University of Zurich, Physics Institute
Report for the module PHY231, fall semester 2018

Group project I
Characterization of a radioactive source

Stefanie Jucker - stefanie.jucker2@uzh.ch
Andrej Maraffio - andrej.maraffio@uzh.ch
Mirko Mirosavljevic - mirko.mirosavljevic@uzh.ch
Manuela Rohrbach - manuela.rohrbach@uzh.ch
Stefan von Rohr - stefan.vonrohr@uzh.ch

December 11, 2018

Contents

1 Introduction

Simulations of experiments are widely used in physics to develop the tools for the data analysis of real experimental data. In this project, we simulate a radioactive decay experiment.

1.1 Theory

There are discrete energy levels an atomic nucleus can be on, some of which are unstable. If a nucleus is in an unstable state, there is a certain chance it will emit energy in the form of radiation and particles and thus reach a lower energy level. This may happen several times until a stable state is reached. This process is known as radioactive decay.

There are three different types of radioactive decay, which are the α-, β- and γ-decay. In the α-decay, an atomic nucleus emits an α-particle (helium nucleus) and thereby creates a new nucleus with a different mass and atomic number. The mass number is reduced by four and the atomic number is reduced by two.

$$^{A}_{Z}X \rightarrow ^{A-4}_{Z-2} Y + ^{2}_{4} He$$

In the β-decay, an atomic nucleus emits a β-particle (energetic electron or positron) and a neutrino. After the decay, the nucleus has a different atomic number which is increased or lowered by one.

$$^{A}_{Z}X \rightarrow ^{A}_{Z+1} Y + e^{-} + \overline{v}_e \text{ resp. } ^{A}_{Z}X \rightarrow ^{A}_{Z-1} Y + e^{+} + v_e$$

In the γ-decay, the nucleus does not change the number of its neutrons and protons, but jumps from a high energy state to a lower one by emitting a photon.

$$^{A}_{Z}X^{*} \rightarrow ^{A}_{Z} X + \gamma$$

These decays emit different types of radiation, which are measurable. From the measured decays, one can determine the mean lifetime, the half-life and the activity of the source. Since every radioactive isotope has a unique half-life, determining the half-life allows the identification of the isotopes in an unknown source. In this project we simulate the data of the β-decay of an ^{192}Ir (iridium) source.

1.2 Experimental set-up

The experimental set-up (Fig. 1) consists of the radioactive source ^{192}Ir that emits electrons through a β-decay and a planar detector with dimensions of 4 cm by 5 cm in the x and y-direction, respectively. The source is surrounded by an absorbing material allowing the electrons to escape only through a small hole pointing towards the detector. Due to the shape of this hole, the spatial distribution of the emitted electrons on the detector plane follows a two dimensional Gaussian profile with means $\mu_x = \mu_y = 0$ and standard deviations $\sigma_x = 3\,\text{cm}$ and $\sigma_y = 6\,\text{cm}$. The x and y positions are not correlated.

During the detector registered the decays over a period of 60 days and was read out once every 24 hours.

1.3 Aims and approach

The aim of the project is to simulate the experiment as described above in order to determine the mean lifetime and the activity of the ^{192}Ir source. We used the following approach:

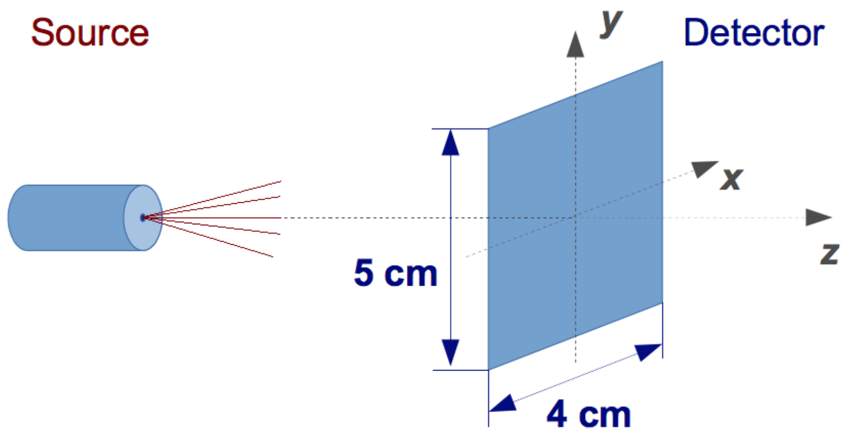

Figure 1: *A schematic representation of the experimental set-up consisting of a shielded source and a planar detector at an unknown distance from the source.*

- We simulated a data set and used the maximum-likelihood method to determine the mean lifetime and the corresponding error.
- We simulated the emitted electrons to calculate the proportion registered by the detector - the geometric acceptance of the detector. From this, we calculated the activity of the source.

2 Methods

2.1 Measurement of the mean lifetime

The first task was to generate a simulated measurement data set for a detector that counted the number of electrons impacting on it. The detector was read out once every 24 hours over a total period of 60 days. An additional constraint was imposed that the total number of decays after the 60 days measurement period should be equal to $N_{\text{tot}} = 113'809$.

Therefore, we wanted to generate N_{tot} decay times that were distributed exponentially between $t_0 = 0\,\text{d}$ and a maximum of $t_{\text{end}} = 60\,\text{d}$, which is described by the following probability distribution

$$p(t|\tau) = \frac{\frac{1}{\tau}\exp\left(-\frac{t}{\tau}\right)}{1-\exp\left(-\frac{t_{\max}}{\tau}\right)}. \tag{1}$$

The corresponding cumulative density function (Sec.A.1 lines 39 - 46) is then given as

$$P(t|\tau) = \int_{-\infty}^{t} p(t)\mathrm{d}t = \frac{1-\exp\left(-\frac{t}{\tau}\right)}{1-\exp\left(-\frac{t_{\max}}{\tau}\right)}. \tag{2}$$

In order to find a value t for the lifetime from a probability P, we need the inverse of the cumulative density function (Sec.A.1 lines 49 - 57), which is

$$t(P) = -\tau \cdot \ln\Big(1 - P \cdot (1 - \mathrm{e}^{-\frac{t_{\max}}{\tau}})\Big) \tag{3}$$

The last thing we needed to simulate the lifetimes is the true mean lifetime. We already know the true half life of ^{192}Ir, which is (73.810 ± 0.019) days [1]. The mean lifetime is then easily found with the following equation:

$$t_{1/2} = \tau \ln(2) \rightarrow \tau = \frac{t_{1/2}}{\ln(2)}. \tag{4}$$

So we generated N_{tot} random numbers following a uniform distribution in the interval from 0 to 1 (Sec.A.1 lines 60 - 89). By interpreting these values as probabilities, the values for the mean lifetimes were then obtained by plugging the values into eq. 3. Then, in order to obtain the measurement data points, these values were histogrammed in a cumulative way, since the detector was not reset between measurements. A visualization of the resulting data can be seen in Fig.2.

