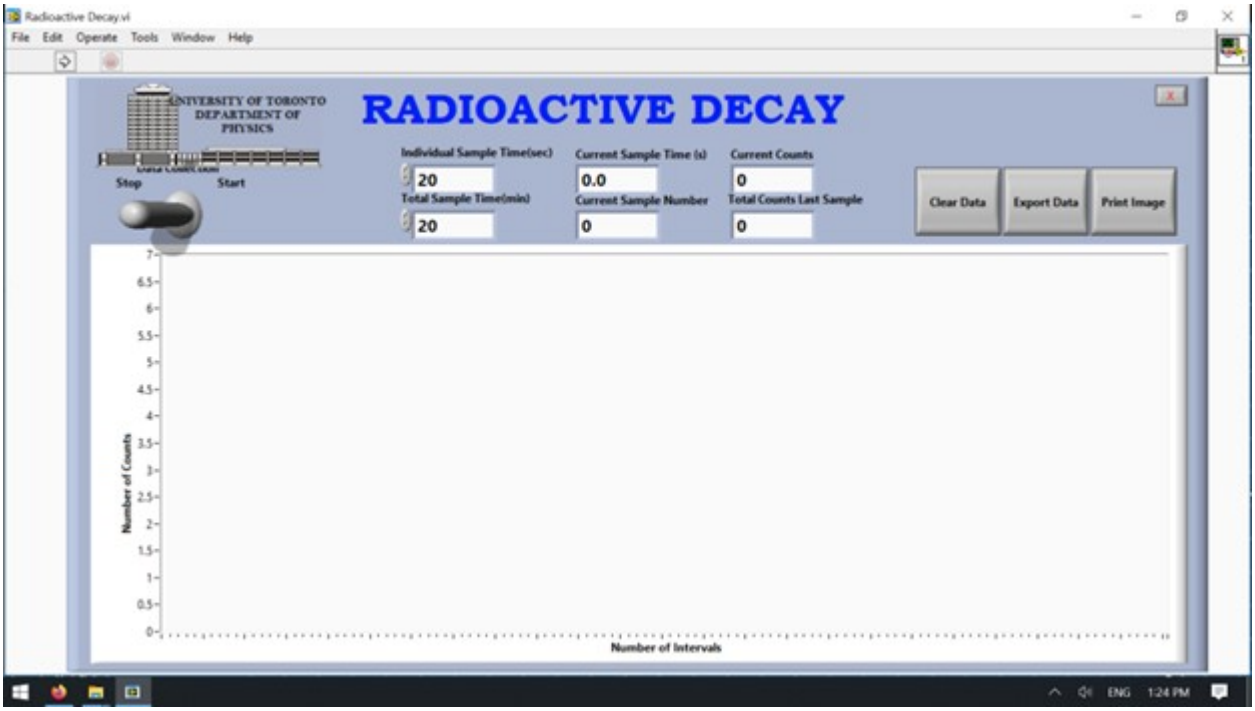
Physics
UNIVERSITY OF TORONTO

Radioactive Decay



Revisions

2022 E. Horsley, T. Vahabi, A. Harlick
2017 C. Lee
2016 R. M. Serbanescu and J. Ladan

Part I: Nonlinear fitting methods

Radiation and nonlinear circuits
We continue with curve fitting, focusing on non-linear models. Radioactive decay is an example of an exponential relation. We will measure the intensity from a radioactive isotope, and use curve fitting tools to find the half-life. Just like in the previous exercise, there are questions along the way that are intended to serve as a prompts for analysis.

Background knowledge for Part I

- **Python**: lists, arrays, numpy, SciPy, PyPlot, `curve_fit()`
- **Error Analysis**: Chi squared (χ^2), goodness of the fit
- **Physics**: Nuclear decay. You may need to review appropriate theory.

Introduction

You have probably learned about exponential decay of radioactive materials in the past. The number of emissions (during a sampling period) depends on the number of atoms of that isotope. Because an emission event corresponds to a death of an isotope, it follows that the intensity of radiation decays exponentially according to the equation,

$$I(t) = I_0 \exp\left(-\frac{t}{\tau}\right), \tag{1}$$

where $I(t)$ is the intensity of radiation at time t, I_0 is the intensity at $t = 0\ s$, and τ is the mean lifetime of the isotope. A more intuitive parameter is the half-life, $t_{1/2}$, defined so that

$$I(t) = I_0 \frac{1}{2}^{\frac{t}{t_{1/2}}} \tag{2}$$

Our interest for this experiment is calculating the half-life of an isotope by measuring the intensity (emissions/sample) of radiation.

