MoMeDa_3

June 6, 2024

1 Moderne Methoden der Datenanalyse SS2024

2 Practical Exercise 3

2.1 Exercise 3: Maximum Likelihood and χ^2 Methods

Fitting parametrized functions to measured data is daily business in research. By this, models can be tested against experiments. Moreover, parameters of the models and their uncertainties can be determined. The physicist often refers to this process as "*fitting*" — in general it is called "*parameter estimation*".

2.2 Exercise 3.1: Decay (obligatory)

Generate uniformly distributed random numbers. Then apply the transformation method to generate random numbers following an exponential distribution $\exp(-x/\tau)$ for $x > 0$. These values can be interpreted as measurements of decay times t (e.g., of radioactive particles) corresponding to a lifetime τ, which have the following distribution:

$$f(t, \tau) = \frac{1}{\tau} \cdot \exp\left(-\frac{t}{\tau}\right)$$

a) Show analytically that the maximum likelihood estimator for τ is the mean $\hat{\tau}$ of the sample ($\hat{\tau}$ = mean of all measured decay times t_i).

$$\prod_{i=1}^{N} f(t_i, \tau) = \prod_{i=1}^{N} \frac{1}{\tau} \exp(-\frac{t_i}{\tau}) = \frac{1}{\tau^N} \exp(-\frac{\sum_{i=1}^{N} t_i}{\tau}) \tag{1}$$

$$\Rightarrow 0 = \partial_\tau (\frac{1}{\tau^N} \exp(-\frac{\sum_{i=1}^{N} t_i}{\tau})) \tag{2}$$

$$= (-N \frac{1}{\tau^{N+1}} + \frac{1}{\tau^{N+2}} \sum_{i=1}^{N} t_i) \exp(-\frac{\sum_{i=1}^{N} t_i}{\tau})) \tag{3}$$

$$\Rightarrow N \frac{1}{\tau^{N+1}} = \frac{1}{\tau^{N+2}} \sum_{i=1}^{N} t_i \tag{4}$$

$$\Rightarrow \tau = \frac{1}{N} \sum_{i=1}^{N} t_i \tag{5}$$

$$\tag{6}$$

b) Generate 1000 samples with τ=1, each with $N = 10$ values of t. Evaluate the mean $\hat{\tau}$ for each sample and create a histogram of the resulting means. Compare the mean of $\hat{\tau}$ with the true value τ=1.

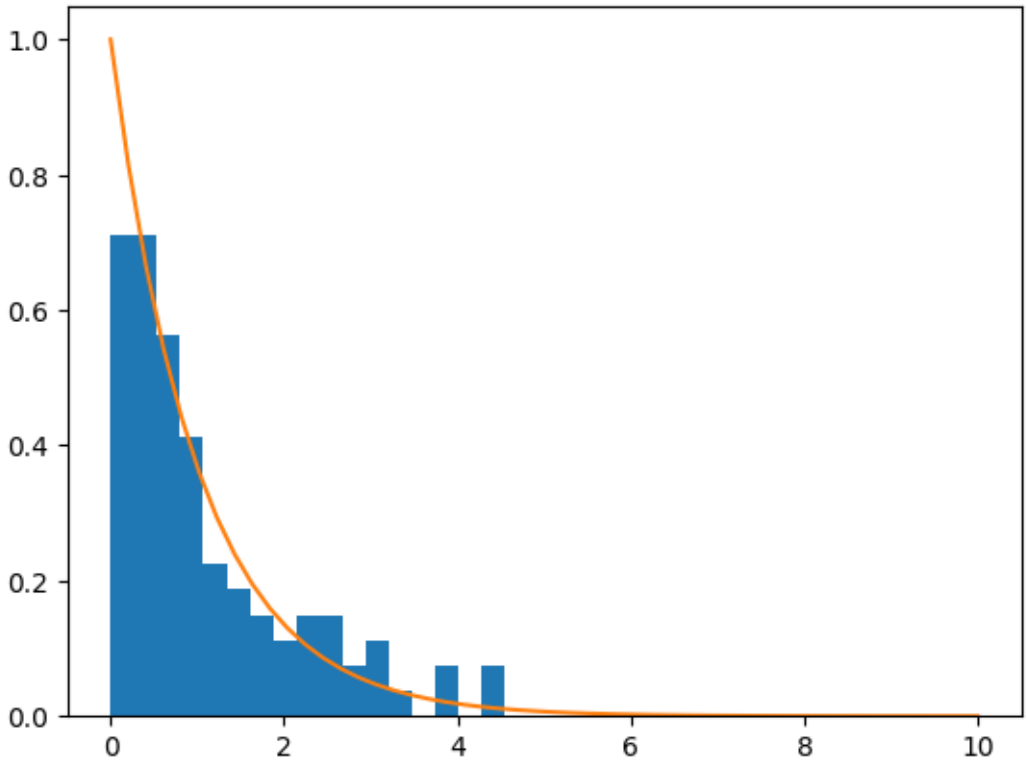
[2]:
```
# Similar to the previous exercises, you can use ROOT or the pythonic approach...

# Pure python:
import numpy as np
import matplotlib.pyplot as plt

from scipy import optimize, stats

# ROOT:
#from ROOT import gRandom, TCanvas, TH1F, TF1
```

Define the function to generate the random numbers first. Use the methods introduced on the previous exercise sheets to get uniformly distributed numbers and then transfrom them as required.