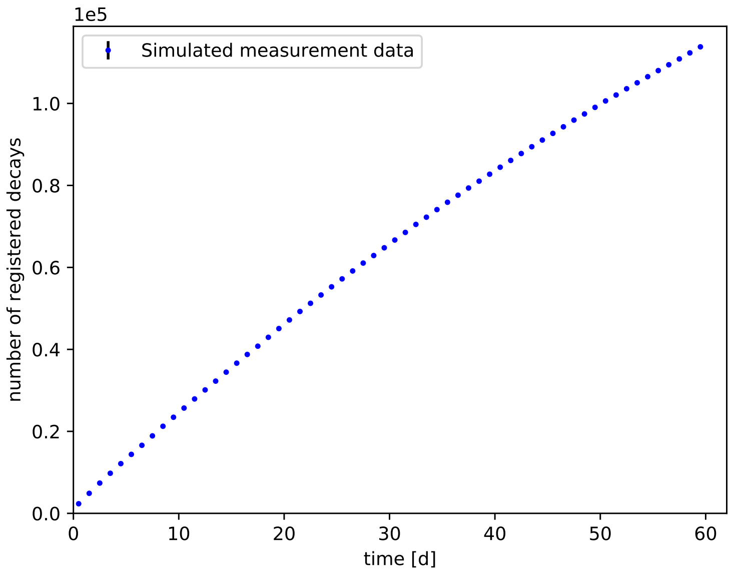

Figure 2: *Generated cumulative data set with error bars. The error bars are smaller than the size of the data points.*

As a next step, we wanted to compute an estimate $\hat{\tau}$ for the mean lifetime τ. In order to do so, the data had to be sliced to obtain the measured decays per day (Sec.A.1 lines 92 - 102), which yields $n_1, \dots, n_{60}$ decays in the respective time intervals $[t_0, t_1], \dots, [t_{59}, t_{\text{end}}]$.

We then computed an estimate for the mean lifetime by means of a maximum likelihood method (Sec.A.1 lines 105 - 130). The log-likelihood for a value of the parameter $\tilde{\tau}$ for binned data is given by

$$\ln L(\tilde{\tau}) = \sum_{i=1}^{60} \frac{1}{n_i!} + \sum_{i=1}^{60} (n_i \cdot \ln v_i(\tilde{\tau}) - v_i(\tilde{\tau})) \tag{5}$$

where $v_i(\tilde{\tau})$ describes the number of expected decays in the time interval $[t_{i-1}, t_i]$. It can be found with the following equation

$$v_i(\tilde{\tau}) = N_{\text{tot}} \int_{t_{i-1}}^{t_i} p(t|\tilde{\tau}) = N_{\text{tot}} \cdot \{P(t_i|\tilde{\tau}) - P(t_{i-1}|\tilde{\tau})\}. \tag{6}$$

After obtaining an estimate for the mean lifetime $\hat{\tau}$ (Sec.A.1 lines 133 - 154), its error $\sigma_{\hat{\tau}}$ was determined as seen in the lecture from the width of the parabola given as

$$\ln L(\hat{\tau} \pm \sigma_{\hat{\tau}}) = \ln L(\hat{\tau}) - 0.5. \tag{7}$$

Lastly, we computed the pull distribution over 1000 simulated experiments (Sec.A.1 lines 190 - 212) to determine whether or not our simulated data is accurate. The pull is calculated by the following formula:

$$\text{pull} = \frac{\text{reconstructed quantity - generated quantity}}{\text{uncertainity on reconstructed quantity}}. \tag{8}$$

2.2 Determination of the total activity

In this part, the first task consisted of determining the geometric acceptance of the detector and the corresponding uncertainty. As stated in the instructions, the impact positions of the electrons were assumed to be distributed according to a two-dimensional Gaussian distribution.

In order to determine the geometric acceptance of the detector, the impact positions of 10'000 electrons in the plane of the detector were simulated using a Monte Carlo method (Sec.A.2 lines 44 - 86 with `systematic_position_error=False`). The acceptance a was found as the fraction of all electrons that landed within the bounds of the detector according to

$$a = \frac{\text{number of electrons landing on the detector}}{\text{total number of electrons}} \tag{9}$$

By repeating this procedure 1000 times, it was possible to find a mean value and a distribution for the geometric acceptance.

The geometric acceptance for a misaligned detector with a systematic error of $\pm 0.2\,\text{cm}$ on the position was determined in the same way as the geometric acceptance with the detector perfectly aligned, except that the detector edges were shifted by a systematic error (Sec.A.2 lines 44 - 86 with `systematic_position_error=True`). The systematic error following a uniform distribution was generated using a Monte Carlo method for each repetition.

In order to visualize the distribution of the impact positions of the electrons in relation to the detector, we plotted the boundaries of the detector on top of a heatmap of one simulation cycle with a million electrons (Sec.A.2 lines 89 - 119). This means, that the impact area was divided into a grid of small blocks and the number of electrons landing in each block were counted by means of a two-dimensional histogram. Each block was then assigned a color according to its electron count, where brighter colors represent a higher count.

As a next step, the value of the acceptance was used to compute the activity of the source at the time of our first reading $t_1 = 1\,\text{d}$. The activity A of the source is

$$A = \frac{\mathrm{d}N}{\mathrm{d}t} \tag{10}$$

where N is the number of electrons emitted by the source. However, due to its geometrical properties, the detector only measures a fraction of all emitted electrons and thus an activity of $\frac{\mathrm{d}N_D}{\mathrm{d}t} = a\frac{\mathrm{d}N}{\mathrm{d}t}$. Thus, the activity is $A = \frac{1}{a} \cdot \frac{\mathrm{d}N_D}{\mathrm{d}t}$.

The source emits an exponentially decreasing number of electrons over time given by

$$N\left(t\right) = \tilde{N}_0 \left(1 - \mathrm{e}^{-\frac{1}{\tau}t}\right) \tag{11}$$

where $\tilde{N}_0$ is the initial number of iridium atoms in the source. Since we know the total number of emitted electrons N_{tot} after $t_{\mathrm{end}} = 60\,\mathrm{d}$, we can calculate $\tilde{N}_0$. Thus, the activity of the source at t_1 is

$$A = \frac{1}{a} \cdot \frac{\mathrm{d}N_D}{\mathrm{d}t} = \frac{1}{a}\left(\frac{N_{\mathrm{tot}}}{1 - \mathrm{e}^{-\frac{1}{\tau}t_1}}\right)\frac{1}{\tau}\mathrm{e}^{-\frac{1}{\tau}t_1} \tag{12}$$

The activity (Sec.A.2 lines 122 - 129) was calculated using the values of τ determined in Sec. 3.1.

