Physics 77/88 - Fall 2024 - Homework 3

Visualization (and more about functions)

Submit this notebook to bCourses to receive a credit for this assignment.
due: **Oct 9 2024**
Please upload both, the .ipynb file and the corresponding .pdf

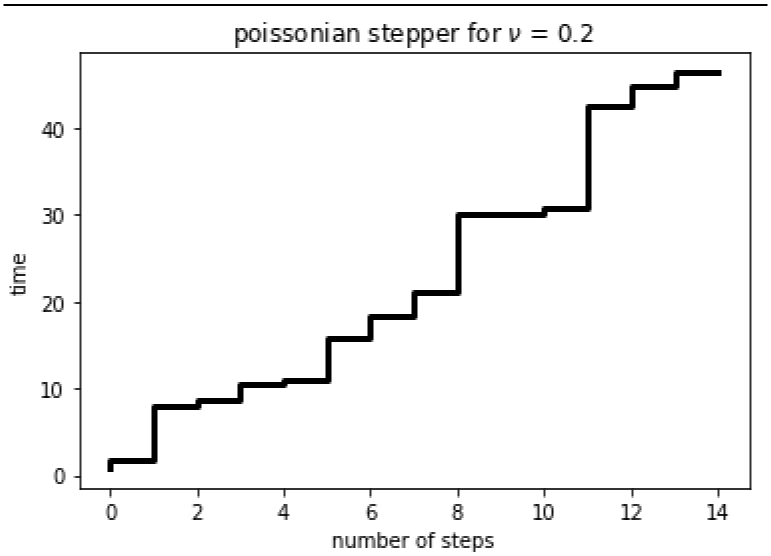
Problem 1 (5P)

The Poissonian Stepper is a common model that helps to understand diffusion processes. In its simplest 1D, oneway version, the stepper takes a step after the time τ, where

$$\tau = -\frac{1}{\nu}\, ln(r)$$

with ν being the hopping rate and r being a uniformly distributed random number (0, 1). Write the function **PoissStep.py** using *def*, that:

- takes the number of steps and the hopping rate as input arguments.
- takes the hopping rate with the default value $\nu = 1$
- runs the Poissonian Stepper and generates and saves the following plot as .pdf:



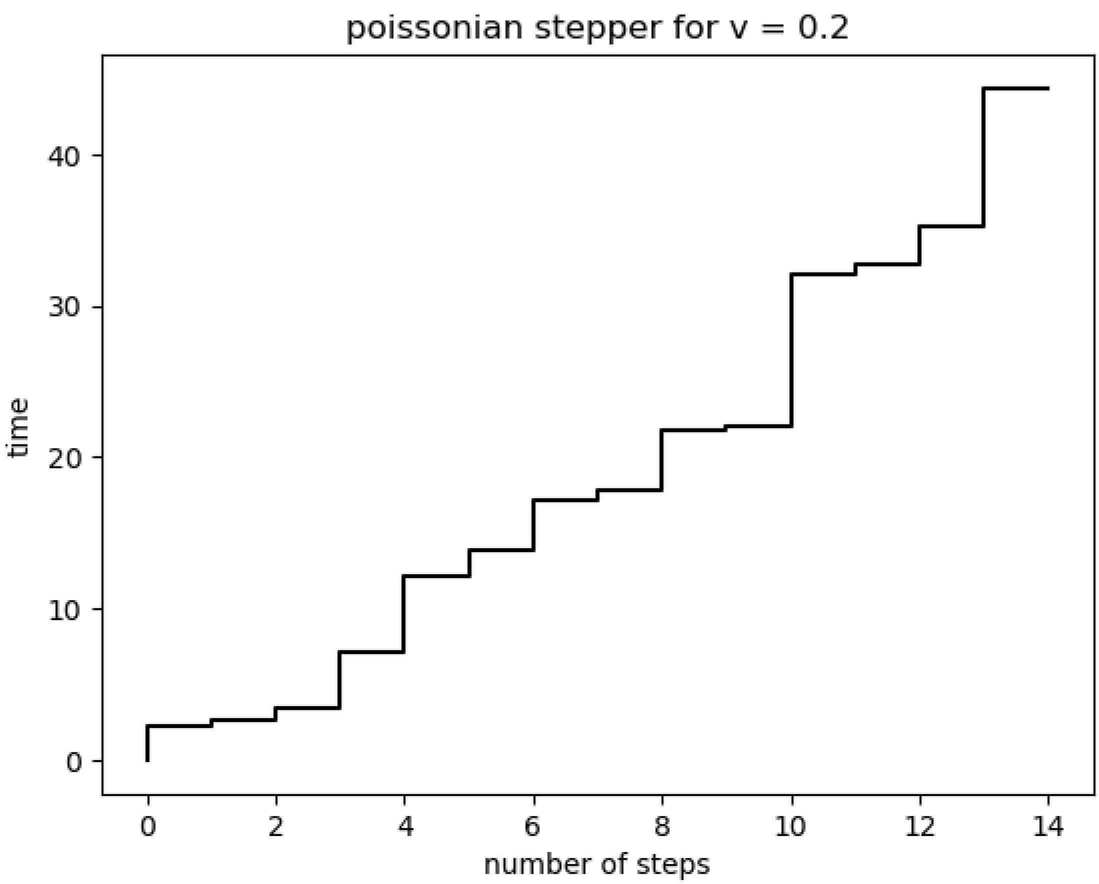
In [2]:
```python
import numpy as np
import matplotlib.pyplot as plt
```

```
def PoissStep(n, v=1):
    r = np.random.uniform(0, 1, n+1)
    v0 = np.repeat(v, n+1)
    n0 = np.arange(0, n+1)
    t = -(1/v0)*np.log(r)

    steptimes = np.array([])
    for i in range(0, n+1):
        k = 0
        for j in range(0, i):
            k = k + t[j]
        steptimes = np.append(steptimes, k)

    plt.step(n0, steptimes, '-', color = 'black', label = 'step')
    plt.xlabel('number of steps')
    plt.ylabel('time')
    plt.title(f'poissonian stepper for v = {v}')
    plt.savefig('poissonianstepper.pdf')
    plt.show()
```

