Assignment #5: Curve fitting, optimizations, and finite difference methods (**total 10 points + 1 bonus point**), due by 11:59 pm Tuesday, 11 April 2023

This practical introduces the following:

- PART 1: (Non-linear) Least square method to fit curve models into data points.
- PART 2: Use of general curve fitting methods applied to experimental data.
- PART 3: Applying Taylor series in finite difference methods.

1 Non-linear least squares

Assume that we have some sequence of measurements at times t_i

$$y_i = y(t_i) \tag{1}$$

and some presumed model of the relationship

$$\tilde{y} = f(t; p) \tag{2}$$

expressed in terms of the independent variable and model parameters p. For example, if our model is a straight line as

$$y = ax + b \tag{3}$$

then there are two parameters which we can express as a vector

$$\vec{p} = \begin{bmatrix} a \\ b \end{bmatrix} \tag{4}$$

How can we find the parameters which provide the "best" fit to the data? This requires that we select some quantitative measures of the fit quality. Ideally, we would like the difference δ between the model and the data

$$\delta_i = y_i - \tilde{y}_i \tag{5}$$

to be equal to zero for each observation. This is not the case for most real-data fittings, therefore we might try to make all the differences as small as possible. This can be tricky because improving the fit for one point will often make it worse for others. A general measure of the distance is

$$\sum_i |\delta_i|^d \tag{6}$$

where for $d = 2$ we obtain a quantity that is usually called *chi-squared*

$$\chi^2 = \sum_i |y_i - \tilde{y}_i(p_1, \ldots, p_K)|^2 \tag{7}$$

The best fit is assumed to correspond to minima in χ^2 with respect to the model parameters

$$\frac{\partial \chi^2}{\partial p_k} = \sum_i 2|y_i - \tilde{y}_i(p_1, \ldots, p_K)| \frac{\partial \tilde{y}_i}{\partial p_k} \tag{8}$$

which in turn depends on how the model changes with respect to the parameters. Note that this will in general require that we solve K coupled non-linear equations.

1.1 Linearity of parameters and reduced chi-squared

A straight line $y = mx + b$ is linear in all model parameters (m, b). So is a quadratic function as $y = c_0 + c_1 x + c_2 x^2$ with parameters (c_0, c_1, c_2), but not an exponential as $y = D \exp(-d\, x)$ with parameters (D, d) because y has a nonlinear dependence on d. Non-linear fitting (aka "optimization") techniques as conducted in this assignment can be used for both linear and non-linear models.

The standard definition of χ^2 as in equation (7) gives values that scale with the number of points. In other words, if we double the number of data points then we might expect χ^2 to be about twice as large. It is convenient to introduce a quantity called "reduced" chi-squared

$$\chi^2_N = \frac{\chi^2}{N} \tag{9}$$

simply by dividing by the number of data points. Note that this does not change the location of minima or maxima, and thus has no effect on the fitting results. Strictly speaking, we would like to define χ^2 more generally by scaling according to the uncertainty in each observation as

$$\sum \left(\frac{\text{expected} - \text{observed}}{\text{uncertainty}} \right)^2 \tag{10}$$

For a "good" model we should expect that the difference between expected (model) and observed (data) values should be on the order of the uncertainty. This would produce reduced chi-squared values of order 1. If we do not have reliable estimates of the uncertainty then we can proceed by just setting it to some constant (i.e. 1). Note that this does not change the location of minima or maxima, and thus has no effect on the fitting results.

1.2 Implementation: least squares optimization

(a) Import all commonly used Python packages (`matplotlib`, `math`, `Numpy`, etc.) and define a linear model function as given below

```
def linear_model(x, param):
    '''for input slope and intercept return $y= m x + b$'''
    slope, intercept = param
    result = slope*x + intercept
    return result
```

Note that `param` is a `set` of two components and its input has to be formatted as such.

(b) Let's imagine you estimated a linear physical model with a slope and intercept given by 1.2 and 0.5, respectively. Input these values in the `linear_model` function as

```
slope0, intercept0 = 1.2, 0.5
y_model = linear_model( x, (slope0, intercept0))
```

(c) Read the data points contained in the file `data_points_assign5.txt` (it is a two column data file). Consider those as data points coming from experiments in which a control variable x (first column) was systematically varied and y is the measured quantity (second column).