The error on the activity can be determined using error propagation. The errors on the mean life time τ and on the acceptance a were computed as described above. Since the total number of decays is Poisson distributed, the error can be found as $\sigma_{N_{\mathrm{tot}}} = \sqrt{N_{\mathrm{tot}}}$. For the error propagation we assumed, that the geometric acceptance a is not correlated with the mean life time τ or the total number of decays N_{tot}, because they belong to two entirely separate systems. The geometry of the detector and, thus, the acceptance does not depend on characteristics of the source. However, we expect some correlation between the mean life time and the total number of decays. The error on the activity (Sec.A.2 lines 146 - 159) is

$$\sigma_A^2 = JVJ^{\mathrm{T}} \tag{13}$$

where J is the Jacobian of A and V is the covariance matrix. The Jacobian at t_1 is

$$J = \begin{pmatrix} \frac{\partial A}{\partial a} & \frac{\partial A}{\partial \tau} & \frac{\partial A}{\partial N_{\mathrm{tot}}} \end{pmatrix} = \begin{pmatrix} \frac{N_{\mathrm{tot}}}{a^2\tau - a^2\mathrm{e}^{t_1/\tau}\tau} & \frac{N_{\mathrm{tot}}\left(\mathrm{e}^{t_1/\tau}(t_1-\tau)+\tau\right)}{a\left(-1+\mathrm{e}^{t_1/\tau}\right)^2\tau^3} & -\frac{1}{a\tau - a\mathrm{e}^{t_1/\tau}\tau} \end{pmatrix}. \tag{14}$$

The covariance matrix is

$$V = \begin{pmatrix} \sigma_a^2 & 0 & 0 \\ 0 & \sigma_\tau^2 & \mathrm{cov}\left(\tau, N_{\mathrm{tot}}\right) \\ 0 & \mathrm{cov}\left(\tau, N_{\mathrm{tot}}\right) & \sigma_{N_{\mathrm{tot}}}^2 \end{pmatrix}. \tag{15}$$

The covariance of τ with N_{tot} was determined using a Monte Carlo method to generate the Poisson distributed number of total decays with a mean equal to the given $N_{\mathrm{tot}} = 113809$ decays (Sec.A.2 lines 131 - 144). The life time was then estimated using the method described in Sec. 2.1 for each of the generated number of decays. The covariance was then calculated according to

$$\mathrm{cov}\left(\tau, N_{\mathrm{tot}}\right) = \frac{1}{n-1}\sum_i^n \left(N_{\mathrm{tot},i} - \overline{N_{\mathrm{tot}}}\right)\left(\tau_i - \bar{\tau}\right). \tag{16}$$

3 Results and Discussion

3.1 Measurement of the mean lifetime

The estimated mean lifetime for our experiment is $(107.8 + 2.0/-1.9)$d for a single run of the experiment, which corresponds well with the literature value of (106.49 ± 0.03)d [1]. The

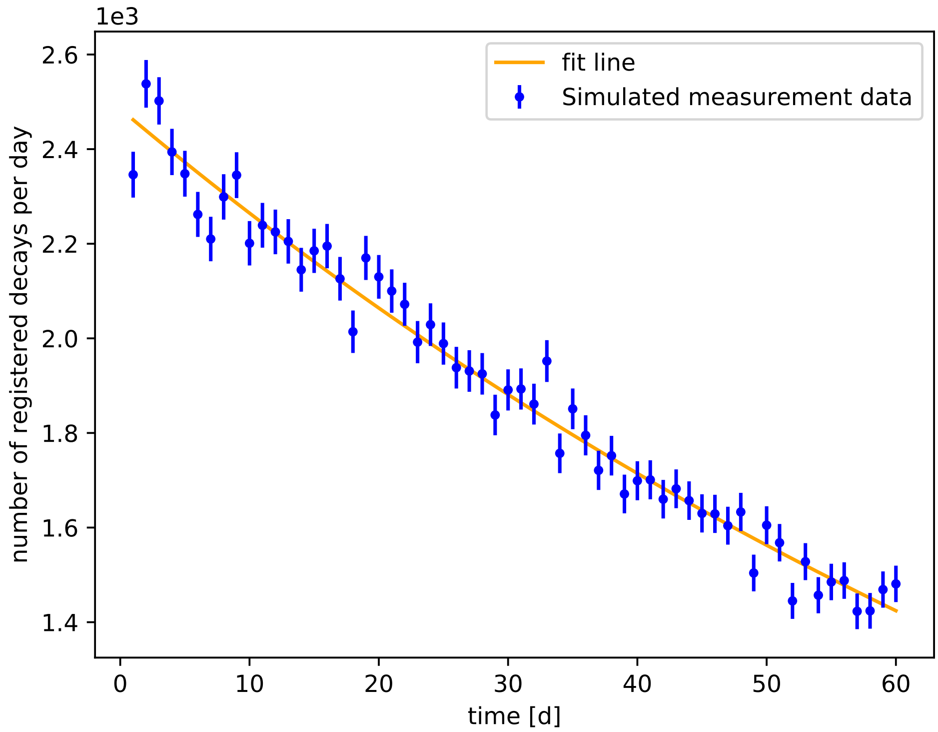

Figure 3: *Visualization of simulated data-set: Declining number of registered decays with fitted line.*

simulated measurement data follows an exponential decay and fits well to the fit line calculated with the estimated mean lifetime as can be seen in Fig.3.

The distribution of the estimated mean lifetime for 10000 repetitions of the experiment seems to follow a Gaussian distribution (Fig.4), which is to be expected due to the central limit theorem. This is further substantiated by the mean and standard deviation of the pull calculated for 1000 repetitions of the experiment. The obtained values of -0.07 and 0.99 for the mean and standard deviation, respectively, of the pull coincide with our expectations of 0 for the mean and 1 for the standard deviation (Fig. 5). This tells us, that the simulation behaves according to our expectations.

