MoMeDa_7

July 3, 2024

1 Exercise 7: Confidence Intervals

In the exercises of this sheet we consider a measured sample of events after applying quality cuts to separate signal from background events. The total number of observed events follows a Poisson distribution:

$$P(n_0|\nu_t) = \frac{\nu_t^{n_0} e^{-\nu_t}}{n_0!}, \tag{1}$$

n_0 is the number of observed events after the quality cuts, and $\nu_t = \nu_{t,S} + \nu_{t,B}$ is the true number of events expected to pass the cuts, where $\nu_{t,S}$ is the contribution from true signal events and $\nu_{t,B}$ the background remaining after the cuts.

[2]:
```
%matplotlib inline
import matplotlib.pyplot as plt

import numpy as np
from scipy.stats import poisson, norm
from scipy.interpolate import interp1d as scipy_interp1d
```

1.0.1 Exercise 7.1: Significance (voluntary)

For a specific experiment, the background is expected to be $\nu_{t,B} = 1$ while the measurement is $n_0 = 3$.

a) How often do you expect to measure n_0 or more events if you expect only background?

b) What is the corresponding significance?

c) Is the measurement compatible with a statistical fluctuation of the background, or is there an evidence (defined as significance $\geq 3\,\sigma$)or a discovery of the signal ($\geq 5\,\sigma$)?

d) How many events would have to be observed to reach the evidence and discovery thresholds?

[ ]:
```
# Define a function to calculate the probability to observe n_0 or more events,␣
↪if the expected number is mu
# Hint: Have a look at the methods of scipy.stats.poisson and in particular the␣
↪method poisson.cdf.
```

```
def pPoisson(n_0, mu):
    # TODO: Calculate the probability to observe n_0 or more events
    #       and also determine the necessary number of observed events for a␣
 ↪discovery / evidence.
    # Hint: Take a look at the method scipy.stats.norm.ppf to calculate the␣
 ↪significances in terms of standard deviations.
    pass
```

[ ]:
```
def exercise7_1():
    # TODO: Implement your solution to this exercise
    pass
```

[ ]:
```

```

1.0.2 Exercise 7.2: Confidence intervals (obligatory)

1.0.3 a) Frequentist approach (Neyman construction)

This classical approach can be used to determine a confidence interval or lower/upper limit, respectively, at a given confidence level (CL).

First, let us assume that the background is negligible: $\nu_{t,B} = 0$. Using the frequentist approach, we will compute a 90% CL upper limit on $\nu_t = \nu_{t,S}$ if the measured number of event is $n_0 = 3$.

1) Determine the 90 % CL upper limit ν_{UL} such that:

$$\sum_{n=0}^{n_0} P(n|\nu_{UL}) \quad = \quad 0.10. \tag{2}$$

2. Now, consider the realistic case with background, i.e., $\nu_{t,B} > 0$:

   - Compute the 90% CL upper limits on $\nu_{t,S}$ as function of $\nu_{t,B}$. By convention, the upper limit on $\nu_{t,S}$ is the limit which would be obtained for $\nu_{t,S}$ without background minus the number of expected background events, i.e., $\nu_{t,B}$, (CL_{SB}-limit). This is done automatically when calling the function `get_upper_poisson_limit` (defined below) with $\nu_{t,B} > 0$. CL_{SB} is the confidence level with respect to the signal-plus-background hypothesis and a measure for the compatibility of the experiment with this hypothesis. **What makes this procedure inconvenient?**

   - Make a plot of the CL_{SB}-limit on $\nu_{t,S}$ as a function of $\nu_{t,B}$ varying n_0 from 0 to 5 (draw all six curves in the same plot).

   - Utilizing the function `get_upper_poisson_limit_normalized` (also defined below) the limit is calculated using the CL_S-method: $CL_S = CL_{SB}/CL_B$. CL_B is a measure for the compatibility with the background-only hypothesis. The CL_S-method provides a useful limit, even if the number of observed events is much smaller than the expected background.