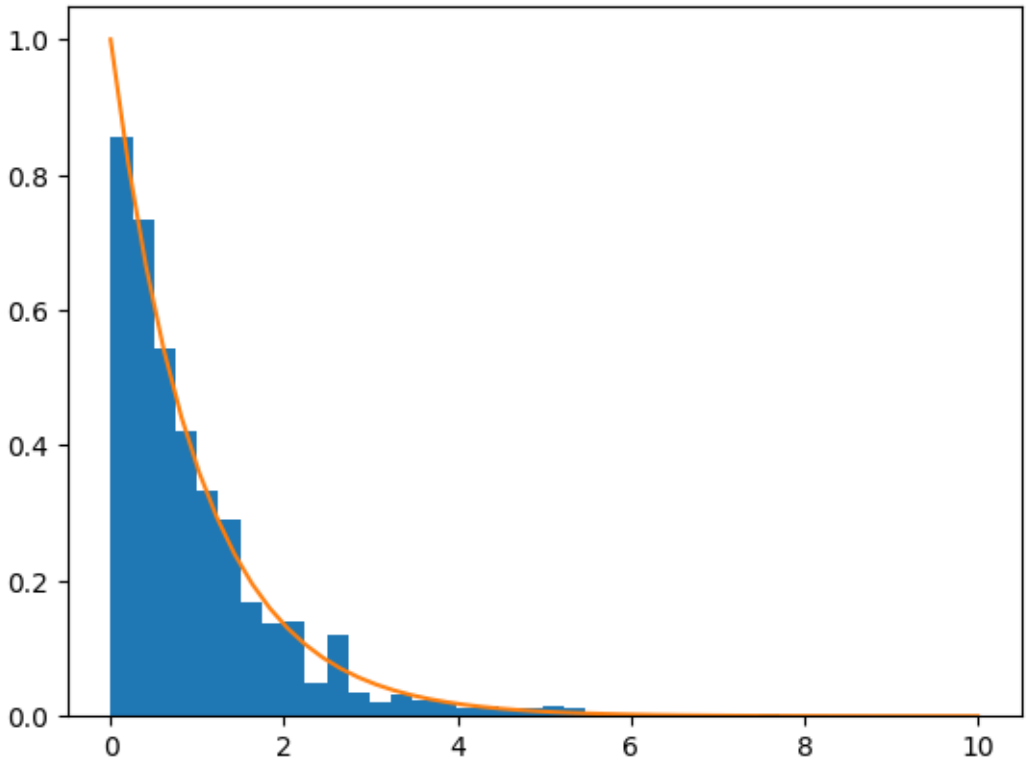
[3]:
```
def generate_data(N: int, tau: float = 1.0) -> np.ndarray:
    # Generate random numbers according to exp(-x/tau) for x>0 using the␣
 ↪transformation method

    # FYI: The type hints in the function signature above tell you that
    #  - the function expects an integer value for the argument `N`
    #  - has a second parameter `tau` for the lifetime, which has a default␣
 ↪value of 1.0 as required by the exercise
    #  - and will return a numpy array.

    rg = np.random.random(N)
    r = - np.log(rg*tau)

    return r
```

[4]:
```
tau = 1
for N in [1e2,1e3,1e4,1e5]:
    print(int((np.log(N)/np.log(2))**1.5))
    plt.hist(generate_data(int(N),tau), density=True, bins=int((np.log(N)/np.
 ↪log(2))**1.5))
    x = np.linspace(0,10)
    plt.plot(x, np.exp(-x/tau)/tau)
    plt.show()
```