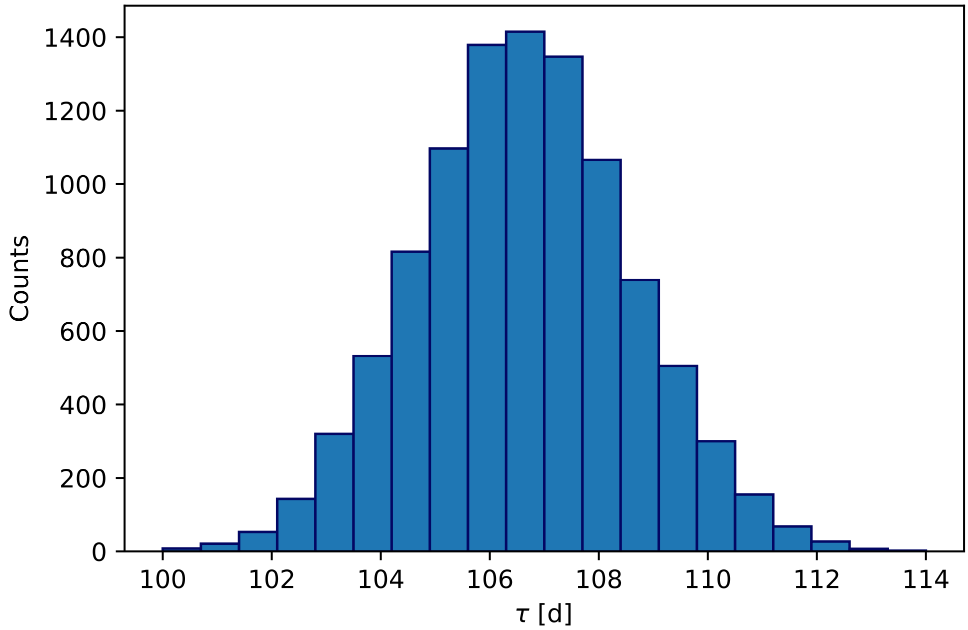

Figure 4: *Distribution of estimated taus for 10000 repetitions of the experiment.*

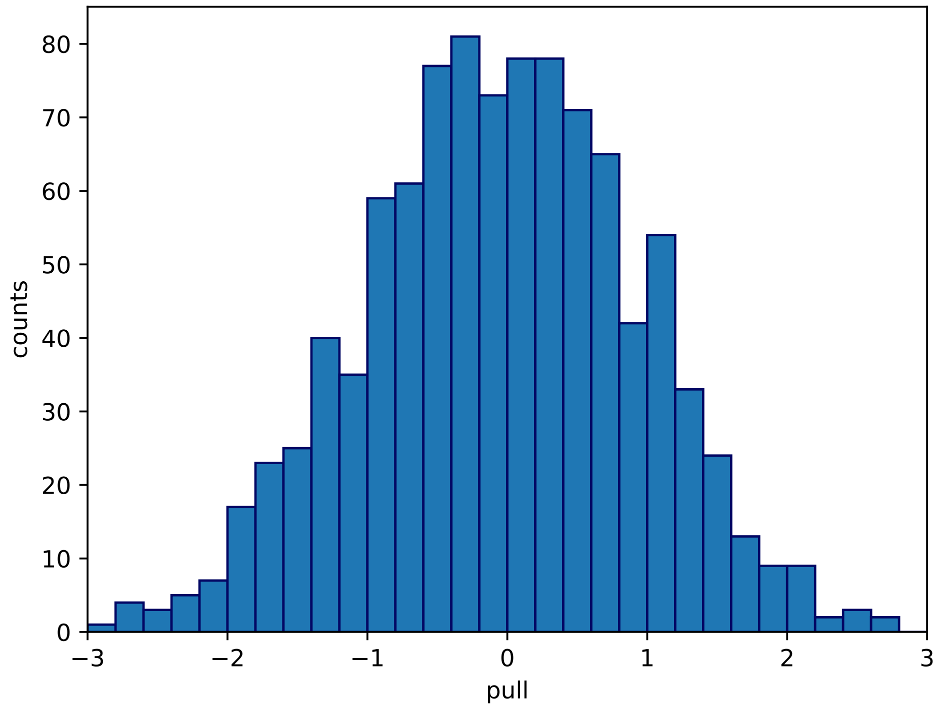

Figure 5: *Visualization of the pull for 1000 repetitions of the experiment. The mean is -0.07 and the standard deviation is 0.99.*

3.2 Determination of the total activity

As can be seen in the obtained heatmap of the impact position of the electrons, the detector covers only a small area of the distribution of the emitted electrons (cf. Fig. 6). For the perfectly aligned detector, the geometric acceptance found from 1000 repetitions is $a = 0.160 \pm 0.004$.

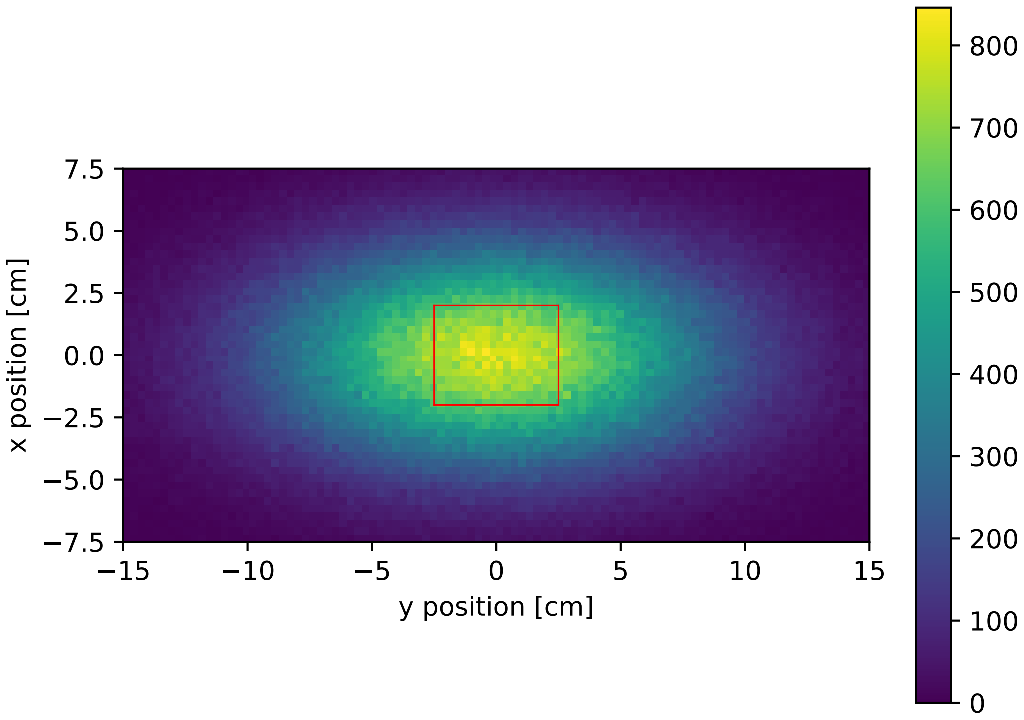

Figure 6: *A 2D histogram of 1'000'000 emitted electrons. The red rectangle shows the size and position of the detector.*

However, if we assume that the position of the detector is only known within $\pm 0.2\,\mathrm{cm}$ for each repetition, then the error on the geometric acceptance becomes $\sigma_a = 0.04$. This means that 1000 laboratories who each have their detector aligned with an accuracy of $\pm 0.2\,\mathrm{cm}$ will see a spread in their geometric acceptance of $\sigma_a = 0.04$ and a mean acceptance of $a = 0.13$.