   - Create also a plot of the CL_S-limit on $\nu_{t,S}$ as a function of $\nu_{t,B}$ varying n_0 from 0 to 5 (draw all six curves in the same plot).

HINT: Read and understand the functions defined below. Use them to solve the following tasks! They use the following definitions:

$$CL_{SB} = \sum_{n=0}^{n_0} P(n|\nu_{t,S} + \nu_{t,B}), \tag{3}$$

$$CL_B = \sum_{n=0}^{n_0} P(n|\nu_{t,B}), \tag{4}$$

$$CL_S = CL_{SB} \,/\, CL_B. \tag{5}$$

[4]:
```
def get_x(f, y0, x_min, x_max, kw_params=None, n_int=1000):
    """
    Returns an approximation of the x value of the function f for a given y␣
 ↪value (y0).
    The paremeter params is a dictionary of keyword-value pairs, which are␣
 ↪passed on to the function f.
    Requires f beeing invertible (monotonic, ...) on the given inervall.
    """

    x = np.linspace(x_min, x_max, n_int)

    if kw_params is None:
        y = np.array([f(x_i) for x_i in x])
    else:
        y = np.array([f(x_i, **kw_params) for x_i in x])

    f_inv = scipy_interp1d(y, x, fill_value="extrapolate")

    return f_inv(y0)
```

1.0.4 90 % CL Upper Limit

[5]:
```
from scipy.optimize import minimize
from scipy.special import factorial

def exercise7_2a_90cl_upper_limit(method=0):
    n0 = 3
    def P(n, nu):
        return nu**n*np.exp(-nu)/factorial(n)
    m = get_x(lambda nu_ul: np.sum(P(np.arange(4),nu_ul)), 0.1, 0, 10)
    display(m)
    print(f"Die {1-np.sum(P(np.arange(4), m))}-Obergrenze ist bei nu_t={m}")

exercise7_2a_90cl_upper_limit()
```

```
array(6.68078975)
```

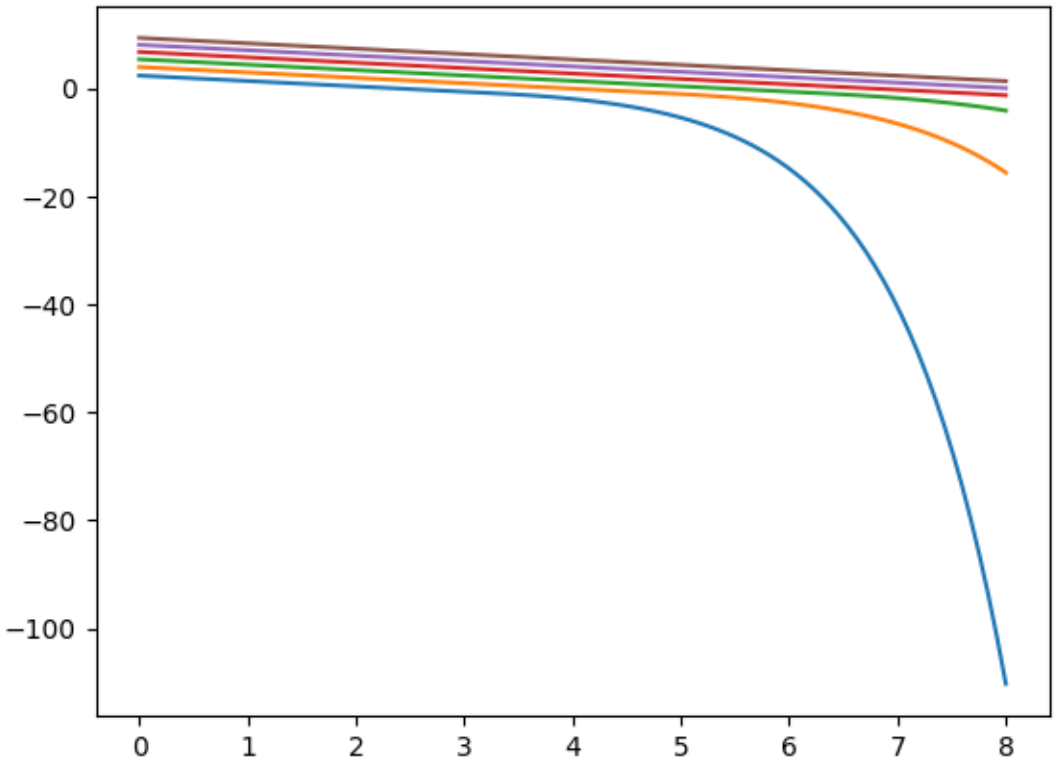
```
Die 0.9000004165502344-Obergrenze ist bei nu_t=6.680789748072546
```

