Finite Elements for Fluid Dynamics
Lab 1: Python & analytical solutions

Reidmen Aróstica, David Nolte, Cristóbal Bertoglio

February 5, 2021

1. Installation of the `Python` working environment

For this course the `FeNiCS` library of finite elements is required. The simplest way to do is to install it via *virtual containers*, as *Docker*. Installing the container for `FeNiCS` has also the advantage of allowing to work with `Jupyter notebooks`, a friendly environment to write `Python` codes and to present the results, where you can also run `FeNiCS`. This will be useful in the discussions of each lab.

Therefore, please read the guide under this link we have prepared about the installation of *Docker*, the `FeNiCS` container and the usage of `Jupyter notebooks`. If you have any questions please contact the TA.

2. Exercises

The exercises are for learning and practicing, and will not be evaluated in the discussion, but the assignments assume that you have done the exercises.

The exercises and assignments require the usage of two popular `Python` packages, `NumPy` and `Matplotlib`, which we will introduce before the exercises.

2.1. NumPy: arrays and matrices

To quote the official website:

"NumPy is the fundamental package for scientific computing with Python."

It offers the powerful n-dimensional `array` object and useful functions for linear algebra, Fourier transform, random numbers.

NumPy's arrays are implemented in C, and much more efficient than Python lists.

Please read the chapter *'The Basics'* of the official NumPy quickstart tutorial (`https://docs.scipy.org/doc/numpy/user/basics.html`)!

For those familiar with MATLAB there is a 'NumPy for MATLAB-users' guide: `https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html`

2.2. Matplotlib: plotting graphs

Matplotlib is a library that enables plotting similar to Octave or Matlab in Python.

An example for plotting a simple graph:

```python
import matplotlib.pyplot as plt
import numpy as np

# Data for plotting
x = np.arange(0.0, 2.0, 0.04)
y = 1 + np.sin(2 * np.pi * x)

# Plot
# optional: enable 'interactive' plotting
plt.ion()

fig = plt.figure()
plt.plot(x, y, '-o')

plt.xlabel('time (s)')
plt.ylabel('voltage (mV)')
plt.title('About as simple as it gets, folks')
plt.legend(['signal'])
plt.grid()

fig.savefig('test.png')
plt.show()
```

This example, and more, can be found on the official website.

2.3. Linear regression

Suppose we have a large set of measurement data $(x_i, y_i)_{0\leq i\leq N}$ and want to analyze the dependence of y_i on x_i. x_i is a controlled and perfectly known input. y_i is the result of an experiment observed for the input x_i and suffers from measurement errors, e_i.

The task is to compute a linear fit through all given data points via simple linear regression, i.e., the model $y = \alpha x + \beta$, where α and β need to be determined. Given (x_i, y_i), the slope of the regression line can be computed via linear least squares as

$$\alpha = \frac{\sum_{i=0}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=0}^{N}(x_i - \bar{x})^2}$$

with the mean values $\bar{x} = \frac{1}{N}\sum_{i=0}^{N} x_i$ and $\bar{y} = \frac{1}{N}\sum_{i=0}^{N} y_i$. The intercept is given by

$$\beta = \bar{y} - \alpha\bar{x}.$$

Without using NumPy array functions:

1. Generate the data as follows: $x_i = \frac{i}{N}$ for $i = 0, ..., N$ and $y_i = x_i + e_i$, where the error e_i is normally distributed with zero mean and standard deviation σ (use the function `gauss` from the package `random`). Store x_i and y_i in lists. Generate two plots with two different values of σ, $\sigma = 0.2$ and $\sigma = 0.4$. Add axis labels, legend and title. Use here $N = 30$.

2. Write a function that takes these data as arguments and computes the linear regression using the formulas above. Plot the points (x_i, y_i), the regression line, and the line $y = x$ for $N = 30$ and $\sigma = 0.4$. Add axis labels, legend and title.

3. Verify that the regression line converges to the line $y = x$ when $N \to \infty$ and when $\sigma \to 0$.

Using ONLY NumPy array functions:

1. Write a second function that implements the linear regression more efficiently **by only using NumPy arrays**. Use the same generated data, but converted to NumPy arrays before, and verify that the results are exactly the same as without python arrays.

2. Compare the speed of both versions. Create a big data set (`N = int(2e6)`), and call each of the not-NumPy and NumPy using functions using `time.time()`. Verify that the NumPy version should run at least an order of magnitude faster.

2.4. Conservation laws in moving domains

Reproduce the computations from **reading_conservation_laws.pdf**.

3. Assignments (9 points)

Important: all members of the group have to prepare the WHOLE lab and the presentation and answers to the questions, including the coding. The grader will decide randomly which student will answer each of the questions. All group members will receive the same grade.
Important 2: Schedule the time of the discussions directly with the TA. The discussions have to take place before the about the lecture lecture on the new topic. The maximal grade at the lab will be reduced by two points by every day which the discussion is postponed without any serious reason. In case of failure, the grade has to be repaired on the next day by redoing the discussion incorporating the feedback.