The spread in the geometric acceptance with a systematic error on the detector alignment is an order of magnitude larger than the error on the perfectly aligned detector. Thus, if the detector is misaligned, there is a high chance that the experiment will not determine an accurate acceptance. In particular, the obtained acceptance will tend to be smaller, because the error shifts the detector away from the center of the Gaussian distributions where the electron density is the highest. The difference between the two obtained values also show this $(0.13 < 0.16)$.

The activity of the source at $t_1 = 1\,\mathrm{d}$ for a perfectly aligned detector is $A = (0.177 \pm 0.003)\mathrm{Bq}$. The error was calculated including a correlation term $\mathrm{cov}\,(\tau, N_{\mathrm{tot}})$. As an aside, if we assume no correlation and only use Gaussian error propagation, the error is nearly the same and is affected by the correlation term only in the sixth decimal place. It is important to note that the obtained value is not the activity of the ^{192}Ir source itself, but rather the activity of the shielded source with one small hole in the shielding through which the electrons are emitted.

Since the activity of the source is inversely proportional to the geometric acceptance, the activity determined using a misaligned detector is not accurate. And since the acceptance tends to be smaller for the misaligned detector, the activity tends to become larger. The activity of the source at $t_1 = 1\,\mathrm{d}$ for a misaligned detector is $A = (0.23 \pm 0.04)\mathrm{Bq}$.

However, the mean life time τ can still be determined accurately, even if the detector is misaligned. Because τ is determined from the slope of the logarithmized number of decay events over a time range, the acceptance of the detector simply shifts each of the measurement points by the factor $\ln(a)$, thus

$$\ln\left(\hat{N}(t)\right) = \ln\left(\tilde{N}_0 - N(t)\right) = \ln\left(\frac{\tilde{N}_0}{a}\right) - \frac{1}{\tau}t = \ln\left(\tilde{N}_0\right) - \ln(a) - \frac{1}{\tau}t \tag{17}$$

This is also the reason, why we set the covariance of τ and a to $\mathrm{cov}(\tau, a) = 0$. The total numbers of decays N_{tot} is simply the number of decays measured by the detector. It is not the real number of decays $\frac{1}{a}N_{\mathrm{tot}}$, thus it also not affected by the acceptance and we set the covariance to $\mathrm{cov}(N_{\mathrm{tot}}, a) = 0$.

4 Conclusion

The goal of this project was to simulate an experiment to characterize a radioactive source, which was carried out in two parts. In the first part, we used Monte-Carlo methods to simulate a data set of the radioactive decay of ^{192}Ir. The mean lifetime of the simulated radioactive source was then determined using a binned maximum likelihood method. The estimated mean lifetime of the simulated data is (107.8 ± 2.0)d which agrees with the reference value of (106.49 ± 0.03)d.

In the second part, we determined the activity of the source under the assumption that the emitted electrons follow a 2D Gaussian distribution on the detector. The geometric acceptance of the perfectly aligned detector was determined using a Monte Carlo simulation for the 2D-Gaussian distribution. The geometric acceptance is 0.160 ± 0.004 which leads to a source activity of $A = (0.177 \pm 0.003)\mathrm{Bq}$ after 1 day. The same process was repeated for a slightly misaligned detector using a Monte Carlo simulation of a uniformly distributed offset of the detector center. The geometric acceptance of the misaligned detector was 0.13 ± 0.04 which leads to a source activity of $A = (0.23 \pm 0.04)\mathrm{Bq}$ after 1 day.

4.1 A note about efficiency

While the main aim was merely to produce a running code and not a runtime optimization, it is clear that a short runtime is desirable especially for parts including many repetitions of a function call such as in the pull. In our initial working code, the function simulating the measurement data took around 8 seconds to execute, while the estimation of the mean life time took around 6 seconds. It is obvious, that calling these functions a hundred or even a thousand times would have taken unbearably long.

Considering that the simulation creates an array with over 100'000 entries, it seemed evident that the array creation and manipulation were the limiting factors. As it turns out, using python's inbuilt lists is significantly less efficient than using numpy arrays. Therefore, by adapting the two mentioned functions such that list comprehensions were eliminated and all arrays were directly created and computed by numpy, their execution times were reduced to around 10 and 50 milliseconds respectively. In total, this allows estimating a thousand mean lifetimes in roughly a minute, which, being more than a hundred times faster than our initial functions, is a drastic improvement.

4.2 Future work

In this simulation study, we did not calculate the activity of the source itself, but rather of the shielded source. A more complete simulation study would include determining the activity of the source itself with an error estimation. For this calculation, the exact geometry of the source, the distance between the hole through which electrons are emitted and the detector, and whether the source emits electrons in all directions with the same probability (i.e. whether the source is isotropic) need to be known.

Furthermore, we assumed a perfect detector, i.e. each electron that hits the detector is registered. However, in reality detectors are not perfect. There is often a dead time in the electronic parts during which a detector is reset after a detection. During this time no electrons can be detected. However, our experiment is unlikely to be strongly affected by this, since the dead time of the electronics is probably less than one second and the source we simulated emits approximately one electron every 5 s.

Additionally, a full simulation of the experiment could be performed repeatedly to determine the error on the activity instead of using error propagation. However, if the assumptions we made in determining the covariance matrix are correct, the error on the activity should be of a similar size.

References

[1] Unterweger, M.P. et al. (2003), *Radionuclide Half-Life Measurements at the National Institute of Standards and Technology* (version 3.0). [Online]
Available: `http://physics.nist.gov/Halflife` [Accessed: Dec 2018].