1.0.5 CL_{SB} Limit

[6]:
```python
def get_poisson_probability(nu_t_s, nu_t_b, n0):
    """
    Calculates the probability to observe n0 or less signal events for given␣
 ↪expected signal and expected background
    @param nu_t_s: expected signal
    @param nu_t_b: number of expected background events
    @param n0: number of observed events
    """
    return poisson.cdf(k=n0, mu=nu_t_s + nu_t_b, loc=0)

def get_upper_poisson_limit(n0, nu_t_b, cl):
    """
    returns the upper poisson limit (classical)
    @param n0: number of observed events
    @param nu_t_b: expected background
    @param cl: confidence level
    """

    # targeted pvalue
    p_val = 1 - cl

    # probability to observe n0 or less events

    # interpolation limits
    x_min = -1
    x_max = 10 * (n0 + 1)

    limit = get_x(
        f=get_poisson_probability,
        y0=p_val,
        x_min=x_min,
        x_max=x_max,
        kw_params={"nu_t_b": nu_t_b, "n0": n0},
    )
    return limit
```

[7]:
```python
def exercise7_2a_Csb():
    nu_t_b_space = np.linspace(0,8,100)

    for n0 in range(6):
        plt.plot(nu_t_b_space, [get_upper_poisson_limit(n0, nu_t_b, 0.9) for␣
 ↪nu_t_b in nu_t_b_space])
    plt.show()
```

```
    # TODO: Use the predefined funcitons above to solve the exercise.
    # TODO: Create plots for 100 points varying the background from 0 to 8
exercise7_2a_Csb()
```