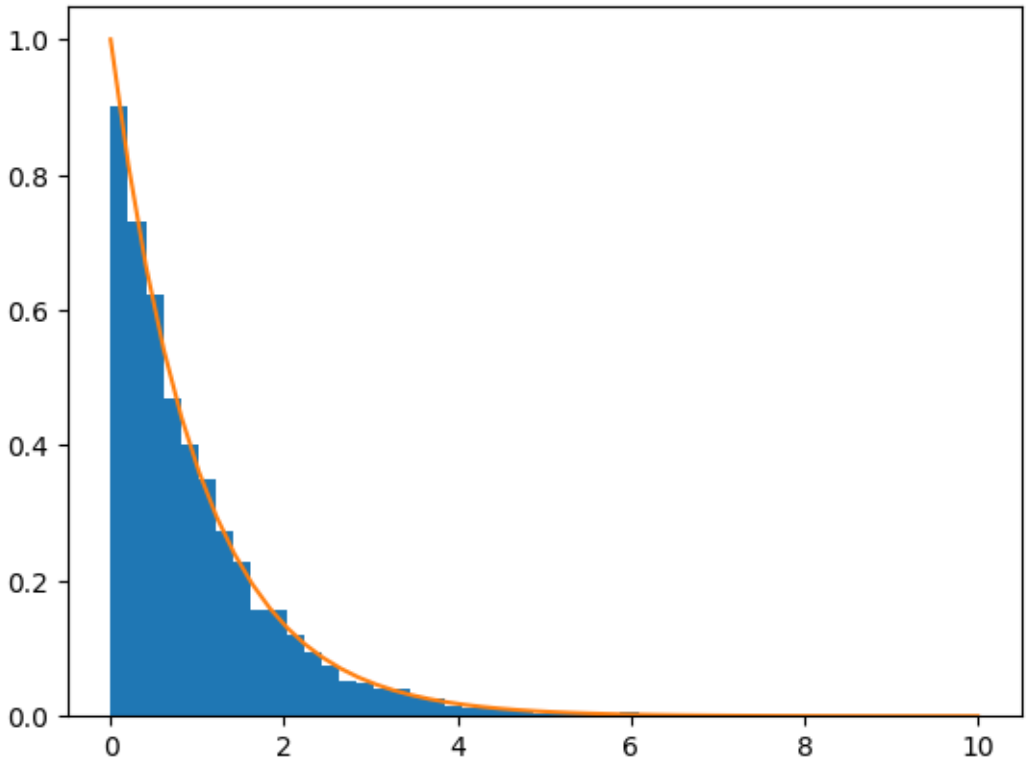
```
17
```



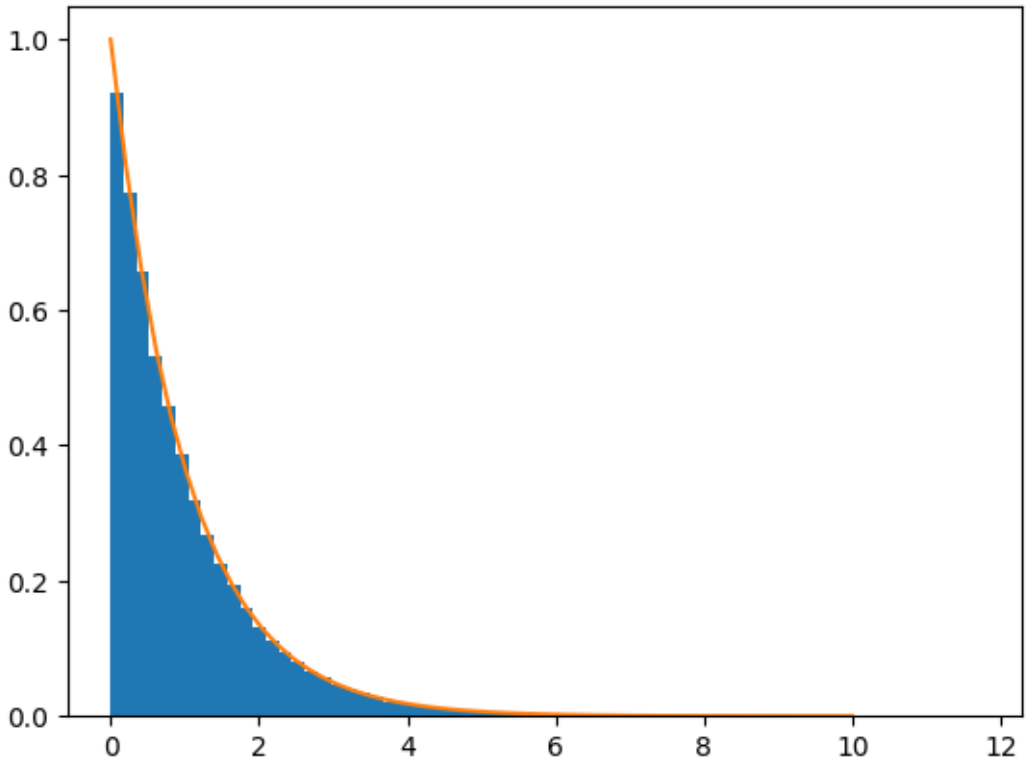
31



48



67



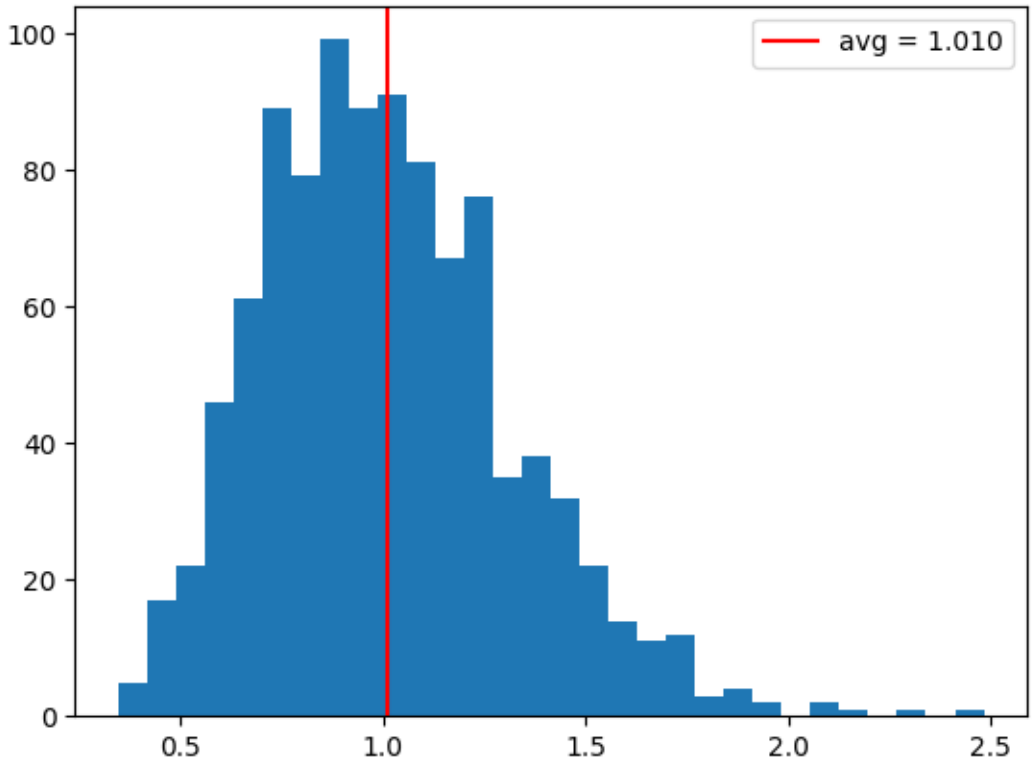
2.2.1 Hints for pure Python approach:

You can use `matplotlib`'s `matplotlib.pyplot.hist` (= plt.hist) function to plot a histogram of given data. You can use for instance 100 bins.

2.2.2 Hints for ROOT approach:

Use ROOT's builtin TH1F histogram class, as introduced in exercise 1.

[5]:
```
# TODO: Add code here to generate the 1000 data samples, create a histogram of␣
 ↪the mean values of the data and draw it

def create_histo_and_calculate_bias(N):