In [26]:
```
PoissStep(14, 0.2)
```



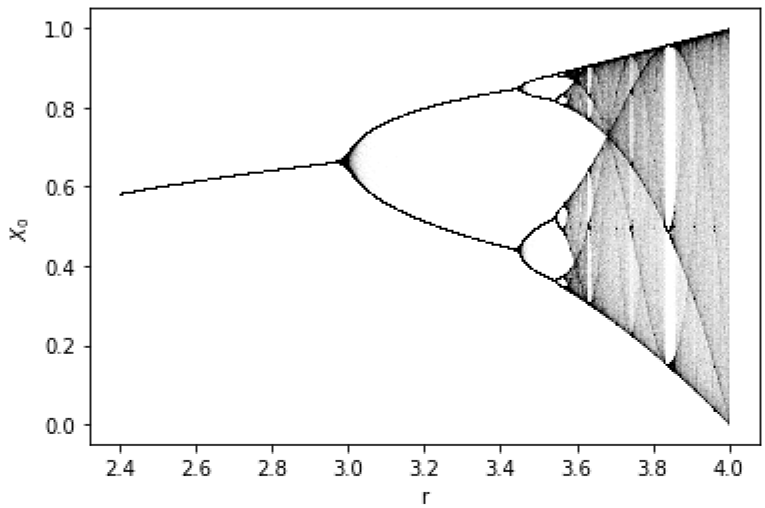
Problem 2 (10P)

When we will talk about ODEs, we will investigate the behaviour of non-linear systems and the stability of their solutions. After many iterations ($t \approx 10^2$), the equation

$$x_{t+1} = r\, x_t\, (1 - x_t)$$

reaches a constant value (aka *fixed point*).