(d) Plot your linear model as a full line together with the data points (as symbols). How good is your model already? Calculate χ^2_N using equations (9) and (10) to quantify how "good" your initial model is. Set χ^2_N as a function in your code which can receive three arguments: `(expected, observed, uncertainty=1)`. **(0.5 points)**

(e) Can we improve the parameters of your model? For that, you will scan a parameter phase-space of slopes and intercepts and calculate χ^2_N for each pair (m, b) being m the slope and b the intercept of the line. The range in which you will scan the parameter domain is given below

```
intercept_sequence = np.linspace(0.0, 3.0, 101)
slope_sequence = np.linspace(0.0, 3.0, 101)
```

Store all values of χ^2_N in a two-dimensional `numpy` array named as `rchi2` in which its row-indexes account for the intercepts whereas its column-indexes account for the slopes.

(f) Use the function `np.argmin` embedded in `numpy` package to locate the minimum of χ^2_N. The documentation of `np.argmin` can be found in `https://docs.scipy.org/doc/numpy/reference/generated/numpy.argmin.html`. Note that you will also need to use the function `np.unravel_index` as specified in the documentation to locate the indexes in the array that gives its minimum value. What is the value of χ^2_N you obtained through such optimization? How does the value of χ^2_N compare with the one you obtained in item (d)? What about the slope and intercept parameters? Do they differ from the initial estimation given in item (b)? **(0.5 points)**

(g) Plot χ^2_N as a function of `intercept_sequence` fixing the slope at the value which gives the minimum of χ^2_N. Similarly, plot, in a different figure, χ^2_N as a function of `slope_sequence` fixing the intercept at the value which gives the minimum of χ^2_N. Do you see a minimum in both plots? If yes, mark the minima with a scatter symbol of your choice. **(0.5 points)**

(h) Create a meshgrid using

```
xx, yy = np.meshgrid(slope_sequence, intercept_sequence)
```

and use `plt.contour` as specified in the manuals `https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.contour.html` to plot a colour diagram of slopes versus intercepts with colourful contour lines that indicate how large χ^2_N is. And add a scatter point that marks where the minimum of χ^2_N **(0.5 points)**

(i) **IMPORTANT:** Attach ALL `.ipynb` files you generated for this exercise. All codes need to be fully documented, i.e. add comments explaining all the procedures in your code. Moreover, include notebook markdown cells below the figures your code generated to explain the content of your figures. **(1.5 points)**.

2 Curve fitting using scipy.optimize package

The way we implemented least square minimization so far is a sort of "brute force" scheme in which we need to explicitly scan the entire parameter phase-space in order to fit our model to the data points. A more efficient implementation can be done using `scipy.optimize` software package which provides a wide variety of tools for non-linear optimization. Its manual can be found in `https://docs.scipy.org/doc/scipy/reference/optimize.html`. In order to use most of these tools, we need to provide a function which takes as input some set of parameters and returns as output some measure of fit quality (e.g. χ^2). In particular, check the numerical routine named `minimize` at `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html#scipy.optimize.minimize`.

(a) Write a new code in which you apply `scipy.optimize.minimize`, following Nelder-Mead algorithm (see `https://docs.scipy.org/doc/scipy/reference/optimize.minimize-neldermead.html`), to determine the optimized slope and intercept parameters that fit our linear model to the data points in `data_points_assign5.txt`.

(b) Plot the data points in `data_points_assign5.txt` together with the line model parameterized by the slope and intercept values you found with `scipy.optimize.minimize`. **(0.5 points)**

(c) In the file `data_points_exp_assign5.txt`, you will find data points of an exponential decay. Repeat items (a) and (b) above to find the optimized parameters that best fit an exponential decay model to the data points. Use `scipy.optimize.minimize` to optimize the parameters

(D, b) of an exponential model given by $y = D\exp(-b\,x)$ and make a plot as described in item (b). **(0.5 points)**

(d) Barium 137 is radioactive. Activity (in counts) of a sample of Barium 137 was measured as a function of time (in minutes), and the results are shown in file `decay.txt`. We expect that for radioactive decay,

$$N = N_0 \exp(-t/\tau) \tag{11}$$

where $\tau = t_{1/2}/\ln(2)$ with $t_{1/2}$ being the half-life of Barium 137. Find the half-life for Barium 137. Support your answer with a graph and a curve fit. Use `scipy.optimize.minimize` to optimize the parameters describing the exponential decay. **(0.5 points)**

(e) Another important routine in `scipy.optimize` for fitting is `curve_fit` documented in `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html`. In this item, you should implement this package in your code.