A Python code

A.1 Measurement of the mean lifetime

```python
# -*- coding: utf-8 -*-
"""
Data Analysis, Fall Semester 2018

Group Project 1: Characterization of a radioactive source, Task 3

Authors: Stefanie Jucker, Andrej Maraffio, Mirko Mirosavljevic,
Manuela Rohrbach, Stefan von Rohr
"""
from math import factorial
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import expon


# Properties of the experiment
total_decays = 113809
reading_intervals = np.array([i for i in np.arange(0, 61)])  # days
true_half_life = 73.81  # days
true_tau = true_half_life/np.log(2)  # days
# Get the middle of the time bins
times = 0.5*(reading_intervals[1:] + reading_intervals[:-1])


def pull_function(true_tau, tau_hat, sig_tau_hat):
    return (tau_hat - true_tau)/sig_tau_hat


#def limited_lifetime_pdf(t, tau, t_max=reading_intervals[-1]):
#    """
#    Probability density function for exponential decay when you measure for
#    a certain time, which limits the possible measured lifetimes
#    """
#    pdf = expon.pdf(t, scale=tau)/expon.cdf(t_max, scale=tau)
#
#    return pdf
#

def limited_lifetime_cdf(t, tau, t_max=reading_intervals[-1]):
    """
    Cumulative density function for exponential decay when you measure for
    a certain time, which limits the possible measured lifetimes
    """
```

```
    cdf = expon.cdf(t, scale=tau)/expon.cdf(t_max, scale=tau)

    return cdf


def inv_limited_lifetime_cdf(u, tau=true_tau, t_end=60):
    """
    Inverse function of the cdf of an exponential with limitied time.
    Finds which t has u = cdf*(t)
    """
    k = expon.cdf(t_end, scale=tau)
    t = -tau*np.log(1-u*k)

    return t


def data_simulation(true_tau, total_decays, reading_intervals):
    """
    Simulate a dataset for the measured decays of the radioactive source

    Parameters
    ----------
    true_tau: float, known mean life time of the source (in days)
    total_decays: int>0, total number of decays to simulate
    reading_intervals: list, boundaries of reading intervals (in days)

    Returns
    -------
    decays: list of int>0, readings of number of decays at the given times
    errors: list of floats, error on decays
    """
    random_uniform = np.random.sample(total_decays)
    lifetimes = inv_limited_lifetime_cdf(random_uniform, true_tau)

    # Count how many decays are in the asked measuring intervals
    # -> Histogram the data in a cumulative way
    decays_per_interval = np.histogram(lifetimes, bins=reading_intervals)[0]

    decays = []
    for i in range(1, len(decays_per_interval) + 1):
        value = sum(decays_per_interval[:i])
        decays.append(value)

    errors = np.sqrt(decays)

```

```
    return decays, errors


def counts_per_day(cumulative_counts):
    """
    Reduce data to decay counts per day from the cumulative experimental data.
    """
    # Get difference between consecutive entries
    reduced_counts = np.diff(cumulative_counts)

    # Add the first measurement unaltered
    reduced_counts = np.insert(reduced_counts, 0, cumulative_counts[0])

    return reduced_counts


def binned_ll(tau_range, hist_values, bin_edges):
    """
    Computes the binned log likelihood for a range of mean lifetimes (tau_range)
    """
    # Total entries in all bins
    total_events = sum(hist_values)

    # Constant addition term for the log likelihood
    const = sum([1/factorial(n) for n in hist_values])

    ll = []

    for tau in tau_range:
        # Expected number of entries in each bin
        vs = total_events*(limited_lifetime_cdf(bin_edges[1:], tau)
                           - limited_lifetime_cdf(bin_edges[:-1], tau))
        summands = hist_values*np.log(vs) - vs
        ll.append(sum(summands))

    # Convert to a numpy array
    ll = np.array(ll)

    # Add the constant term to all list entries
    ll += const

    return ll


def tau_estimator(decays_cumulative, reading_intervals, accuracy=0.1)
```

```
:
    """
    Compute an estimate tau hat for the mean lifetime using a binned
    log likelihood method
    """
    # Get the measured decays per day (not cumulative)
    decays_per_day = counts_per_day(decays_cumulative)

    # Define a range of possible taus and compute the binned log-likelihood
    tau_range = np.arange(100, 115, accuracy)
    log_likelihood = binned_ll(tau_range, decays_per_day, reading_intervals)

    # Determine the estimator through the maximum of the log likelihood
    max_index = np.argmax(log_likelihood)
    tau_hat = tau_range[max_index]

    # Determine an error on the estimate
    log_likelihood -= max(log_likelihood)
    tau_e = tau_range[log_likelihood > -0.5]
    sig_tau_hat = (tau_e.max() - tau_hat, tau_hat - tau_e.min())

    return tau_hat, sig_tau_hat


def fitted_plot(reading_intervals, decays_cumulative, errors, times, tau_hat):
    # Create the best fit line on the cumulative data
    fit_n = total_decays*limited_lifetime_cdf(reading_intervals[1:], tau_hat)

    # Process data and fit line for declining plots
    decays_sliced = counts_per_day(decays_cumulative)
    errors_sliced = np.sqrt(decays_sliced)
    fit_sliced = counts_per_day(fit_n)

    # Plot of cumulative data set (our measured data)
    plt.figure()
    plt.errorbar(times, decays_cumulative, yerr=errors, ecolor="black",
                 fmt='b.', ms=5, mew=0.5, label="Simulated measurement data")
    plt.xlabel("time [d]")
    plt.ylabel("number of registered decays")
    plt.xlim(0, reading_intervals[-1] + 2)
    plt.ylim(0, total_decays+5000)
    plt.ticklabel_format(style='sci', axis='y', scilimits=(0, 0))
```

```python
    plt.legend()
    plt.savefig("decay_measurement_cumulative.pdf")

    # Plot of declining data set with fitted line
    plt.figure()
    plt.plot(reading_intervals[1:], fit_sliced, color='orange', label="fit line", zorder=1)
    plt.errorbar(reading_intervals[1:], decays_sliced, yerr=errors_sliced, fmt='b.', linestyle="None",
                 label="Simulated measurement data", zorder=2)
    plt.xlabel("time [d]")
    plt.ylabel("number of registered decays per day")
    plt.ticklabel_format(style='sci', axis='y', scilimits=(0, 0))
    plt.legend()
    plt.savefig("decay_measurement_declining_fitted.pdf")


def pull_calc(repetitions):
    pull = []

    for i in range(repetitions):
        # Generated simulated measurement data again
        decays_cumulative, errors = data_simulation(true_tau, total_decays,
                                                    reading_intervals)

        # Estimate the mean lifetime again
        tau_hat, sig_tau_hat = tau_estimator(decays_cumulative, reading_intervals)

        # calculating the pull again
        sig_tau_hat_average = max(sig_tau_hat)
        pull.append((tau_hat - true_tau)/sig_tau_hat_average)

    plt.figure()
    plt.hist(pull, bins=30, edgecolor="xkcd:darkblue", range=(-3,3))
    plt.xlabel("pull")
    plt.ylabel("counts")
    plt.xlim(-3,3)
    plt.savefig('pull.pdf')

    return pull


def task_3():
    print("Actual mean lifetime:    {:.1f}d".format(true_tau))

    # Generated simulated measurement data
```

```
    decays_cumulative, errors = data_simulation(true_tau, total_decays,
                                                reading_intervals)

    # Estimate the mean lifetime
    tau_hat, sig_tau_hat = tau_estimator(decays_cumulative, reading_intervals)

    print("Estimated mean lifetime: ({:.1f} + {:.1f} / - {:.1f})d".format(
          tau_hat, sig_tau_hat[0], sig_tau_hat[1]))

    # Create a plot with a maximum likelihood fit
    fitted_plot(reading_intervals, decays_cumulative, errors, times, tau_hat)

    # calculating the pull
    pull = pull_calc(1000)
#     print("Pull of this meausurement:", pull[0])

    print("mean of the pull:", np.mean(pull))
    print("standard deviation of the pull:", np.sqrt(np.cov(pull)))
#     print(pull)


if __name__ == "__main__":
    # Task 3: Measurement of mean lifetime
    task_3()
```