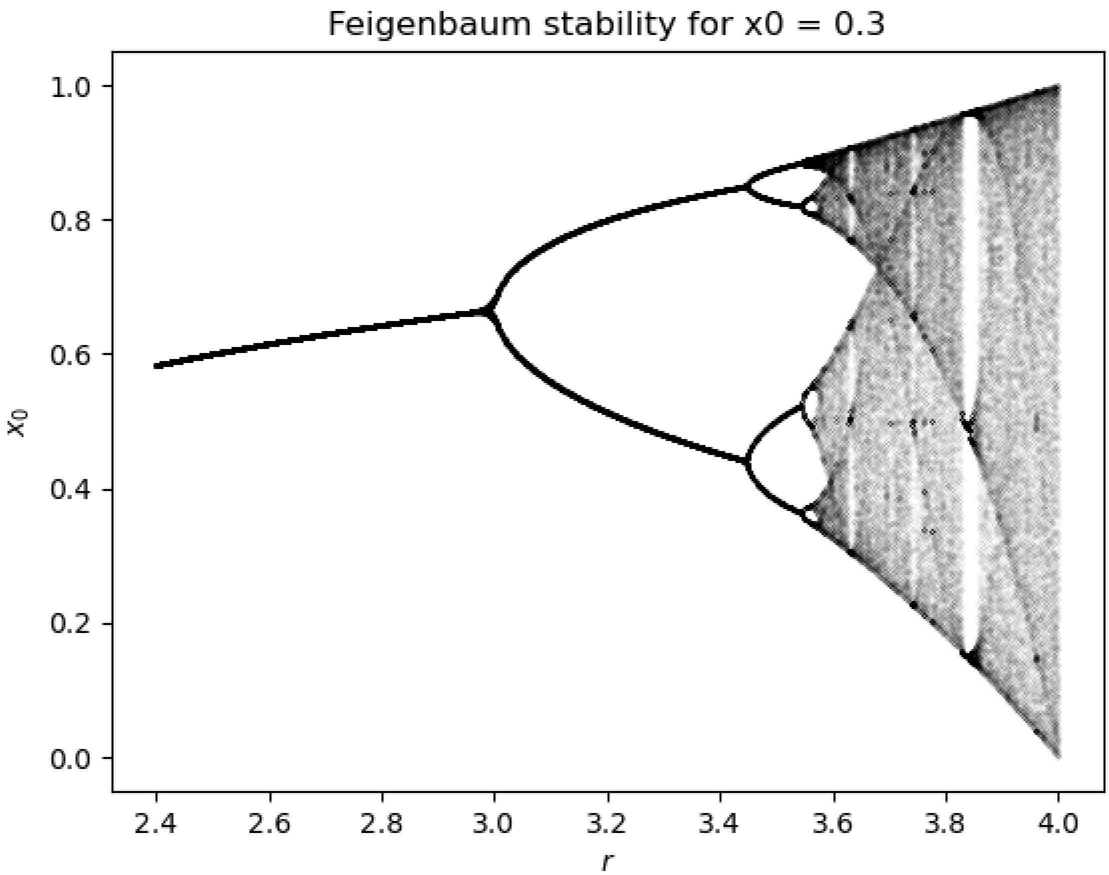
Write the function **Feigenbaum.py** using *def*, that runs the above equation for different x_0 and different r and that generates the following plot.



In [32]:

```
import numpy as np
import matplotlib.pyplot as plt
def Feigenbaum(rmin, rmax, M, x0, N, t):
    def logistic(rmin, rmax, M, xo, N, t):
        R = np.linspace(rmin, rmax, M)
        X = np.empty((M,N-t))
        xprev = np.full(M,x0)
        for i in range (1, t):
            xprev = xprev * R * (1-xprev)
        X[:,0] = xprev
        for i in range(1,N-t):
            X[:,i] = X[:,i-1] * R * (1-X[:,i-1])
        return X,R
    X, R = logistic(rmin, rmax, M, x0, N, t)
    R = np.repeat(R,N-t)
    plt.scatter(R, X, c = 'black', s = 0.001)
    plt.title('Feigenbaum stability for x0 = ' f'{x0}')
    plt.xlabel('$r$')
    plt.ylabel('$x_0$')
    plt.show()

Feigenbaum(2.4, 4, 1000, 0.3, 300, 100)
```