In the file `centripetal.txt`, you will find data showing the angular velocity, ω of a rotating "point mass" m and the force F_c required to keep the mass rotating in a circle of radius R. This is a 3-column file containing ω (first column), F_c (second column), and the estimated error for the force (third column). ω is in radians/second and the force in Newtons. If you plot this data, it looks linear (except for that one point at $(0, 0)$). Theoretically, it is expected that the equation for centripetal force should be

$$F_c = mR\omega^2 \tag{12}$$

- Does the quadratic model for F_c in equation (12) fit the data? Plot a graph to support your answer. Include the error for the force in your plot as error bars. **(0.5 points)**
- The rotating mass had $m = 200$ grams, and the radius of rotation was $R = 18$ cm. Is this consistent with parameters from your curve fit? **(0.5 points)**

(f) **IMPORTANT:** Attach ALL `.ipynb` files you generated for this exercise. All codes need to be fully documented, i.e. add comments explaining all the procedures in your code. Moreover, include notebook markdown cells below the figures your code generated to explain the content of your figures. **(1.5 points)**.

3 Taylor series and finite difference methods

A real-valued function $f(x)$ can be approximated with a Taylor series near some point $x = a$ as

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \ldots \tag{13}$$

If we restrict our attention to a set of points equally spaced by Δ, this becomes

$$f(x_{i+n}) = f(x_i) + f'(x_i)n\Delta + \frac{f''(x_i)}{2}(n\Delta)^2 + \frac{f'''(x_i)}{6}(n\Delta)^3 + O(\Delta^4) \tag{14}$$

Let's define $f_{i+n} \equiv f(x_{i+n})$. For three adjacent points, we can write

$$\begin{aligned} f_{i+1} &= f(x_i) + f'(x_i)\Delta + \frac{f''(x_i)}{2}\Delta^2 + \frac{f'''(x_i)}{6}\Delta^3 + O(\Delta^4) \\ f_i &= f(x_i) \\ f_{i-1} &= f(x_i) - f'(x_i)\Delta + \frac{f''(x_i)}{2}\Delta^2 - \frac{f'''(x_i)}{6}\Delta^3 + O(\Delta^4) \end{aligned} \tag{15}$$

We can subtract the expressions for two neighbouring points as

$$f_{i+1} - f_i = f'(x_i)\Delta + \frac{f''(x_i)}{2}\Delta^2 + \frac{f'''(x_i)}{6}\Delta^3 + O(\Delta^4) \tag{16}$$

To get a *forward difference* estimate of the first derivative, we write

$$\boxed{\frac{f_{i+1} - f_i}{\Delta} = f'(x_i) + \frac{f''(x_i)}{2}\Delta + \frac{f'''(x_i)}{6}\Delta^2 + O(\Delta^3) \approx f'(x_i) + O(\Delta)} \tag{17}$$

A similar approach gives a *backward difference* estimate of the slope as

$$\boxed{f'(x_i) \approx \frac{f_i - f_{i-1}}{\Delta} + O(\Delta)} \tag{18}$$

By subtracting f_{i-1} from f_{i+1} in expressions (15), we can cancel all of the even terms (i.e. f, f'') as

$$f_{i+1} - f_{i-1} = 2f'(x_i)\Delta + 2\frac{f'''(x_i)}{6}\Delta^3 + O(\Delta^5) \tag{19}$$

and re-arrange the terms to get an equation for the first *centered difference*

$$\boxed{f'(x_i) \approx \frac{f_{i+1} - f_{i-1}}{2\Delta} + O(\Delta^2)} \tag{20}$$

with an error that is quadratic in Δ.

By adding f_{i-1} to f_{i+1} in expressions (15), we can cancel all of the odd terms (i.e. f', f''') as

$$f_{i+1} + f_{i-1} = 2f(x_i) + f''(x_i)\Delta^2 + O(\Delta^2) \tag{21}$$

and obtain an expression for the second difference estimate

$$\boxed{f''(x_i) \approx \frac{f_{i+1} - 2f(x_i) + f_{i-1}}{\Delta^2} + O(\Delta^2)} \tag{22}$$

with an error that is also quadratic in Δ.

3.1 Electrostatics

As we learn in electrostatics, the electric field $\vec{E}$ is defined as the negative gradient of the electric potential V. Using this in the differential form of Gauss's law, we will get the Poisson's equation for the electric potential as

$$\nabla^2 V = -\frac{\rho}{\epsilon_0} \tag{23}$$

where ρ is a charge density and ϵ_0 is the vacuum permittivity. A frequent use of this equation is in determining the electric potential in an empty space where $\rho = 0$. In this case, Poisson's equation reduces to the Laplace's equation

$$\nabla^2 V = 0 \tag{24}$$

which in Cartesian coordinates is given by

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} + \frac{\partial^2 V}{\partial z^2} = 0 \tag{25}$$

Consider the problem first in 1-dimensional (1-d) with $V(x)$. Solving equation (24) analytically, we get the general solution $V(x) = ax + b$ where a and b can be determined from established boundary conditions. By discretizing the x-axis as $x_i = i\Delta x$ (i is an integer), we can approximate the first derivative of the potential with centred differences (cf. equation (20)) and the second derivative as in equation (22), resulting in the discretized form of the 1-d Laplace's equation as

$$\boxed{V_i = \frac{V_{i-1} + V_{i+1}}{2}} \tag{26}$$

In this scheme, the potential at any given location is equal to the average of the nearest neighbours. This suggests that we might be able to code an iterative process to solve for the potential numerically and this solution should converge to the analytical solution of $V(x) = ax + b$.