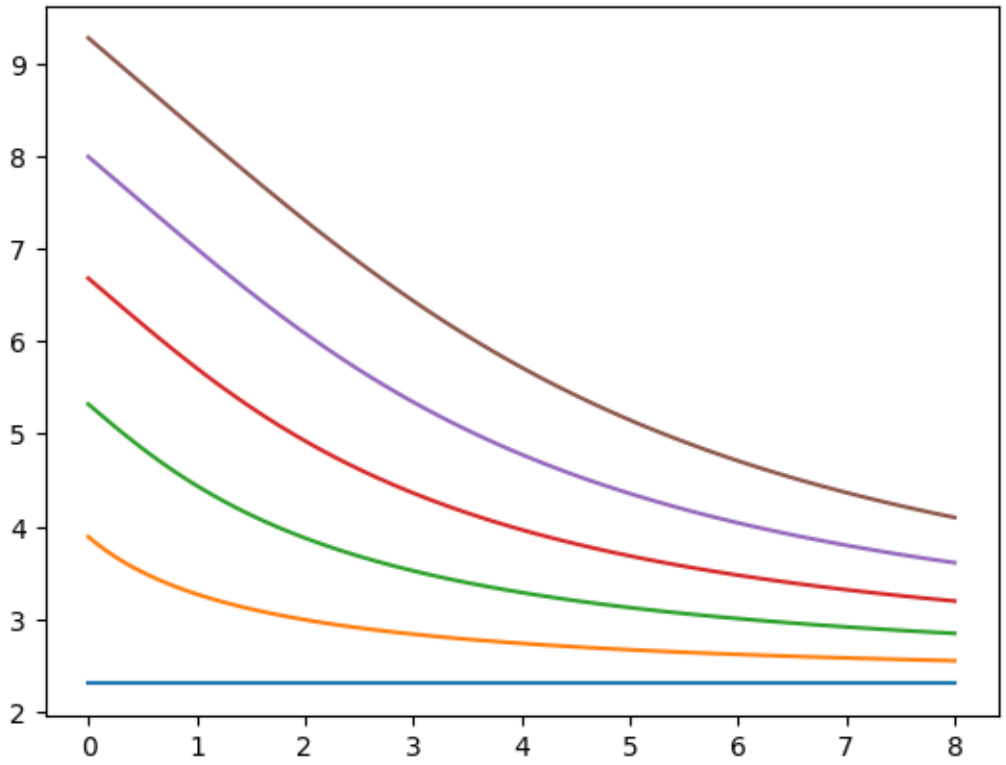


1.0.6 CL_S Limit

[8]:
```
def get_normalized_probability(nu_t_s, nu_t_b, n0):
    """
    Calculates the probability, to observe n0 or less signal events for given␣
 ↪expected signal and expected background
    @param nu_t_s: expected signal
    @param nu_t_b: number of expected background events
    @param n0: number of observed events
    """

    # p_value for signal+background hypothesis
    p1 = poisson.cdf(k=n0, mu=nu_t_b + nu_t_s, loc=0)

    # p_value for background only hypothesis
    p2 = poisson.cdf(k=n0, mu=nu_t_b)
```

```
    return p1 / p2

def get_upper_poisson_limit_normalized(n0, nu_t_b, cl):
    """
    Returns the normalized upper poisson limit (classical)
    @param n0: number of observed events
    @param nu_t_b: expected background
    @param cl: confidence level
    """

    # targeted pvalue
    p_val = 1 - cl

    # interpolation limits
    x_min = -1
    x_max = 10 * (n0 + 1)

    limit = get_x(
        f=get_normalized_probability,
        y0=p_val,
        x_min=x_min,
        x_max=x_max,
        kw_params={"nu_t_b": nu_t_b, "n0": n0},
    )

    return limit
```

[9]:
```
def exercise7_2a_Cs():
    # TODO: Use the predefined funcitons above to solve the exercise.
    # TODO: Create plots for 100 points varying the background from 0 to 8
    nu_t_b_space = np.linspace(0,8,100)

    for n0 in range(6):
        plt.plot(nu_t_b_space, [get_upper_poisson_limit_normalized(n0, nu_t_b,␣
 ↪0.9) for nu_t_b in nu_t_b_space])
    plt.show()

exercise7_2a_Cs()
```



1.0.7 b) Likelihood approach

The likelihood function for a Poisson process for one single measurement is:

$ L(n_0|t) = t^{n_0} e^{-\nu_t} \frac{}{n_0!.}$

where $n_0 = 3$ is the number of measured events.

- Draw the curve $-2 \ln L$ as function of ν_t, performing a scan over a meaningful range of values. Where is the minimum, $-2 \ln L_{min}$, of this curve?
- Calculate the boundaries of the confidence interval at 68% CL. These are the points for which $2 \cdot \Delta \ln L = 1$, where $\Delta \ln L$ is defined as $\ln L_{min} - \ln L$.
- Determine the 90% CL upper limit, i.e. the point with $\nu_t > n_0$ for which $2 \cdot \Delta \ln L = (1.28)^2$.
- Plot the likelihood function and the convidence intervals of interest.

Note: To translate a CL into the proper $\Delta \ln L$ cut for a **one-sided** interval, you can use the equation

$ 2 ln L = (\sqrt{2} \cdot \texttt{erfinv}(2 \cdot CL - 1))\hat{}2$,

where `erfinv` is the inverse of the error function as provided for instance by `scipy` as `scipy.special.erfinv`.