    tauhat = np.array([np.average(generate_data(N,tau)) for _ in range(1000)])
    plt.hist(tauhat, bins=30)
    plt.axvline(x=np.average(tauhat),color="red", label=f"avg = {np.
 ↪average(tauhat):.3f}")
    plt.legend()
    plt.show()

create_histo_and_calculate_bias(10)
```



c) Assume that the probability density function (p.d.f.) has been parametrized in terms of $\lambda = 1/\tau$, which means:

$$f(t, \lambda) = \lambda \cdot \exp(-\lambda \cdot t)$$

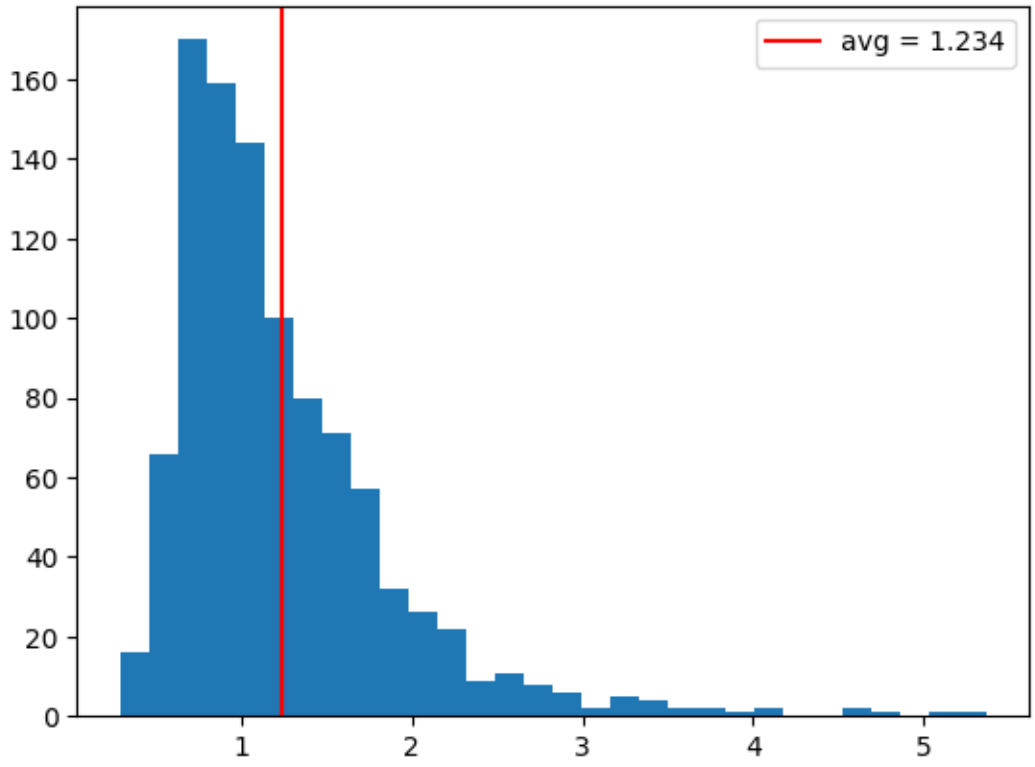
Create a histogram of the estimations $\hat{\lambda}$. Compare the mean value of $\hat{\lambda}$ with the true value λ=1, and determine numerically the bias for $N = 5, 10, 100$.

