Overview

In electron spectroscopy matter is examined by radiating it with a bright light and measuring the kinetic energy of electrons that come off it. When the photonic energy of light and kinetic energy of the electron known, they can be used to derive the amount of force that was required to break off the electrons. This provides valuable information about the matter's electron structure, and its chemical and physical propert This phenomenon where photons break off electrons is called photoionization, and the broken off electro are called photoelectrons.

In this course project you'll learn how to read data into a program and how to perform small operations or data, and how to plot data using Python libraries. Your task is to write a program for analyzing the photoionization spectrum of argon. For this purpose we have provided you with simulated data where arg atoms have been ionized and the kinetic energy of broken off electrons has been measured.

About Libraries

This project requires you to use two third party libraries: numpy and matplotlib. There are multiple ways t install them. Windows installers can be found for both, and you can also install them using pip. There are various full stack installers available in the internet (full stack means they install Python along with a bunc libraries, usually replacing your "generic" Python installation). Please write a comment in your code if you any such full stack installer because they might contain different versions of the libraries than those availa generally.

You'll also need our small graphical user interface library - more about it below.

Specific Requirements

The measurement has been performed multiple times, and each measurement session has been recorded a different, numbered file. The file names are in the format `measurement_i.txt`. Each file contains rows o data with two floating point numbers. The first number on each line is the binding energy of electrons, de from the measured kinetic energy (unit: electronvolt); the second number is the corresponding intensity (r specific unit; this described the amount of electrons measured with this particular binding energy). In each measurement file, the first column contains the same uniformly distributed binding energy values. Your program should add together the intensity values from each file. The purpose is to eliminate noise from th measurements.

spektri.zip

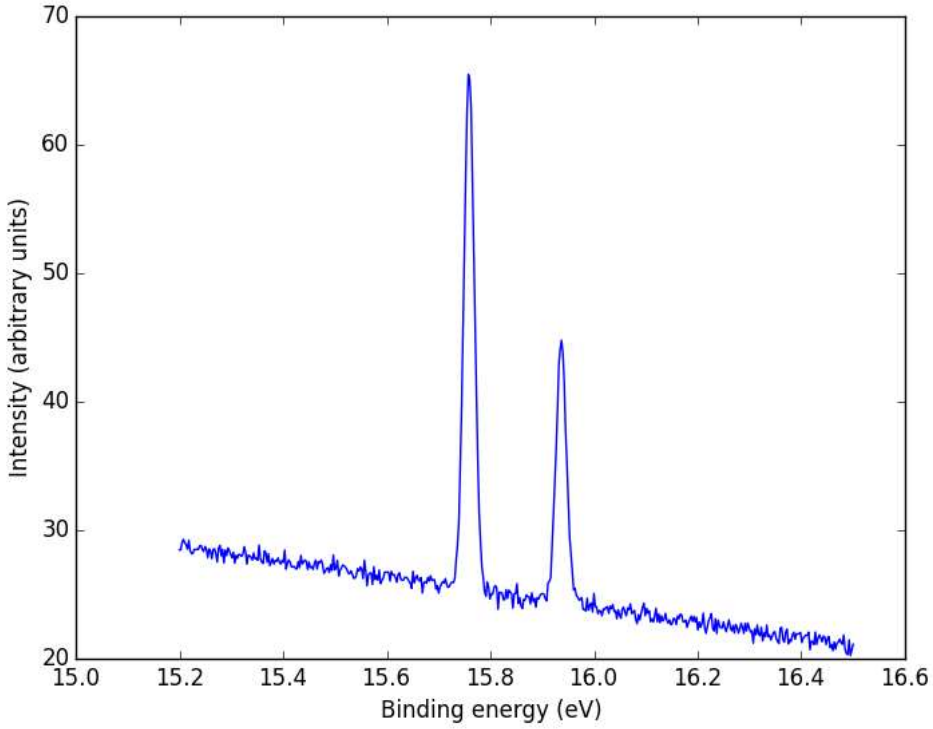
Due to the measuring equipment, the spectrum has a linear background. Aside from the obvious peaks it looks like a downward sloping line. The background signal that causes the sloping should be removed befo analyzing the spectrum. This can be done by choosing two points from the spectrum and fitting a line between these points. After this, at each data point, values obtained from this line are subtracted from the measured intensity values.

When analyzing the spectrum our primary interest are the two rather obvious peaks in intensity; in partic their relative intensity. The intensity of each peak is obtained by computing their area by obtaining its inte This can obtained by using the trapezoidal rule to estimate the integral. According to theory the first peak should have approximately double the intensity of the second one.