3.1. Pulsating channel flow (4 points)

Consider the flow of a fluid through a channel with height L and "infinite" length and width, as discussed in the lecture. Infinite width allows us to consider a 2D model, assuming that the flow is in complete equilibrium and does not change across the width of the channel. The 2D channel is illustrated in Fig. 1.

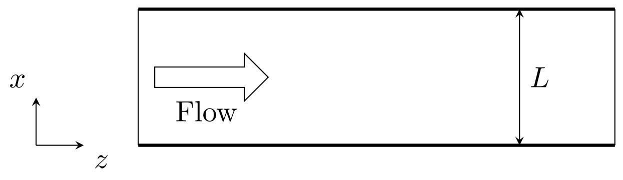

Figure 1: 2D Channel

Suppose that by means of some external mechanism, the pressure gradient of the fluid in the channel is controlled. The pressure gradient in the z-direction is constant in space and oscillates in time,

$$\frac{\partial p}{\partial z} = -a\sin(\omega t) \qquad \text{with } a > 0,$$
$$\frac{\partial p}{\partial x} = 0.$$

Under these assumptions, the Navier-Stokes equations reduce to

$$\rho\frac{\partial u}{\partial t} - \mu\frac{\partial^2 u}{\partial x^2} = a\sin(\omega t).$$

The vertical velocity component is zero and the shape of the horizontal (z) velocity component u depends only on the vertical coordinate x.

The boundary conditions for the velocity in z-direction are $u(0,t) = u(L,t) = 0$ and the initial condition is given by $u(x,0) = 0 \ \forall x \in [0,L]$.

Then, u can be computed with the following sequence:

$$u(x,t) = a\sum_{k=0}^{\infty} d_k \Psi_k(t) S_k(x) \tag{1}$$

with

$$\sigma = \frac{\mu}{\rho L^2}, \tag{2}$$
$$d_k = \frac{4}{\rho\pi(2k+1)((2k+1)^4\sigma^2\pi^4 + \omega^2)}, \tag{3}$$
$$S_k(x) = \sin\left(\pi\,(2k+1)\,\frac{x}{L}\right), \tag{4}$$
$$\Psi_k(t) = (2k+1)^2\sigma\pi^2\sin(\omega t) - \omega\cos(\omega t) + \omega\exp\left(-(2k+1)^2\sigma\pi^2 t\right). \tag{5}$$

Formulas (1)–(5) compute the full transient behavior of the flow, starting from the initial condition $u = 0$.

For the situation where the channel flow is already fully developed, i.e., for $t \gg 0$, the velocity profile has the following periodic solution:

$$u(x,t) = \frac{a}{\omega\rho}\left[\frac{f_2(x)}{f_3(x)}\sin(\omega t) - \left(1 - \frac{f_1(x)}{f_3(x)}\right)\cos(\omega t)\right] \tag{6}$$

with

$$\begin{aligned}
f_1(x) &= \mathrm{cc}\left(\kappa(x - L/2)\right)\mathrm{cc}\left(\kappa L/2\right) + \mathrm{ss}\left(\kappa(x - L/2)\right)\mathrm{ss}\left(\kappa L/2\right)\\
f_2(x) &= \mathrm{cc}\left(\kappa(x - L/2)\right)\mathrm{ss}\left(\kappa L/2\right) - \mathrm{ss}\left(\kappa(x - L/2)\right)\mathrm{cc}\left(\kappa L/2\right)\\
f_3(x) &= \mathrm{cc}^2\left(\kappa L/2\right) + \mathrm{ss}^2\left(\kappa L/2\right)
\end{aligned}$$

and

$$\begin{aligned}
\mathrm{cc}(x) &:= \cos(x)\cosh(x)\\
\mathrm{ss}(x) &:= \sin(x)\sinh(x)\\
\kappa &= \sqrt{\frac{\omega\rho}{2\mu}}.
\end{aligned}$$

In the context of biomedical flow, this type of flow (i.e., pulsating channel or pipe flow) is referred to as Womersley flow. It can be characterized by the dimensionless Womersley number,

$$\alpha = L\sqrt{\frac{\omega\rho}{\mu}}$$

The constants are given in CGS units as $a = 1\,\mathrm{g/(cm\,s^2)}$, $\mu = 0.035\,\mathrm{g/(cm\,s)}$, $\rho = 1\,\mathrm{g/cm^3}$, $L = 1\,\mathrm{cm}$, $\omega = 1\,\mathrm{s^{-1}}$ for an example of arterial blood flow.

1. **(3 points)** Write functions that compute the velocity profile $u(x,t)$ at any time t (for all $x \in [0, L]$ at once). Implement
   - the transient case via Eq. (1) and
   - the periodic case via Eq. (6).