The term which is passed on as argument to the `erfinv` function is equal to $1-2\cdot(1-CL) = 2\cdot CL-1$.

[10]:
```
from scipy.special import erfinv
from scipy.optimize import minimize, root_scalar
```

Hint: You can use the `root_scalar` function provided by `scipy.optimize.root_scalar` to find the intersects.

[11]:
```
print(f"2 * DeltaL = (\sqrt(2) * erfinv(0.68...))^2 = ({np.sqrt(2) * erfinv(0.
↪68268949214)})^2")
```

```
2 * DeltaL = (\sqrt(2) * erfinv(0.68…))^2 = (1.0000000000060216)^2
```

[12]:
```
print(f"2 * DeltaL = (\sqrt(2) * erfinv(2 * 0.9 - 1))^2 = ({np.sqrt(2) *␣
↪erfinv(2 * 0.90 -1)})^2")
```

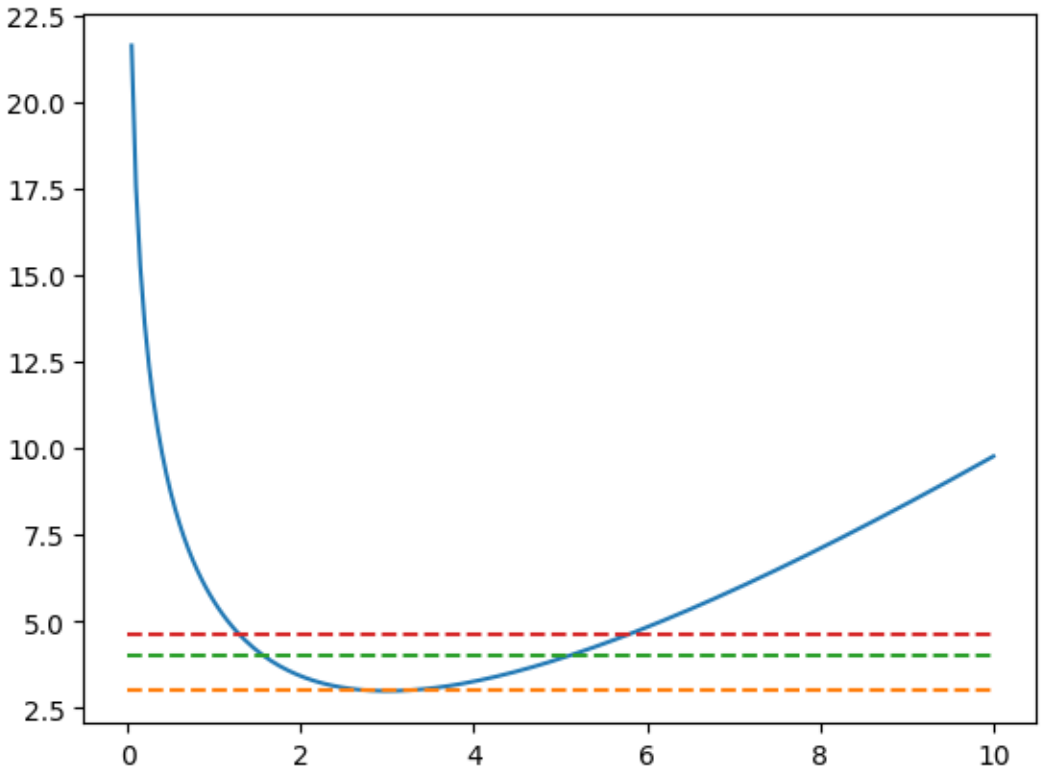
```
2 * DeltaL = (\sqrt(2) * erfinv(2 * 0.9 - 1))^2 = (1.2815515655446006)^2
```

[13]:
```
def negative_log_likelihood(nu_t, n0):
    """
    Returns 2 * negative log likelihood for poisson pdf
    @param nu_t: number of expected events
    @param n0: number of observed events
    @return: NNL
    """

    return -2. * poisson.logpmf(k=n0, mu=nu_t, loc=0)
```

