Introduction to Scientific Computing

Winter 2020

In this assignment we focus on using experimental data, calculating estimates of parameters from the data, and fitting the data with Python, numpy, and scipy. The example data used here is simulated for two experiments - The speed of the Saturn V rocket on its journey to the Moon, and the dropping of a feather on the moon to measure Lunar gravity.

1 Reading and plotting data

In an experiment to measure the speed of a Saturn V rocket on its way to the moon, the distance from the Earth was recorded every 4 hours and saved into the file `rocket.csv`. Read the file into Python and plot the data with uncertainties.

You should use `numpy.loadtxt` to load data from the file, and the plot should include errorbars, axes labels and a title. Do not join the data with a line. Finally make sure you save your figure with a reasonable filename.

2 Estimating the speed

The next step in finding the speed of the ship is to estimate the speed by taking the average value of distance/time. Using the data in the `rocket.csv` file, calculate the mean and standard error of speed from the data. You can ignore the uncertainties for this question.

Note that using mean and standard deviation in this way is useful for a quick estimate of your data, but it's not statistically robust because uncertainties are not considered.

3 Linear Regression

The distance vs. time equation is simple enough that we can analytically derive the best estimate of s_0 and u considering all data at once, given by the following equations:

$$\hat{u} = \frac{\sum_{i=1}^{n}(t_i - \bar{t})(d_i - \bar{d})}{\sum_{i=1}^{n}(t_i - \bar{t})^2}$$

and

$$\hat{s_0} = \bar{s} - \hat{u}\bar{t}.$$

where $\hat{s_0}$ is the best estimate of variable s_0, and $\bar{s_0}$ is the average value of the s_0 data (and similarly for u). Write a script to read the data from the `rocket.csv` file, and calculate the best estimates for u and s_0 using the above equations.

4 Plotting the prediction

With a prediction of the speed and starting position, we can add our prediction to the data plot. Write a function that takes time, initial position, and speed, and predicts the location of the rocket (i.e. write

$s = s_0 + ut$ as a Python function). Use this function to make a new plot with your measured data and a *predicted* position generated for each measurement time. You can use your estimated values $\hat{u}$ and $\hat{s_0}$, or the initial value of $\bar{u}$ (and 0 km for the distance).

Your plot should now have the collection of data points with uncertainties, and a straight line showing the predicted position. You will need to add a `legend` to your plot to identify the data and prediction clearly.

5 Characterizing the fit

Now that you have a set of parameters for the model function, you can determine how well the model fits the data. One way to characterize the model fit is to calculate the χ_r^2 metric, defined as

$$\chi_r^2 = \frac{1}{N-m}\sum_i^N \frac{(y_i - f(x_i))^2}{\sigma_i^2}.$$

χ_r^2 squared is made up of four parts

- $y_i - f(x_i)$ - The difference between the measured data y_i and the prediction with your model $f(x_i)$
- σ_i - The uncertainty on each measurement's dependent data (the errors on y).
- The sum of the squared-ratio of the previous two values. $\sum_i$
- $\frac{1}{N-m}$ Is the denominator that scales χ_r^2 to be around 1 for a good fit. In this term N is the number of data points, and m is the number of parameters in your model (2 in this case).

Write a function to calculate χ_r^2, and use it to calculate χ_r^2 for your best fitting parameters from the last exercise.

6 Curve fitting

The linear regression equations you used above do not account for uncertainties, and the algebra gets more complicated quickly as the model gets more complicated (e.g. when more terms are included). In addition, the regression is *linear* and requires a model that is linear in the unknown parameters ($ax + b$ is linear, ax^b is not).

Instead, we are going to fit the measured data using a scipy function called `curve_fit`. `curve_fit` uses a mathematical algorithm to fit (almost) any model to a dataset by making progressively better guesses at the right answer and internally scoring each guess to make sure it improves the model fit. `curve_fit` comes from the `scipy.optimize` library, and documentation can be found at https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html

Typically, `curve_fit` will be called with the following code.

```
from scipy.optimize import curve_fit
popt, pcov = curve_fit(model_function, xdata, ydata,
                                      sigma=yerrors, absolute_sigma=True,
                                      p0=first_guess_parameters)
pstd = np.sqrt(np.diag(pcov))
```

In this code the variables are:

- `model_function` is the model of the data being measured.
- `xdata` are the independent data (a numpy array).
- `ydata` are the dependent data (a numpy array).
- `yerrors` are the errors on the dependent data (a numpy array).
- `absolute_sigma=true` is needed to tell curve_fit to treat your errors as absolute, not relative, values

- `first_guess_parameters` are your first guess at each parameter in your model (a list, tuple, or numpy array)

`curve_fit` then returns two variables, `popt` is a list of the optimized (i.e. best estimate) parameters. `pcov` is the covariance matrix, whose diagonals are the variances of the optimized parameters. The square root of the diagonal values are the standard deviations of each parameter independent of all other parameters (these are the values you might use in error propagation). The `pstd` in the above code are the standard deviations.

Write the Python code to run `curve_fit` on the rocket dataset to find the best estimate of the starting position and speed.

- Print out the estimates and their associated uncertainties (standard deviations).
- Print out the χ_r^2 for the model with the new best estimate parameters.
- Plot the data with uncertainties, *and* plot the model using your *linear regression* parameters and the `curve_fit` parameters as separate lines.

7 Feather Drop Experiment

Now that they are on the moon, one of the astronauts makes a measurement of the Lunar gravity by dropping a feather from shoulder height, saving the data in `feather.csv`. This time the equation of motion includes acceleration due to gravity

$$s = s_0 + ut + \frac{1}{2}at^2$$

Write a program that will

1. Load the time, position, and uncertainty data from the file `feather.csv`
2. Predict the position of the feather above the Lunar surface as a function of time given the initial position s_0, initial speed u, and constant acceleration a.
3. Fit the model to the data to find the optimal (i.e. best) parameters.
4. Print out the parameters fit by your model, including their uncertainties and units.
5. Plot the data including errorbars, and a prediction made by your model function using the best parameters.

It is often helpful to provide `curve_fit` with an initial guess of the parameters. Assume that the astronaut is between 1.5m and 2.0m tall, and Lunar gravity is around 1/7th of Earth's gravity.

The code your write to answer this question will be a good starting point for the code your write for all experiments in this course that involve fitting a model to data to derive a parameter value.