   Make an animated plot (see section A for an example) to visualize the oscillating velocity profiles. Show both solutions in the same figure.

2. **(0.5 points)** Above, we stated that the periodic velocity solution (6) is valid for $t \gg 0$. Comparing the results with those of the transient formula (1), after how much time is the periodic solution a good approximation?

3. **(0.5 points)** The constants above were defined for a pulsating blood flow in an artery. Instead of blood, consider a fluid with a higher dynamic viscosity of $\mu = 0.1\,\mathrm{g/(cm\,s)}$. Compare the times the transient solution needs to reach equilibrium (the periodic solution). By looking at the transient formula, what is the relationship between the equilibrium time and the viscosity?

3.2. Conservation of angular momentum (5 points)

We saw in the lectures and in the complementary reading that the general conservation law reads:

$$\partial_t L + L\sum_{i=1}^{3}\partial_{x_i}\mathbf{u}_i + \sum_{k=1}^{3}(\mathbf{u}_k - \mathbf{w}_k)\partial_{x_k}L - \sum_{\ell=1}^{3}\partial_{x_\ell}\mathbf{b}_\ell = 0 \tag{7}$$

The goal is to proof that the Cauchy stresses need to be symmetric, i.e. $\sigma = \sigma^\intercal \in \mathbb{R}^{3\times 3}$ as stated in the lecture, by assuming that the fluid flow satisfies the conservation of angular momentum. For this we will proceed in some steps:

1. **(1 point)** As you have seen, on a control volume Ω, the vector forces acting on the boundary are $\sigma\mathbf{n}$, and hence the angular momentum vector produced by those forces is $\mathbf{r}\times\sigma\mathbf{n}$, with $\mathbf{r}_i = x_i(X,t)$ and $\mathbf{w} = \partial_t\mathbf{r}$.

   In order to use the differential relation (7), we need to convert the external boundary source into a volume source . Therefore, for the j-th component of the angular momentum vector, show that:

$$\int_{\partial\Omega}[\mathbf{r}\times\sigma\mathbf{n}]_j = \int_{\Omega}\sum_{\ell=1}^{3}\partial_{x_\ell}([\mathbf{r}\times\sigma_{:,\ell}]_j) \tag{8}$$

   with $\sigma_{:,i}$ the ith-column of σ (recall the notations from the lecture and complementary material on conservation laws in moving domains).

2. **(2 points)** Asssume that Equation (7) holds for linear momentum (i,.e for $L = \rho\mathbf{u}_1, \rho\mathbf{u}_2$ and $\rho\mathbf{u}_3$ with $\mathbf{b}_\ell = \sigma_{\ell,1}, \sigma_{\ell,2}, \sigma_{\ell,3}$, respectively). Then, prove that if $L = [\mathbf{r}\times\rho\mathbf{u}]_j$, $\mathbf{b}_i = [\mathbf{r}\times\sigma_{:,i}]_j$, Equation (7) for $j = 1,2,3$ leads to:

$$(\partial_t\mathbf{r})\times\rho\mathbf{u} + \left\{\sum_{k=1}^{3}(\mathbf{u}_k - \mathbf{w}_k)(\partial_{x_k}\mathbf{r})\right\}\times\rho\mathbf{u} = \sum_{i=1}^{3}(\partial_{x_i}\mathbf{r})\times\sigma_{:,i} \tag{9}$$

3. **(1 point)** Show now that the left-hand-side of Equation (9) is zero.

4. **(1 point)** Show now that the Cauchy stress matrix is symmetric.

   You have to deliver all the steps written on paper for grading.

A. Example: animation with matplotlib

The easiest way to create an animated plot with matplotlib is as follows:

```python
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 1, 100)
times = np.linspace(0, np.pi, 20)

plt.ion()
plt.figure()
p1, = plt.plot([], [])
plt.xlim(0, 1)
plt.ylim(-1, 1)
for t in times:
    y = x**2 * np.cos(t)
    p1.set_data(x, y)
    plt.draw()
    plt.pause(0.1)
```

`p1.set_data(x, y)` updates the existing line plot. This is much faster than deleting the plots with `plt.clf()` and calling `plt.plot` repeatedly inside the `for`-loop.

The figure can be closed with the `plt.close()` command.

B. Example: animation within Jupyter-Notebooks

In order to make an animation, we rely on the Ipython module HTML, as the example below:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation


t = np.linspace(0,2*np.pi)
x = np.sin(t)

fig, ax = plt.subplots()
ax.axis([0,2*np.pi,-1,1])
l, = ax.plot([],[])

def animate(i):
    l.set_data(t[:i], x[:i])

ani = matplotlib.animation.FuncAnimation(fig, animate, frames=len(t))

from IPython.display import HTML
HTML(ani.to_jshtml())
```