Calculate the bias also for the experiments made in the exercise part **b)** and compare the results of the two approaches **b)** and **c)**.

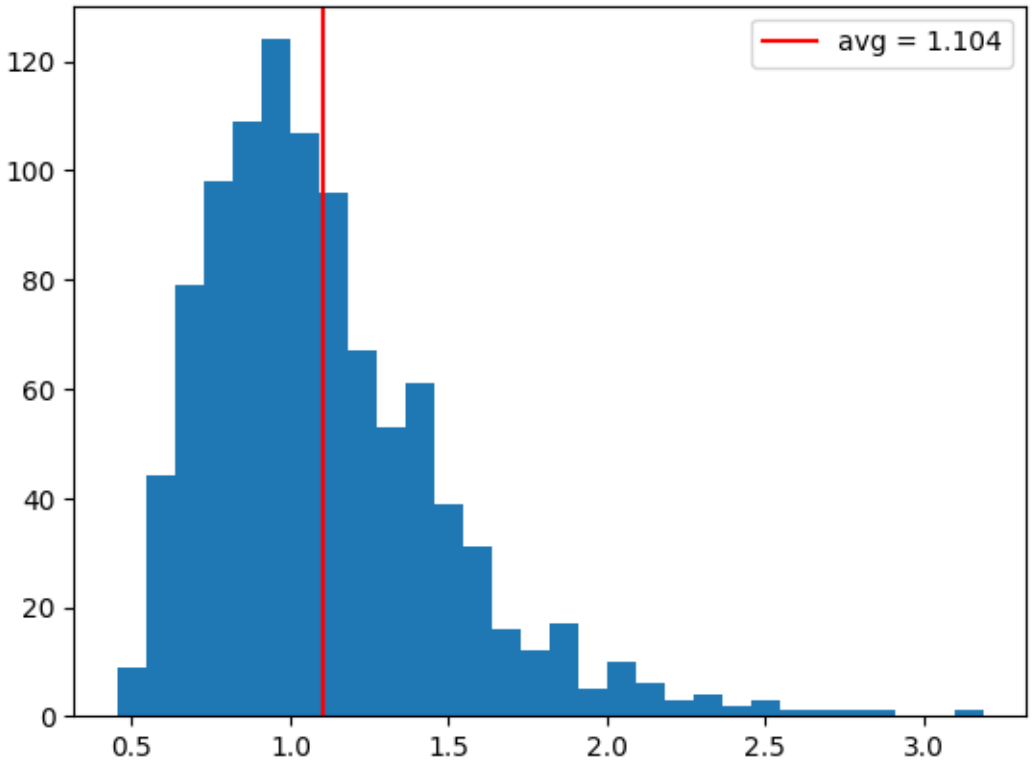
Use a similar approach as above to obtain the histograms using the alternative function definition.

[6]:
```
# TODO: Add code here to generate the 1000 data samples, create a histogram of␣
 ↪the mean values of the data and draw it

def create_lambda_histo_and_calculate_bias(N):
    lamdahat = [1/np.mean(generate_data(N)) for _ in range(1000)]
    plt.hist(lamdahat, bins=30)
    plt.axvline(x=np.average(lamdahat),color="red", label=f"avg = {np.
 ↪average(lamdahat):.3f}")
    plt.legend()
    plt.show()
```