(a) Consider a region in space along the x-axis of 2 meter length. Discretize the space with a number of points $N = 100$. Initiate the potential values along the length at zero volts and establish fixed boundary conditions for the potential in the first and last elements of the array as $V_0 = 0$ V and $V_N = 5$ V. Use `numpy` arrays to create x and V arrays.

(b) Set a scheme in which one sweeps through the voltage array and replaces every interior (non-boundary) value with the average of the two neighbours using equation (26). This characterizes 1 iteration step that should bring the potential closer to its analytical solution of $V(x) = ax + b$. Try to code this step without any loops by using the *slicing* property of `numpy` arrays as detailed in `https://numpy.org/doc/stable/reference/arrays.indexing.html`. Other supplemental material about array slicing can be found in our Week 6-8 D2L module (Technical Demos).

(c) Place the scheme implemented in item (b) above into an iterative procedure that will lead your numerical solution for $V(x)$ to converge to the analytical solution of $V(x) = ax + b$ at each iteration step. Plot $V(x)$ versus x for the first iteration, the final iteration (converged), and a number of intermediate $V(x)$ versus x curves calculated during the iteration process. Note that besides the code, obtain analytically the coefficients a and b of the linear relation $V(x) = ax + b$ and so you can compare your numerical solution with the analytical one. **(0.5 points)**

(d) Based on what you obtained in 3.1.c), how far is your numerical solution from the analytical one? To answer this question, make a plot of the absolute error versus x calculated at the first iteration, the final iteration (converged), and at a number of intermediate iteration steps to see how the absolute error evolves as more iterations to solve the Laplace's equation are conducted. **(0.5 points)**

(e) **IMPORTANT:** Attach ALL `.ipynb` files you generated for this exercise. All codes need to be fully documented, i.e. add comments explaining all the procedures in your code. Moreover, include notebook markdown cells below the figures your code generated to explain the content of your figures. **(1.5 points)**.

(f) **BONUS (extending to 2-d):** Consider a point charge q located in the middle of a 2-d square box of dimensions $L \times L$. You can choose the values for q and L. For the boundary conditions, assume that the potential is zero along the edges of the box (i.e. the edges of the box are grounded). Solve the Poisson equation (23) numerically and plot the solution as a color contour map of $x \times y \times V$ where V values are represented by a color scale. Use `contourf` method embedded in `matplotlib` to build this mapping. Check the documentation at `https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.contourf.html` **(1 point)**.

4 Submission

The due date for the submission of your assignment is depicted on the first page of the assignment. **For this assignment, you only need to submit your Jupyter Notebook codes.** All your codes need to be fully documented, i.e. add comments explaining all the procedures in your code. Moreover, include notebook markdown cells below any figure your code generates to explain the content of the figure. All figures need to have proper axis labels and units (if dealing with physical quantities). Pure code lines without explanatory comments or without written markdown cells will have reduced marks.

This assignment is arranged in three parts: 1. Non-linear least squares, 2. Curve fitting, 3. Taylor series and finite difference methods. Name your notebook files in accordance with these parts, e.g. any code you wrote for part 1 name as `code_<index>_part_1.ipynb` in which `index` should refer to the exercise item the code is designed for, e.g., 12a, 12b, ..., 2a, 2b, ..., 31a, 31b, ... If you decided to include all codes for part 1 in the same Notebook file (but in different cells), name your

file as `codes_part_1.ipynb`. Apply the same principle to the other codes developed for parts 2 and 3. Upload all your `.ipynb` files AT ONCE in the Gradescope folder for this assignment. We will test-run all your submitted codes.

Please, remember to add comments in all your submitted Notebook codes! Pure code lines without explanatory comments or without written Markdown cells will have reduced marks. In addition, provide (brief) written explanations of the figures and outputs your codes generate using Markdown cells.

Standard `.py` Python codes are not accepted in this submission. Work only with Notebook environments. If any member of the group is experiencing problems with their coding environment, please, contact the instructor immediately.

*****