[14]:
```
def exercise7_2b():
    minimum = minimize(lambda nu_t: negative_log_likelihood(nu_t, 3), x0=(3,))
    print(minimum)

    t = np.linspace(0,10,200)
    plt.plot(t, negative_log_likelihood(t, 3))
    plt.plot(t, np.full(t.shape, minimum.x[0]),"--")
    plt.plot(t, np.full(t.shape, minimum.x[0]+1),"--")
    plt.plot(t, np.full(t.shape, minimum.x[0]+(1.28)**2),"--")
    plt.show()

    a1 = minimize(lambda nu_t: np.abs(negative_log_likelihood(nu_t, 3)-minimum.
↪x[0]-1), x0=(2,)).x[0]
    b1 = minimize(lambda nu_t: np.abs(negative_log_likelihood(nu_t, 3)-minimum.
↪x[0]-1), x0=(6,)).x[0]
    print(f"68% confidence in [{a1};{b1}]")

    a1 = minimize(lambda nu_t: np.abs(negative_log_likelihood(nu_t, 3)-minimum.
↪x[0]-1.28**2), x0=(2,)).x[0]
    b1 = minimize(lambda nu_t: np.abs(negative_log_likelihood(nu_t, 3)-minimum.
↪x[0]-1.28**2), x0=(6,)).x[0]
```

```
    print(f"90% confidence in [0,{b1}]")

    # TODO: Implement your solution:
    #       - Get the minimum of the likelihood
    #       - Get the 68% confidence level interval (points where the the␣
 ↪likelihood is by 1 higher than its minimum value
    #       - Get the 90% confidence level upper limit
    #       - Plot the likelihood function

exercise7_2b()
```

```
 message: Optimization terminated successfully.
 success: True
  status: 0
     fun: 2.9918452064474517
       x: [ 3.000e+00]
     nit: 0
     jac: [ 0.000e+00]
hess_inv: [[1]]
    nfev: 2
    njev: 1
```



```
68% confidence in [1.5794271066892471;5.090180265806246]
90% confidence in [0,5.80210060243859]
```

1.0.8 c) Bayesian approach

The Bayesian posterior probability $P(\nu_t|n_0)$ is given by the Bayes theorem:

$$P(\nu_t|n_0) = \frac{L(n_0|\nu_t)\ \Pi(\nu_t)}{\int_{\text{all } \nu_t} L(n_0|\nu_t)\ \Pi(\nu_t) d\nu_t}.$$

$\Pi(\nu_t)$ is called the prior probability on ν_t and describes our prior belief about the distribution of this parameter.

Try two different priors and compare the results:

i) $\Pi(\nu_t) = const.$ for $\nu_t > 0$ and 0 otherwise,

ii) $\Pi(\nu_t)$ proportional to $1/\nu_t$ for $\nu_t > 0$ and 0 otherwise.

- Compute and draw the posterior probability for $n_0 = 3$ and $\nu_{t,B} = 0$.
- What are the 90% credibility upper limits?