Your program needs to have the following features:

1. Your program has a graphical user interface with all the features available for the user.
2. Load data: loads data from a user-specified location and reads it into program memory in a format tha suitable for processing. Should return one list for each column in the data. The first list should contain measured kinetic energy values (measurement points) and the second one the sums of all measureme for each row.
3. Plot data: this plots the current data (the user is prompted to load the data first if it hasn't been loade yet). You can plot the data using matplotlib, but it must be plotted inside the application window - do use pyplot. The figure should look like the one below.
4. Remove linear background: removes the linear background from the data as described above. The use selects two points from the figure, and a line is fitted between these points. The line is then subtracte from the data. If there's no data in the program memory yet, the user is given an error message about
5. Calculate intensities: The intensity of peaks can be calculated. This is done with the trapz function fro numpy. The use selects the interval by clicking on the figure. The result is printed somewhere inside tl window. If there's no data in the program memory yet, the user is given an error message about it.
6. Save figure: this feature allows the user to save an image of the current plot. The user uses a separate dialog for select a filename and destination for saving the figure. matplotlib provides you with the necessary features to do this.

You should name the axes in your figure with appropriate names. There are tools for doing this in matplotl



Example plot of the data

Graphics

Deployment

We've provided a library that is built on top of TkInter, and offers a heavily simplified interface to some of features through functions. The library's docstrings describe how to use it. The lirary's main program also a short example of how to make a simple interface.

guilib.py

User Interface Libraries 101

In user interface programming the main loop is usually contained inside the library being used. All of the u actions are connected to handler functions. For instance, when the user presses a button, the handler function attached to that button is called, and it does Something. The program doesn't proceed as linearly we've used to. User interfaces consist of components or elements that are also often called widgets. Thes can be simple, like buttons, or more complex like an entire file opening dialog - one you can see in many programs.

Typically an interface is created by choosing what components to put in there. While doing so, their attrib and handler functions are also defined. Rest of the implementation comes down to creating these handler functions and bunch of utility functions (that are used by the handlers).

One special characteristic is that functions are called kind of externally from the user interface library, anc therefore we cannot control what arguments they are given. It's not possible to transmit the program's sta function arguments - another way to share it needs to be devised. A good choice in the context of this prc is to make a dictionary that has keys for things like loaded data and data points selected by the user. If the dictionary is defined in the main program scope it can be handled in all functions, thanks to its mutability. way any function can change the program state without having been given it as an argument.

Collaboration

We like collaboration on this course. These weekly nice little exercises are actually made in cooperation w several different assistants. It's also probably much more pleasant to solve them together. Your friend may have a better understanding about something, and two pairs of eyes spot bugs more easily anyway. In mo cases you realize your mistake when you start to complain to someone that your program doesn't work. Usually things work more smoothly in your head, compared to what it sounds like when you start to expla to someone else. That's why we don't by any means want to take away the joy of collaboration from you a we actually encourage you to ask help not only from assistants, but your friends as well. We are quite goo giving advice, but your friend is more likely on the same wavelength with you and may be able to give you even better advice than we can.

However, on this course it's required that every student learns to write programs independently. This aspe good to keep in mind when you are writing code with one or more friends. You shouldn't use code in your program if you don't know what it does. Ask your friend to explain if you don't understand the given piece advice! Of course, if your friend is a good team player and has read this guide, then maybe he or she can instruct you in a way which you too understand what the code is doing. When you copy someone else's c everyone loses. The one who copies doesn't get any smarter when they can't understand what the copied of the code does, and the one who allowed their code to be copied can't get to deepen their understandir the issue by shaping it to a more comprehensible format.

Unfortunately these cases are encountered annually, so along with the beautiful thoughts we also have explicit and easily understandable rules concerning collaboration on this course. By following these rules

while the case is being resolved. In the worst case you may have to retake the whole course. These rules c be wrapped up to a few sections:

1. Don't copy code from anywhere or anyone and make sure you understand everything you write to you program
2. Always report if you have collaborated with someone. There's an individual text field for this in file tas
3. If you take an example from somewhere else (internet), tell this in comments above the code and tell where you found it

Group projects are allowed in the course (max group size 3). However if you do so, it's your collective responsibility to keep everyone in the group on the same page about how the code works, and that every contributes. Project review is an evaluation of how well you learned programming and show it in your wor Even the greatest code ever written will be rejected if it was clearly made by someone else.

