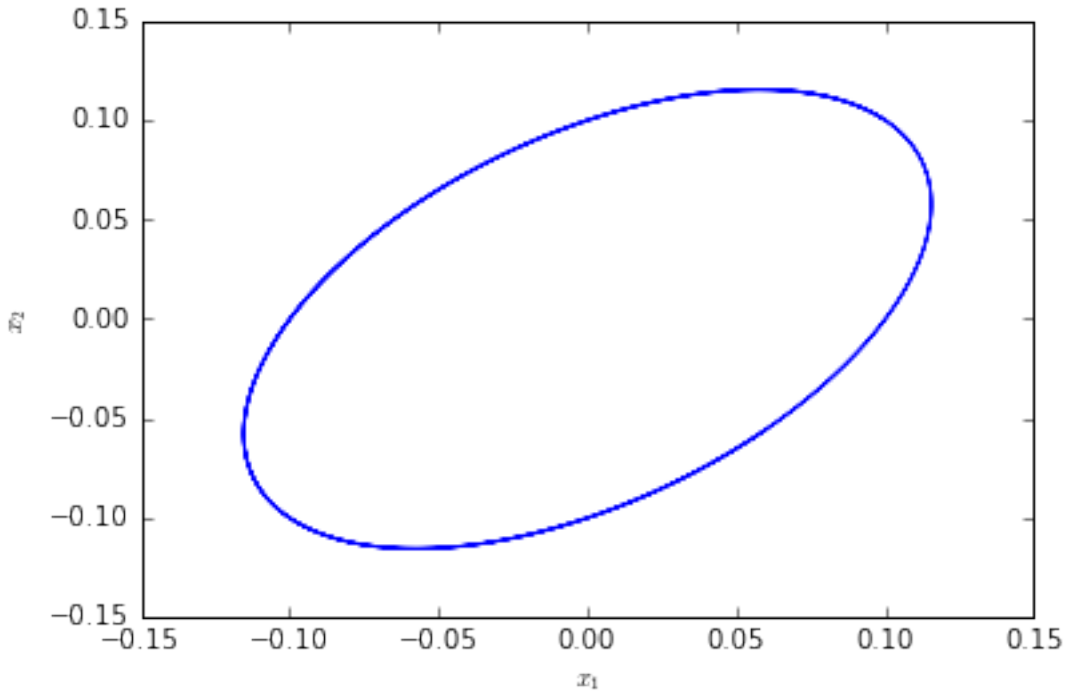
$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \Delta t\, \mathbf{u}_{avg} \quad (9)$$

where the average velocity is

$$\mathbf{u}_{avg} = \frac{1}{2} \left(\mathbf{u}(\mathbf{x}, t) + \mathbf{u}(\mathbf{x}, t + \Delta t) \right). \quad (10)$$




```
nt=1000 #number of steps
deltat=0.02 #step size

x10=0.1 #initial position
x20=0.1
x1=np.zeros(nt) #thats where we store the result
x2=np.zeros(nt)
x1[0]=x10  #instert the initial conditions
x2[0]=x20

for n in range(nt-1):
  t=deltat*n
  u1,u2=velocity(x1[n],x2[n],t) #get velocity at that position

  #what would be the velocity at the new position
  x1_pred=x1[n]+deltat*u1
  x2_pred=x2[n]+deltat*u2
  u1_pred,u2_pred=velocity(x1_pred,x2_pred,t)

  #a better approximation is the averaged velocity
  u1_avg=.5*(u1+u1_pred)
  u2_avg=.5*(u2+u2_pred)

  #Do the Euler step with the averaged velocity
  x1[n+1]=x1[n]+deltat*u1_avg
  x2[n+1]=x2[n]+deltat*u2_avg


plt.plot(x1,x2);
plt.xlabel('$x_1$'),plt.ylabel('$x_2$');
```



Now we achieve closed path lines with the same time stepping but at the costs of one more Euler step per time step.

Out[4]: <IPython.core.display.HTML object>