A.2 Determination of the total activity

```
# -*- coding: utf-8 -*-
"""
Data Analysis, Fall Semester 2018

Group Project 1: Characterization of a radioactive source, Task 4

Authors: Stefanie Jucker, Andrej Maraffio, Mirko Mirosavljevic, Manuela Rohrbach, Stefan von Rohr
"""
import matplotlib.pyplot as plt
from numpy.random import multivariate_normal
import numpy as np
from scipy.stats import uniform, poisson
from Task_3 import tau_estimator, data_simulation

# Properties of the experiment
true_half_life = 73.81  # days
true_tau = true_half_life/np.log(2)  # days
total_decays = 113809
```

```python
reading_intervals = np.array([i for i in np.arange(0, 61)])  # days

#calculated in Task_3:
tau = 107.8  # days
tau_err = 2.0 # days
half_life = tau*np.log(2) # days
half_life_err = tau_err*np.log(2) # days

# Mean values for 2D Gaussian distribution
mu_x = 0  # cm
mu_y = 0  # cm

# Standard deviations for 2D Gaussian distribution
std_x = 3  # cm
std_y = 6  # cm

# Covariance matrix of uncorrelated x and y
cov_xy = np.array([[std_x**2, 0],
                   [0, std_y**2]])

# Position and dimensions of the detector
detector_width_x = 4  # cm
detector_width_y = 5  # cm
detector_center = (0, 0)


def detector_acceptance(mus, cov, width_x, width_y, center, number_of_points,
                        number_of_repetitions, plot=False, systematic_position_error=False):
    """
    Determine the geometric acceptance of the detector
    with a systematic uncertainty on the detector position if systematic_position_error=True
    """
    # Define the borders of the detector
    x_low = center[0] - 0.5*width_x
    x_high = center[0] + 0.5*width_x
    y_low = center[1] - 0.5*width_y
    y_high = center[1] + 0.5*width_y

    # Repetition of experiment multiple times to get a mean and std
    acceptances = []
    for i in range(number_of_repetitions):

        # If systematic error on position, generate uniform uncertainty on borders of detector
        if systematic_position_error:
            x_error = uniform.rvs(loc=-0.2, scale=0.4) # cm
```

```
            y_error = uniform.rvs(loc=-0.2, scale=0.4) # cm
            x_low += x_error
            x_high += x_error
            y_low += y_error
            y_high += y_error

        # Generate position of particles
        points = multivariate_normal(mus, cov, size=number_of_points)

        # Determine which points will hit the detector
        detected_points = 0
        for x, y in points:
            if (x_low <= x <= x_high) and (y_low <= y <= y_high):
                detected_points += 1

        acceptances.append(detected_points/number_of_points)

    acceptance = np.mean(acceptances)
    error = np.std(acceptances, ddof=1)

    if plot:
        heatmap(mus, cov, x_low, x_high, y_low, y_high)

    return acceptance, error


def heatmap(mus, cov, x_low, x_high, y_low, y_high, size=1000000):
    points = multivariate_normal(mus, cov, size=size)

    plt.figure()
    plt.ylabel("x position [cm]")
    plt.xlabel("y position [cm]")

#       # Plot the scattered points
#       plt.plot(points[:, 0], points[:, 1], 'b.', ms=2)

    # Make a heat map of the scattered points
    n_std = 2.5
    x_lim = [mus[0] - n_std*np.sqrt(cov[0, 0]),
             mus[0] + n_std*np.sqrt(cov[0, 0])]
    y_lim = [mus[1] - n_std*np.sqrt(cov[1, 1]),
             mus[1] + n_std*np.sqrt(cov[1, 1])]
    hist_values, x_edges, y_edges = np.histogram2d(points[:, 1],
                                                   points[:, 0],
                                                   bins=(100, 50),
                                                   range=[y_lim,
                                                      ↪ x_lim])
    extent = [x_edges[0], x_edges[-1], y_edges[0], y_edges[-1]]
    plt.imshow(hist_values.T, extent=extent, origin='lower')
```

```python
    plt.colorbar()

    # Plot the boundaries of the detector
    plt.plot(2*[y_low], [x_low, x_high], 'r-', linewidth=0.6)
    plt.plot(2*[y_high], [x_low, x_high], 'r-', linewidth=0.6)
    plt.plot([y_low, y_high], 2*[x_low], 'r-', linewidth=0.6)
    plt.plot([y_low, y_high], 2*[x_high], 'r-', linewidth=0.6)

    plt.savefig("Electron_scatter_heatmap.pdf")


def detector_activity(acceptance, tau=tau, total_decays=total_decays,
                      t_activity=1, t_end=60):
    """ Compute the activity of the source """
    N_0 = total_decays/(1-np.exp(-t_end/tau))
    C = N_0/(tau*24*60*60*acceptance)
    act = C*np.exp(-t_activity/tau)

    return act

def tau_total_decays_cov(true_tau, reading_intervals, reps):
    """ Finds the covariance of the total number of events and the mean life time"""
    # Generate total number of events poisson distributed
    tot_decay = poisson.rvs(total_decays, size=reps)

    # Determine mean life time for each of the total number of events
    tau_est = []
    for i in range(reps):
        data = data_simulation(true_tau, tot_decay[i], reading_intervals)[0]
        tau_est.append(tau_estimator(data, reading_intervals)[0])
    tau_est = np.array(tau_est)

    # Calculate the covariance of the mean life time and the total number of evenets
    return np.cov(tau_est, tot_decay)

def activity_error(acc, acc_error, total_decays, tau_hat, tau_error, cov_tau_N, no_days=60):
    """Calculates the error on the activity of the source"""
    a = acc
    t = no_days*24*60*60 # seconds
    T = tau_hat
    n = total_decays
    derivative_acc = n/((a**2)*T - (a**2)*np.exp(t/T)*T)
    derivative_tau = (n*(np.exp(t/T)*(t - T) + T))/(a*((-1 + np.exp(t/T))**2)*(T**3))
    derivative_total_decays = -(1/(a*T - a*np.exp(t/T)*T))
```

```python
    jacobian = np.array([derivative_acc, derivative_tau, derivative_total_decays])
    cov_matrix = np.array([[acc_error**2, 0, 0],
                           [0, tau_error**2, cov_tau_N],
                           [0, cov_tau_N, total_decays]])
    return np.sqrt(np.dot(jacobian, np.dot(cov_matrix, jacobian.T)))

def activity_error_uncorrelated(acc, acc_error, total_decays, tau_hat, tau_error, no_days=60):
    """Calculates the error on the activity of the source"""
    a = acc
    t = no_days*24*60*60 # seconds
    T = tau_hat
    n = total_decays
    derivative_acc = n/((a**2)*T - (a**2)*np.exp(t/T)*T)
    derivative_tau = (n*(np.exp(t/T)*(t - T) + T))/(a*((-1 + np.exp(t/T))**2)*(T**3))
    derivative_total_decays = -(1/(a*T - a*np.exp(t/T)*T))
    jacobian = np.array([derivative_acc, derivative_tau, derivative_total_decays])
    cov_matrix = np.array([[acc_error**2, 0, 0],
                           [0, tau_error**2, 0],
                           [0, 0, total_decays]])
    return np.sqrt(np.dot(jacobian, np.dot(cov_matrix, jacobian.T)))

def task_4(ON):
    """ Compute task 4 without generating plots if ON=False"""
    # Calculated in Task_3:
    tau_s = tau*24*60*60  # seconds
    tau_err_s = tau_err*24*60*60 # seconds

    # Compute the acceptance of our detector and the
    acc, acc_err = detector_acceptance([mu_x, mu_y], cov_xy, detector_width_x,
                                       detector_width_y, detector_center,
                                       10000, 1000, plot=ON)
    print("The geometric acceptance of the detector is {:.3f} +/- {:.3f}".format(
          acc, acc_err))

    activity = detector_activity(acc)

    repetitions = 1000

    cov_tau_N_mat = tau_total_decays_cov(true_tau, reading_intervals, repetitions) #days*number
    cov_tau_N = cov_tau_N_mat[0][1]*24*60*60 # seconds