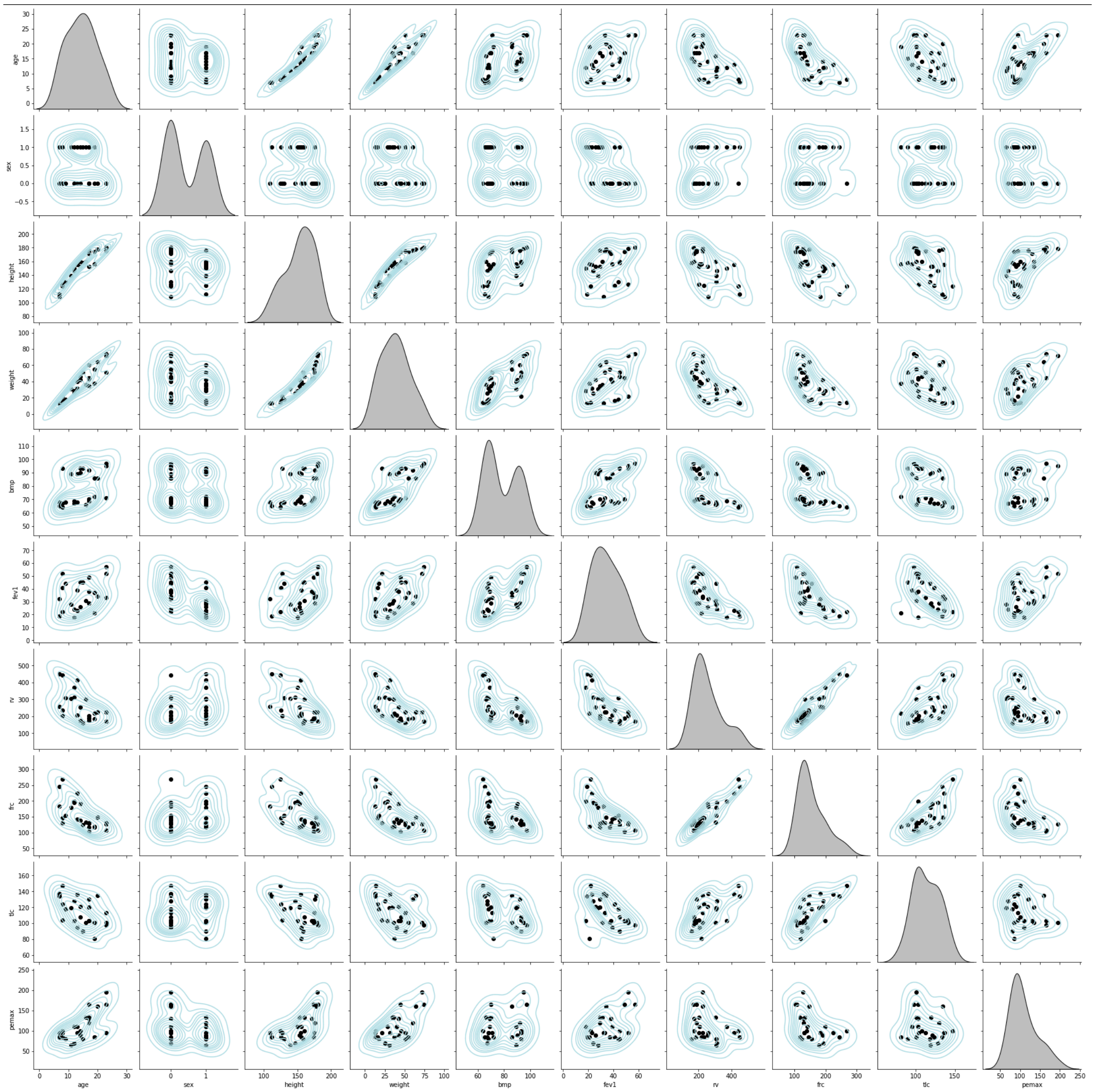


Problem 3 (7P)

We read the following dataset using **pandas** by running

In [ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Write a function using *def*, that creates an sns pairplot in kde mode like in the following figure:

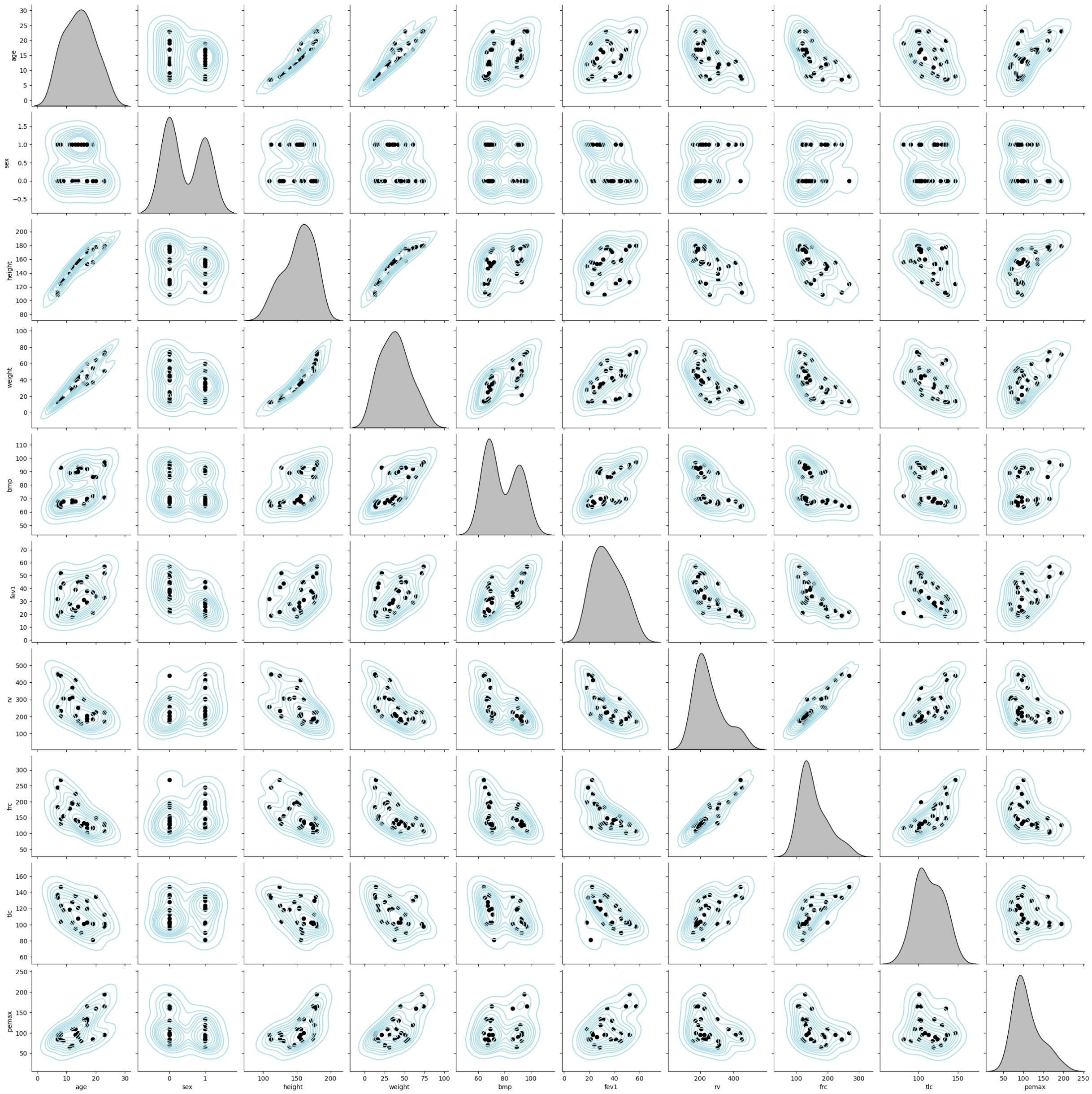


In [33]:

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

def plot(data):
    data = pd.read_csv(data, sep = r'\s+')
    out = sns.pairplot(data, kind = "kde", \
                             plot_kws = {'color':[176/255,224/255,230/255]}, \
                             diag_kws = {'color':'black'})
    out.map_offdiag(plt.scatter, color = 'black')
    plt.show()

plot('cystfibr.txt')
```



Problem 4 (optional 3P)

Create a **class**, *"MyClass"*, that takes an item when initialized. The class should contain an attribute that allows the item to be multiplied by a second item (input is type *int*), i. e. construct an operator overflow like e. g. for type *list*. Furthermore, the class should have an attribute allowing to return the length of the item.

Problem 5 (optional 3P)

The decorator *"My_Timer"* is a useful function that measures runtime of Python scripts

In [24]:

```
def My_Timer(my_function):
    def get_args(*args,**kwargs):
        t1 = time.monotonic()
        results = my_function(*args,**kwargs)
        t2 = time.monotonic()
        dt = t2 - t1
        print("Total runtime: " + str(dt) + ' seconds')
        return results
    return get_args
```

Write a **class**, *"My_Timer"*, that serves the same purpose and has the same functionality.