[15]:
```
from scipy.integrate import quad
```

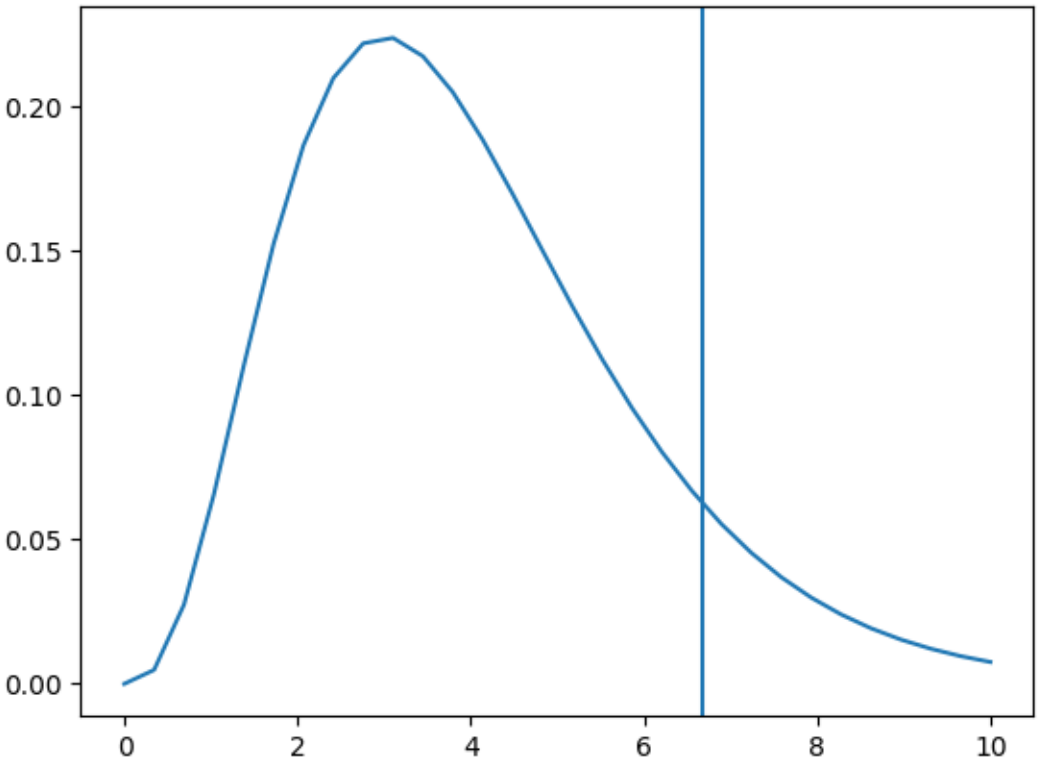
[16]:
```
def likelihood_pdf(nu_t,  n0):
    """
    Returns likelihood for poisson pdf
    @param nu_t: number of expected events
    @param n0: number of observed events
    @return: NNL
    """

    return poisson.pmf(k=n0, mu=nu_t, loc=0)
```

[17]:
```
def prior_flat_function(nu_t):
    return nu_t > 0


def prior_inverse_function(nu_t):
    return (1/nu_t) * (nu_t > 0)


def posterior_function_with_flat_prior(nu_t_list, n0):
    return np.array([likelihood_pdf(nu_t,  n0)*prior_flat_function(nu_t)/
 ↪(quad(lambda tx :likelihood_pdf(tx,  n0)*prior_flat_function(tx), 0,np.
 ↪inf)[0]) for nu_t in nu_t_list])


def posterior_function_with_inverse_prior(nu_t_list, n0):
```