The sample you will measure today is a meta-stable Barium ($Ba-137m$). It is prepared by flowing an acid through a generator containing Cesium ($Cs-137$). Inside the generator, the $Cs-137$ beta decays (with a half-life of 30.1 years), into $Ba-137m$. The acid washes out the $Ba-137m$ which decays by gamma emission (662 keV) with a half-life of 2.6 minutes. We will measure the rate of emissions using a Geiger counter.

Analysis with logarithms

Nonlinear relations become more complicated to analyze in general. There are analytical formulae to find the parameters in a linear regression, but typically not for a nonlinear

regression.

One method for nonlinear regression is transforming the data into a linear model and using linear regression methods. Logarithms are a particularly useful tool for analyzing exponential relations. Consider the general exponential relation,

$$y = y_0 e^{ax} \tag{3}$$

Taking the natural logarithm of both sides,

$$ln(y) = ax + ln(y_0). \tag{4}$$

Thus the exponential relation for y became a linear relation for $ln(y)$. Using a semi-log plot, with a logarithmic scale for the y-axis turns an exponential into a straight line, where the slope of the line is the growth/decay rate.
Note: in Python `math.log(numeric expression` returns the natural logarithm, $ln(numeric\ expression)$, of the value, i.e. `math.log(2)=0.69147...`. In order to return any other logarithm, base value needs to be provided, i.e. `math.logn(numeric expression` returns the logarithmic value of a numeric expression of base n, i.e. `math.log2(2)=1`.

By first transforming the relation, linear regression can be used to find the parameter a. The model function is the same, $f(x) = ax + b$, but now, the data to match is $z_i = ln(y_i)$. The transformation of the data also requires an uncertainty propagation to get more accurate results from the regression,

$$u(z_i) = \sqrt{\left(\frac{\partial z_i}{\partial y_i} u(y_i)\right)^2} = \left|\frac{u(y_i)}{y_i}\right| \tag{5}$$

Where $u(y_i)$ is the uncertainty of the value y_i. This propagation formula is the standard one for natural logarithms.

Note that when you are plotting the predictions from your model, you will need to convert the $ln(y)$ back into y so that it is in the correct units and can be compared with the experimental data.

Nonlinear regression

Traditionally, the method described in previous section was used for analyzing exponential relations, because a formula could be used to calculate the parameters. However, there is a problem with performing a linear regression on $z_i = ax_i$.

The maximum likelihood of a χ^2 linear model assumes that the *measurement errors* are normally distributed. When the axes are re-scaled, this assumption is *potentially* invalidated. A related side-effect is that certain ranges of y are given more weight in the regression.

With the use of computers and nonlinear optimization, it is now possible to perform a least-squares optimization using a wide range of nonlinear models. Recall that `curve_fit()` uses least squares to minimize

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - y(x_i)}{u(y_i)} \right)^2, \tag{6}$$

where y_i is the *dependent* data you measured, x_i is the *independent* data, $u(y_i)$ is the uncertainty measurement in the *dependent* data. For an exponential model, we simply use $y(x) = a \exp(bx)$ as the model function.

Computer-aided nonlinear regression brings with it other things to consider like whether the model function is suitable, if the initial guess for parameters is close enough to the true optimum values, and still if the measurement errors fit a normal distribution.

The Experiment

The radioactive sample used in this experiment will be prepared for you by a lab technologist, as a demonstration.

You will not acquire the data on your computer. Instead, the lab demonstrator will collect the data at the front of the lab and upload the data to Quercus. The instrument used to acquire the data is called a Geiger counter. You will also be provided with a text file with data for background radiation.

The radioactive energy is quite low and safe. We are using a single demonstration Geiger counter because of the complexity of setting up 30 identical experiments before the sample has completely decayed. You still need to document the experiment in the notebook (physical or digital) as if you were conducting it.

Uncertainty in the Geiger counter

The Geiger counter follows *counting statistics* for determining the uncertainty. This is much like counting the number of heads in a series of coin-flip trials the (the $\sqrt{N}$ rule). The rate R is the number of events N divided by the sample time Δt. The best estimate of the count rate is given by,

$$R = \frac{N}{\Delta t} \pm \frac{u(N)}{\Delta t} \tag{7}$$

There is also background radiation to consider (N_b); you should have heard the counter beeping with no sample present. The counts from the two sources of radiation add together for our final reading,

$$N_s = N_T - N_b \tag{8}$$

The uncertainty can be calculated via standard error propagation, so

$$u(N_s) = \sqrt{u(N_T)^2 + U(N_b)^2} = \sqrt{N_T + N_b} \tag{9}$$

where $u(N_i)$ are appropriate uncertainties in counts.