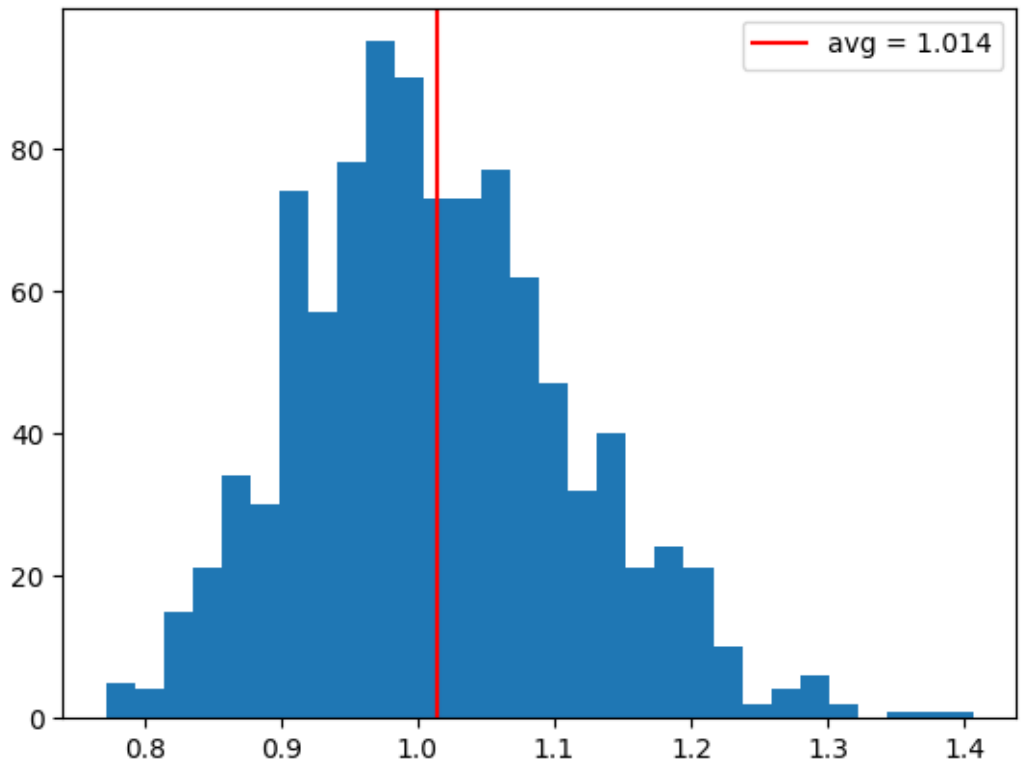
```
    print(f"Bias = {np.average(lamdahat)-1}")

for N in [5,10,100]:
    create_lambda_histo_and_calculate_bias(N)
```




```
Bias = 0.23440257749638338
```




```
Bias = 0.10447815535638649
```




```
Bias = 0.014135368346126942
```

d) Compare the results of the maximum likelihood method and the χ^2 method: Make three different histograms with 1000 bins from 0 to 10 containing N generated decay times t (try $N = 10, 1000, 100000$). Fit the function $f(t, \tau)$ to each histogram using the χ^2 method and the binned likelihood method. Compare the fitted parameters and the χ^2 values of both methods and discuss the results.

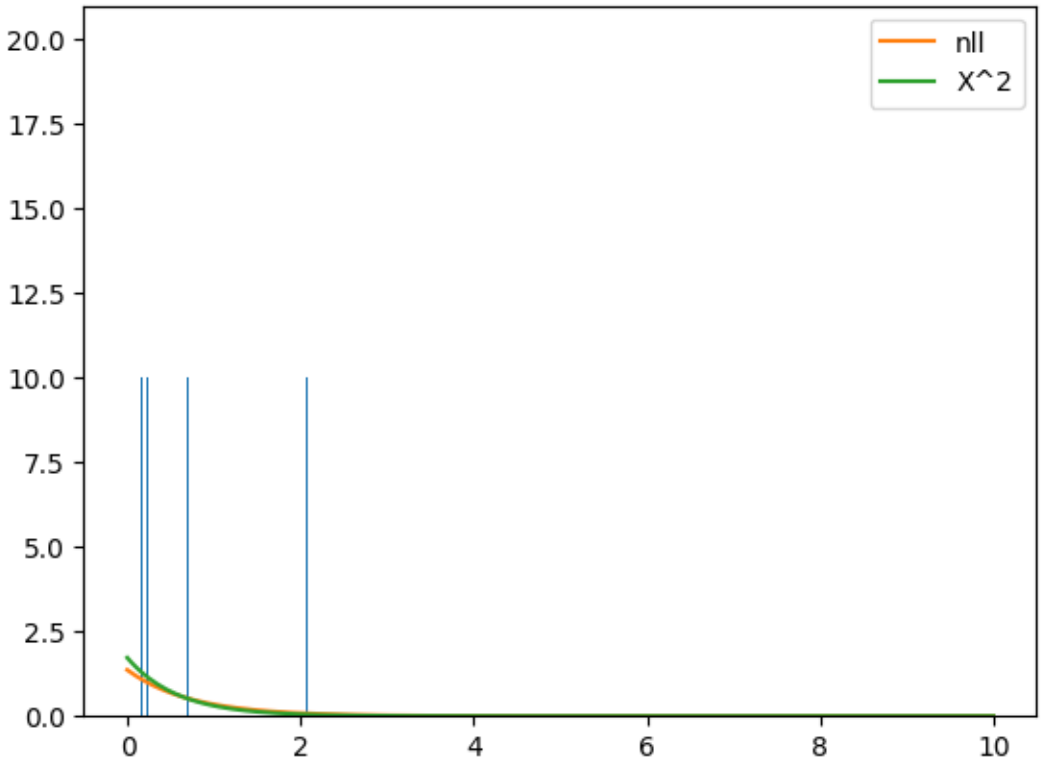
The following function template contains some hints for the ROOT approach. If you are using the pure python approach, you can get replace function body with respectively. Use `matplotlib.pyplot.hist` (= plt.hist) and `scipy.optimize` as in the previous exercises instead.