```

```python
    activity_err = activity_error(acc, acc_err, total_decays, tau_s, tau_err_s, cov_tau_N)

    print(f"The source's activity for t = 1d is {activity:.3} Bq")
    print(f'Error on activity: {activity_err} for {repetitions} repetitions')

    #Uncorrelated activity error
    uncor_act_err = activity_error_uncorrelated(acc, acc_err, total_decays, tau_s, tau_err_s)
    print(f'Gaussian error propagation: {uncor_act_err}')

    # Compute the acceptance of the detector
    acc_syst, acc_syst_err = detector_acceptance([mu_x, mu_y], cov_xy, detector_width_x,
                                                  detector_width_y, detector_center,
                                                  10000, 1000, systematic_position_error=True)
    print("The geometric acceptance with a systematic error for the position of the detector is {:.3f} +/- {:.3f}".format(
            acc_syst, acc_syst_err))

    activity_syst = detector_activity(acc_syst)

    print("The source's activity for t = 1d is {:.3} Bq".format(activity_syst))
    # Same for acc_with_systematic

    activity_syst_err = activity_error(acc_syst, acc_syst_err, total_decays, tau_s, tau_err_s, cov_tau_N)
    print(f'Error on activity with systematic detector missalignement: {activity_syst_err} for {repetitions} repetitions')

if __name__ == "__main__":
    ON = True
    task_4(ON)
```

A.3 Histograms of many repetitions

```python
total_decays = 113809
reading_intervals = np.array([i for i in np.arange(0, 61)])  # days
true_half_life = 73.81  # days
true_tau = true_half_life/np.log(2)  # days
# Get the middle of the time bins
times = 0.5*(reading_intervals[1:] + reading_intervals[:-1])


def read_file(filename):
```

```
    """
    Read data from a csv file to an array. Each row in the file
      ↪ becomes a row
    in the array
    """
    # Open the file
    with open(filename, 'r') as file:
        data = []
        # Define a csv reader
        rows = csv.reader(file, delimiter=',')
        for row in rows:
            # Convert the entries from string to float
            row_new = [float(x) for x in row]
            data.append(row_new)

        data = np.array(data)

        return data


def write_data_to_file(data, filename):
    """
    Write data from an arrow to a csv file. Each row of the array is
      ↪ written
    as a row in the file.
    """
    # Open the file
    with open(filename, 'w') as file:
        # Define a csv writer
        writer = csv.writer(file, delimiter=',')
        # Write to file
        for row in data:
            writer.writerow(row)


def repeated_simulation_file(repetitions, filename):
    """
    Repeat the experimental data simulation multiple times and save
      ↪ the
    data to a csv file
    """
    # Generate the simulated data sets
    data = []
    for i in range(repetitions):
        decays, errors = simulate(true_tau, total_decays,
          ↪ reading_intervals)
        data.append(decays)

    # Write the data to the file
```

```python
    write_data_to_file(data, filename)


def many_taus(filename, tau_filename='taus.csv', save=False, accuracy=0.1):
    """
    Read the data contained in the file corresponding to simulated measurement
    data sets and estimate the mean life time for each data set.
    Save output to a file
    """
    # Read cumulative decay data from file
    data = read_file(filename)
    taus = []
    errors = []

    # Compute the mean life time for each data set
    for data_set in data:
        tau, error = estimate(data_set, reading_intervals)
        taus.append(tau)
        errors.append(error)

    # Save the list of computed taus and corresponding errors to a file
    # Each row is: tau, lower_error, upper_error
    if save:
        n = len(taus)
        zipped = [[taus[i], errors[i][0], errors[i][1]] for i in range(n)]
        write_data_to_file(zipped, tau_filename)

    return taus


def tau_distribution(filename='taus.csv'):
    # Read the mean life times from the file
    file_contents = read_file(filename)
    taus = file_contents[:, 0]

    plt.figure()
    plt.xlabel(r"$\tau$ [d]")
    plt.ylabel("Counts")
    hist_values, bins, patches = plt.hist(taus, bins=20,
                                          edgecolor="xkcd:darkblue")

    plt.savefig("tau_distribution.pdf")

    tau_mean = np.mean(taus)
    tau_std = np.std(taus, ddof=1)
```

```
    print("Estimate for the mean lifetime tau_hat = ({:.1f} +/- {:.1f})d".format(
            tau_mean, tau_std))

    return tau_mean, tau_std
```