Finally, because the background radiation cannot be measured at the same time, we assume it has constant statistics, and use its mean in the calculations ($N_b = \bar{N}_b$) . The mean background radiation must be subtracted from each data point before analysis.

The Python program

We will use our programs to compare the transformation method and the non-linear least-squares method. The best parameters will be found with both methods, and used to plot best fit curves over the data. For this experiment, we have the luxury of comparing our calculated half-time to the theoretical value.

The program should be organized as follows:

- Import the required functions and modules (`numpy`, `scipy.optimize.curve_fit()`, `matplotlib.pyplot`).
- Define the model functions ($f(x, a, b) = ax + b, g(x, a, b) = b \exp(ax)$)
- Load the data using `loadtxt()`.
- Subtract the mean background radiation from the data.
- Calculate the uncertainty for each data point.
- Convert the count data into rates (divide by Δt).
- Perform the linear regression on $(x_i, log(y_i))$ using f as the model function.
- Perform the nonlinear regression on (x_i, y_i) using g as the model function.
- Calculate and output half-life values for each regression method.
- Plot the error bars, both curves of best fit (linear and non-linear), and the theoretical curve. Include a legend for the different curves.
- Make a second plot of the same data, but using a logarithmic y axis. One way of doing this is to call the `pylab.semilogy` function. Another way is to call `pylab.yscale('log')` after you make the plot.

An alternate form of the non–linear g function can be used where the *exp* is replaced by $\frac{1}{2}$ or 2. These transforms might change the way you process your data for input into the function, or change the way you interpret the errors from the `curve_fit`.

Write the program, and run it using the data gathered in the experiment. Save all plots and the parameters you determined.

Q1 Which regression method gave a half-life closer to the expected half-life of 2.6 minutes?

Q2 Are there any noticeable differences in the plots?

Analyzing the quality of the fit

There are two ways that we can assess the quality of the fit of our model: variance of the calculated parameters and the reduced chi-squared statistic.

Variance of parameters

The variance of the parameters is returned by `curve_fit()` as the diagonal entries in the covariance matrix, `p_cov`. Recall that the error in measurements is understood as the standard deviation,

$$t_{1/2} = \bar{t}_{1/2} \pm u(t_{1/2}) \tag{10}$$

where the uncertainty in half-life time $u(t_{1/2}) = \sqrt{Var(t_{1/2})}$ The variance of the parameter a is `p_cov[0, 0]`.

To get $\sigma_{t_{1/2}}$ from σ_a, you will need to use the uncertainty propagation formula for a reciprocal,

$$u\left(y = \frac{1}{x}\right) = \frac{u(x)}{x^2} \tag{11}$$

Modify your program from calculate the uncertainty of the half-life.

Q3 What values did you find? Does the nominal half-life (2.6 minutes) fall in the range?

Reduced χ^2

Recall that the χ^2 distribution gives a measure of how unlikely a measurement is. The more a model deviates from the measurements, the higher the value of χ^2. But if χ^2 is too small, it's also an indication of a problem: usually that there were not enough samples.

This best (most likely) value of χ^2 depends on the number of degrees of freedom of the data, $v = N - n$, where N is the number of observations and n is the number of parameters. This dependence is removed with the reduced *chi-squared* distribution,

$$\chi^2_{red} = \frac{1}{v} \sum_{i=1}^{N} \left(\frac{y_i - y(x_i)}{u(y_i)} \right)^2 \tag{12}$$

where $u(y_i)$ is the uncertainty in the dependent variable.

For χ^2_{red}, the best value is 1:

- $\chi^2_{red} \gg 1$ indicates that the model is a poor fit;
- $\chi^2_{red} > 1$ indicates an incomplete fit or underestimated error variance;
- $\chi^2_{red} < 1$ indicates an over-fit model (not enough data, or the model somehow is fitting the noise).

Q4 Add a function to your program to calculate χ^2_{red}. What values were computed? How would you interpret your results?