[7]:
```
#from ROOT import kRed, kGreen  # Use this if you want get some color into your␣
 ↪ROOT plots..

from scipy.optimize import minimize
from scipy.optimize import curve_fit

def make_histogram_and_fit(N):
    t = generate_data(N)
    n, binedges = np.histogram(t, bins=1000, range=(0,10), density=True)

    def v(lower, upper, tau):
```

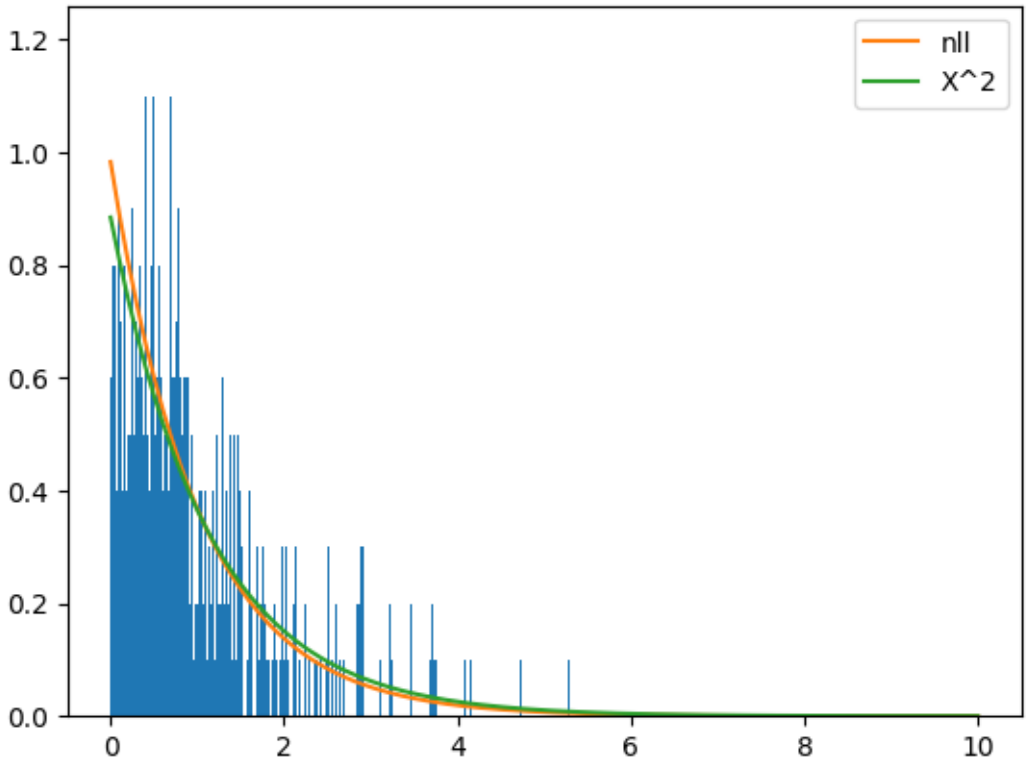
```
        return -(np.exp(-upper/tau)-np.exp(-lower/tau))

    def loglikelihood(tau):
        return -np.sum(n*np.log(v(binedges[:-1], binedges[1:], tau)))

    minimized = minimize(loglikelihood, (3,), method="Nelder-Mead")
    taunll = minimized.x[0]
    print(f"nll: {taunll}")

    def density(t, tau):
        return 1/tau*np.exp(-t/tau)

    def x2(tau):
        return np.sum((density((binedges[:-1]+binedges[1:])/2, tau) - n)**2)

    #taux2 = curve_fit(density, (binedges[:-1]+binedges[1:])/2, n)[0][0]
    minimizedx2 = minimize(x2, (3,), method="Nelder-Mead")
    taux2 = minimizedx2.x[0]
    print(f"x^2: {taux2}")


    tlin = np.linspace(0,10,200)
    plt.hist(t,bins=1000, range=(0,10), density=True)
    plt.plot(tlin, 1/taunll*np.exp(-tlin/taunll), label="nll")
    plt.plot(tlin, 1/taux2*np.exp(-tlin/taux2), label="X^2")
    plt.legend()
    plt.show()

for N in [10,1000,100000]:
    make_histogram_and_fit(N)
```

```
nll: 0.7319824218749946
x^2: 0.5790527343749943
```