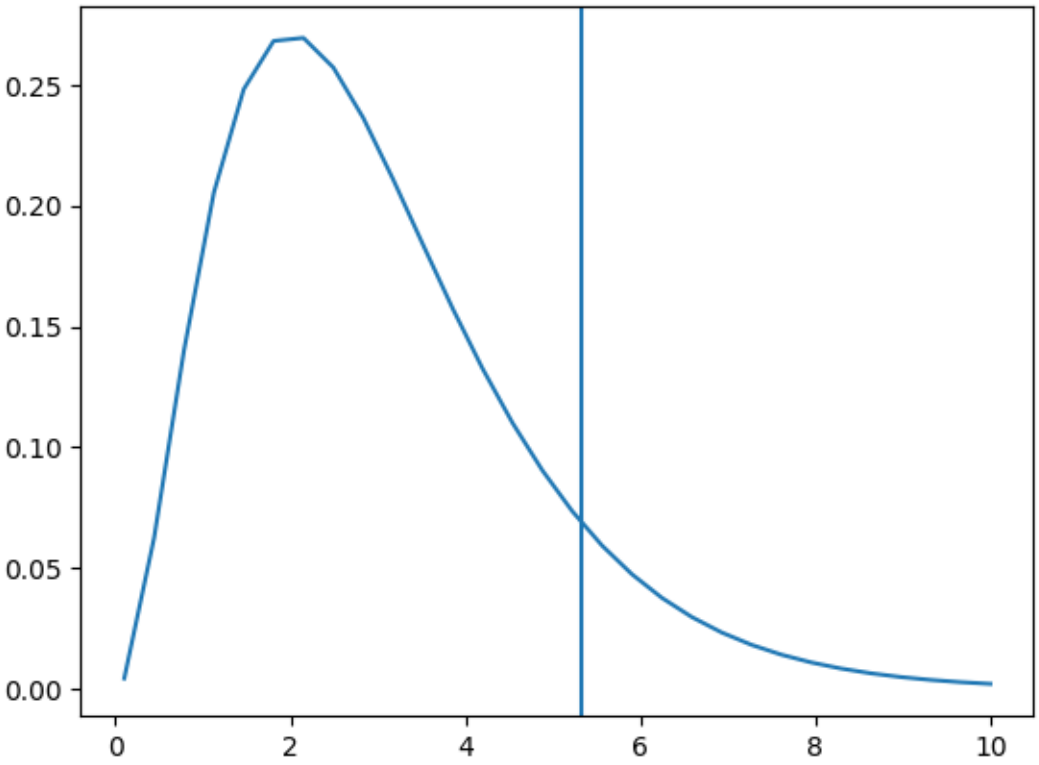
```
    return np.array([likelihood_pdf(nu_t,  n0)*prior_inverse_function(nu_t)/
↪(quad(lambda tx :likelihood_pdf(tx,  n0)*prior_inverse_function(tx), 0,np.
↪inf)[0]) for nu_t in nu_t_list])
```

[25]:
```
def exercise7_2c():
    t = np.linspace(0,10,30)
    plt.plot(t,posterior_function_with_flat_prior(t,3))
    div = quad(lambda tx :likelihood_pdf(tx,  3)*prior_flat_function(tx), 0, np.
↪inf)[0]
    percentage = lambda x: quad(lambda tx :likelihood_pdf(tx, ␣
↪3)*prior_flat_function(tx), 0, x)[0] / div
    m = minimize(lambda x: np.abs(percentage(x)-0.9), x0=(4,) )
    print(f"Die {percentage(m.x[0])}-Obergrenze ist bei nu_t={m.x[0]}")
    plt.axvline(m.x[0])
    plt.show()

    t = np.linspace(0.1,10,30)
    plt.plot(t,posterior_function_with_inverse_prior(t,3))
    div = quad(lambda tx :likelihood_pdf(tx,  3)*prior_inverse_function(tx), 0,␣
↪np.inf)[0]
    print(div)
    percentage = lambda x: quad(lambda tx :likelihood_pdf(tx, ␣
↪3)*prior_inverse_function(tx), 0, x)[0] / div
    m = minimize(lambda x: np.abs(percentage(x)-0.9), x0=(4,) )
    print(f"Die {percentage(m.x[0])}-Obergrenze ist bei nu_t={m.x[0]}")
    plt.axvline(m.x[0])
    plt.show()



    # TODO: Implement your solution:
    #       - Calculate the posterior probabilities
    #       - Draw the posterior probabilities
    #       - Determine the 90 % credibility upper limits


exercise7_2c()
```

```
Die 0.8999999996988178-Obergrenze ist bei nu_t=6.680783063426109
```




```
0.33333333333333687
Die 0.8999999994848809-Obergrenze ist bei nu_t=5.322320330383766
```



Finally, compare the upper limits of the methods a), b), c) for $n_0 = 3$ and $\nu_{t,B} = 0$

[21]:
```
nu_t_a  = 6.680789748072546
nu_t_b  = 5.80210060243859
nu_t_c1 = 6.680783063426109
nu_t_c2 = 5.322320330383766
```

Die Obergrenze bei inversem Prior ist minimal.