Part II: Random number analysis

Random events like radioactive decays obey the rare event statistics given by the Poisson distribution (see "Notes on Error Analysis" found at https://www.physics.utoronto.ca/~phy224_324/web-pages/Notes_Error.pdf:

$$P_{n,\Delta t} = P_{\mu}(n) \tag{13}$$

where n stands for number of counts in any Δt and the Poisson *probability mass function* or *pmf*, $P_{\mu}(n)$, is given by:

$$P_{\mu}(n) = e^{-\mu}\frac{\mu^n}{\Gamma(n+1)} \tag{14}$$

where μ is the expected average number of counts per counting interval Δt and $\Gamma(n+1)$ is the Gamma function.

Note that this function is the mass function because the we're using discrete random variables (counts) and the *pmf* gives the probability that a discrete random variable is exactly equal to a certain function. For continuous variables, we use the probability density function *(pdf)*. The integral of a *pdf* over a range of possible values (a, b) gives the probability that the continuous random variable falls within that range.

Background knowledge for part II

- **Python**: lists, arrays, numpy, SciPy, PyPlot, `curve fit()`, matplotlib.
- **Error Analysis**: Poisson Statistics.

Introduction

Let's look at an example: A radioactive source emits alpha particles at a rate of three particles per minute. Assuming a counting time of two minutes, the expected average number of counts will be the average rate of emission (three per minute) multipled by two minutes: $\mu = 6$. The probability of observing any number (n) of particles in the counting interval is given by:

$$P_6(n) = e^{-6}\frac{6^n}{n!} \tag{15}$$

where e is Euler's number.

In particular, the probability of observing exactly four particles in two minutes is given by:

$$P_6(4) = e^{-6}\frac{6^4}{4!} = 0.13 = 13\% \tag{16}$$

This means that the event happens once in about eight trials.

Analysis of random data

Radioactive decay counts are commonly analyzed by displaying data in a bar-chart or histogram format, as seen in Figure 1. The bars correspond to count bins and the height of each bar is proportional to the count in the corresponding bin. A histogram module is available in numpy. syntax can be found at: http://docs.scipy.org/doc/numpy/reference/generated/numpy.histogram.html. The histogram function does not t plot the data, but allows you to change many configuration options in the histogram calculation. The corresponding matplotlib.hist https://matplotlib.org/devdocs/api/_as_gen/matplotlib.pyplot.hist.html function does plot the histogram, but doesn't provide access to as many configuration options.

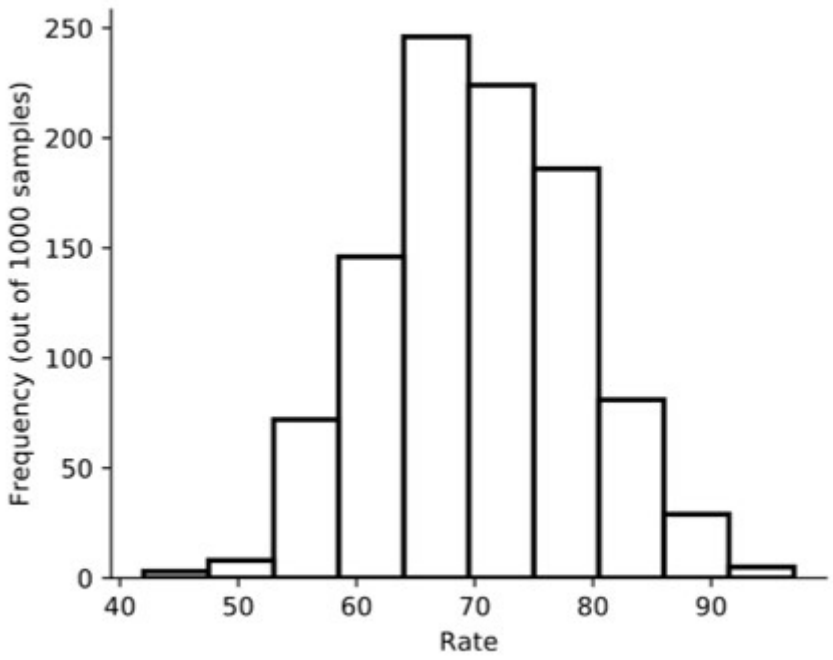


Figure 1: A typical histogram distribution, showing relative frequency as a function of rate (in arbitrary units).