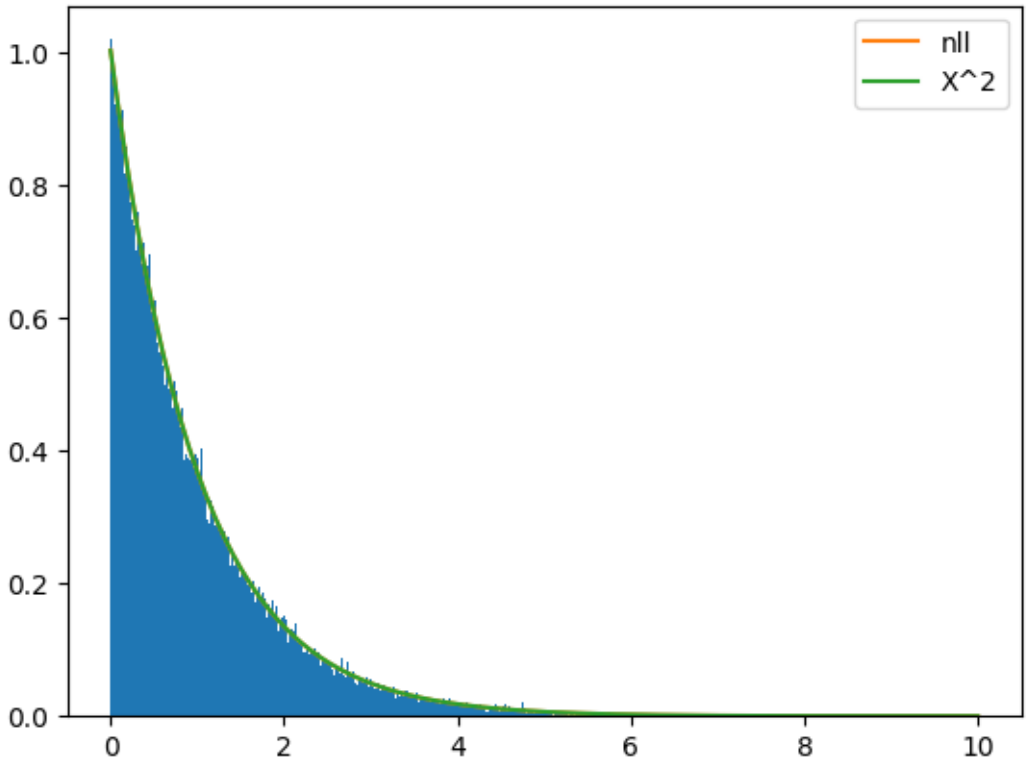
```
/tmp/ipykernel_8136/2111959965.py:14: RuntimeWarning: invalid value encountered
in log
  return -np.sum(n*np.log(v(binedges[:-1], binedges[1:], tau)))
```




```
nll: 1.0174804687499952
x^2: 1.1307128906249955
```




```
nll: 0.9972656249999954
x^2: 0.9981445312499952
```



3 Exercise 3.2: MINUIT (voluntary)

The goal of this exercise is to make you familiar with the minimizer package `MINUIT` which was developed at CERN in the 70s in `FORTRAN`. This well-tested toolbox provides different minimization algorithms, the most famous one being `MIGRAD`. The package is particularly liked by physicists due to it's sophisticated methods for the parameter uncertainty estimation.

For the purpose of this exercise, it is suggested to use the Python frontend to `MINUIT`, which is available in the form of the package `iminuit`.

Take the function $f(t, \tau)$ and the generated data set from the previous Exercise 3.1, and perform an unbinned log likelihood fit for $N = 10, 1000, 100000$ entries.

Plot a histogram from 0 to 10 with the N entries and the fitted function normalized to the number of entries. Display the value of the negative logarithmic likelihood as a function of the fit parameter τ from 0.5 to 5. How is this plot related to the uncertainty of the fitted parameter?

The `iminuit` Python package provides predefined cost function classes which also cover the unbinned case we are interested in. However, to learn how to define your own cost function, you should use the `scipy`-like interface which allows you to provide your own cost function in form of the argument `fun`. You can use either of the two approaches and take a look at the output that MINUIT provides after the cost function is minimized.

Please note, that the minimizers provided in `scipy.optimize` are also capable of performing the

minimization task. However, the MINUIT package has proven itself in particle physics for decades and provides certain functionality which is not built into the scipy optimizers by default (e.g. the handling of uncertainties).

Hint: Make sure, that you are using a current version of iminuit, e.g. 2.11.2! Check this with the command in the next cell:

[11]:
```
!pip list | grep iminuit
```

```
iminuit                   2.25.2
```

[12]:
```
# For a predefined cost function use
from iminuit import cost, Minuit

# or use the following, if you want to define your own cost function
from iminuit.minimize import minimize as iminuit_minimize
```

[51]:
```
# Definition of the fit function

def fit_func(x, tau):
    # TODO
    return ...
```

[52]:
```
# TODO: Generate the data as in the previous exercises, plot the histograms and␣
 ↪fit the fit function to the data.
# Draw also the function with the estimated tau value obtained with the fit.
# Do this for N = 10, 1000, 100000
```

Hint: Check out the tutorials / exampes available in the iminuit or scipy.optimize documentation, depending on which method you choose.