Some input parameters for histogram:

- `a` : array-like input data
- `bins` : an integer defining the number of equal-width bins in the given range (10 by default),
- `range` : (float, float) is a float, meaning the lower and upper range of the bins,
- `normed` : a bool, optional. If True, the result is the value of the probability density function at the bin. Note: deprecated in Numpy 1.6, replaced by *density*

Outputs:

- `hist` : array: the values of the histogram
- `bin edges` : array of dtype float. For bins = 10 in the input, there will be 10 probability density values in n, 11 bin edges in bins.

The experiment

You will analyze the radioactive count from a Fiesta plate. Dinner plates from the Fiesta collection made before 1970 have a low amount of Uranium Oxide in the glaze. For more information, check: http://www.orau.org/ptp/collection/consumer%20products/fiesta.htm

You will not acquire the data on your computer. Instead, the lab demonstrator will collect the data at the front of the lab and upload the data to Quercus. The instrument used to acquire the data is called a Geiger counter. You will also be provided with a text file with data for background radiation.
The background count can be found in another file on Quercus. The sample time for the background data might be different, and occurred after the Fiesta plate data was taken. You should not compare the data in two files directly, but instead calculate the *average* background radioactivity before subtracting it from the sample activity.

Uncertainty in the Geiger counter

The Geiger counter follows *counting statistics* for determining the uncertainty. This is much like counting the number of heads in a series of coin-flip trials the (the $\sqrt{N}$ rule). The rate is the number of events Ndivided by the sample time Δt. The best estimate of the count rate R is given by,

$$R = \frac{N}{\Delta t} \pm \frac{\sqrt{N}}{\Delta t} \tag{17}$$

There is also background radiation to consider; you should have heard the counter beeping with no sample present. The counts from the two sources of radiation add together for our final reading,

$$N_s = N_T - N_b \tag{18}$$

The uncertainty can be calculated via standard error propagation, so

$$\begin{aligned} u(N_s) &= \sqrt{u(N_T)^2 + u(N_b)^2} \\ &= \sqrt{N_T + N_b} \end{aligned}$$

Finally, because the background radiation cannot be measured at the same time, we assume it has constant statistics, and use its mean in the calculations ($N_b = \bar{N}_b$) . The mean background radiation must be subtracted from each data point before analysis.

Note: there are issues using decimal values with the Poisson distribution function which can be avoided by rounding the average background to a whole number and/or using count and not rate. It makes the plotting of the Poisson function much easier, but requires some explanation to be included in the report.

The Python program

The program should be organized as follows:

- Import the required functions and modules. You can find the Poisson mass function in `scipy.stats.poisson`, and the Gaussian function in `scipy.stats.norm`. You should also import `matplotlib.pyplot`, `numpy` if needed.
- Load the data using `loadtxt()`.
- Analyze the Fiesta plate by writing a program to output the histogram.
- Plot the histogram of the count (or rate, keeping in mind using the count data is easier due to the discrete nature with respect to the Poisson function) data. You should modify the histogram's appearance, and normalize as needed.
- Add the Poisson probability mass function to qualitatively fit the data. The most appropriate value for μ can by taking the average value of all of your count data. This is the *Maximum Likelihood Estimation* of the parameter.
- Add the Gaussian distribution to your plot. From the *Central Limit Theorem* the Poisson function should tend towards a Gaussian for large enough data sets (number of measurements). In this case the mean value of the Gaussian is μ and the standard deviation $\sigma = \sqrt{\mu}$. Plot a Gaussian with the mean value and standard deviation from the measurements and compare it to the Poisson distribution.
- **Q5** Are there enough data points for the Poisson and Gaussian functions look the same?

  Submit all the plots, histogram with normpdf.

Background data

Perform the same analysis on the background data alone. Plot the histogram of background count and plot the Gaussian and Poisson functions with appropriate values of μ and σ. Comment on any differences between this analysis and the analysis from the Fiesta plate.

Your submission for this exercise should include:

- All sets of data, clearly labeled, presented in an appropriate manner (tables/figures).
- All uncertainties, uncertainty budgets, sample uncertainty calculations
- Important information about the set up - keep in mind you do not need to submit the methodology, but you need to include enough details so it is clear what part of the experiment is being analyzed.
- Answers to all the questions, weaved throughout the report.
- All relevant plots - histograms, fits, theoretical curves, etc. - as required
- All obtained values of the half-life parameters.
- All other values determined from the fits.
- Plots of residuals
- The final version of your Python code (included separately as a .py file(s))
- Analysis and Discussion of your results.
- Conclusions - as a separate section, reiterating the findings, quoting determined results, and summarizing all the work (half-page maximum).