All projects are ran through a tool that inspects code similarity rather extensively. If we find too high simil between two or more projects, all involved parties will be questioned before deciding how to proceed.

Acceptance

Your course work will be accepted if it fulfills the core requirements specified in this document. Unless you code is extremely obnoxious, any fully functional programs are accepted, and there is a chance to improve your work after review. In addition to submitting a functional program, you are also required to reserve a 1 minute review session with a course assistant.

Projects are not given a grade. We've left the old code quality scoring criteria visible below as guidelines f you. You can use them to evaluate whether your code is suitable for someone else's eyes.

Quality of the code

A good overview of the quality of your code can be obtained from the Pylint review which has been includ in the project return box. Some further notes have been written below.

Variables and naming

- bad: It is impossible to deduce what the variables and functions are for or what they contain based on their names.
- good: The names for variables and functions are usually informative about what they are used for, although some of the names can be a bit off. Variables can be named with several different languages and/or the naming convention is not very consistent.
- excellent: The program has a logical and consistent naming convention. The number of variables is low and assignment to a variable is not used where it's not needed. Similarly, assignment to a variable has been used to simplify long lines of code.

Conditional statements

- bad: No use of `elif`- or `else`-clauses or their use is very lacking. `and` and `or` operators are not used understood. As a result, the use of conditional statements are very shaky and might work more by accident than by intent.
- good: Conditional statements are mostly sensible. Proper use of `elif` and `else`, nested conditionals,

- excellent: Conditionals are pretty and succinct. All the previously mentioned tricks are used in a way t they improve the readability of code. They make it possible to detect different states so that the user be told what is happening in the game.

Data structures (tuples, lists, and dictionaries)

- bad: No use or very little use of data structures. This is apparent in the code by having a lot of number variables. The few data structures are misused in horrible and naughty ways.
- good: Lists and such are used to create sensible data structures in places where they are obviously needed. In places where the need is not as obvious, there might be use of numbered variables or othe comparable means.
- excellent: Data structures have been used in places where they are obviously needed and in places wh they make the code clearer and more succinct. Nested data structures are also clearly understood.

Loops and iteration

- bad: All loops are made in the same way, sometimes by brute force. For example all loops might be ma with a `while True`, even if the loop clearly iterates over a list.
- good: The student is able to recognize when to use `while` and when to use `for`. The use of loop varia in for loops might be a bit unclear, or loop creation in general is not optimal.
- excellent: Loops are efficient and the number of iterations in them is as low as possible. Loop variables used well, especially in the case of nested datastructures. Tricks such as enumerate are used where proper.

Functions

- bad: No functions at all, or used just for the sake of using functions. Code is not divided into logical pieces. Using global variables instead of proper arguments for functions.
- good: Functions divide the program into logical segments, even if the segments can be a bit too large times. The use of arguments and return values is correct. The use of global variables is low and only in places where it makes sense.
- excellent: Functions are divided so that they are all short and limited in their purpose. Only the main function can be a bit longer and fatter, but even it should be within sensible limits. The passing of arguments is sensible in the way that everything does not have to be passed everywhere. If there is a use of global variables, all of them are contained in one or two dictionaries.

Modules

- bad: No modules, the program does not function the proper way. Zero points can also come from usir `from import` even when it does not make any sense.
- good: The required modules have been found and are used properly.
- excellent: The program uses some more complicated features of modules (for example the formatting time) or a module has been used to create additional features. The program has been divided into modules of its own, if needed (in the case of clear and succinct code, usually not needed).

Error handling

- bad: Program has no error handling, and every time something goes wrong, the Python stack trace is shown to the user. The program also doesn't solve erroneous states without breaking, which can resu a locked up or inescapable state in addition to crashing.

- good: Error are handled properly with the exception of some exotic cases. The user always has some s of clue how to act so that the program does not report an error. Error handling methods might not be most optimal.
- excellent: Error handling methods have been selected so that the user always gets the best possible information in the case of an error. `try`-dblocks mostly contain a minimal amount of code. The progra can deal with even the exotic error cases.

Files

- bad: No use of files or reading files is done with some odd way (that was not taught on this course). Alternatively, the code only writes files but does not read them.
- good: Files are read and written in some way that the statistics requirement is filled. Alternatively the game has some other way to use files, like saving and loading.
- excellent: Files are used intelligently so that anyone can deduce just by looking at the files how it shou be interpreted by the program.

Return Box

1.00 / 1

3 answers
Frequently Asked Questions
Assessment Criteria

You can return your project here. If you're returning multiple files, you can either zip them or use `Ctrl` wh selecting files to upload many at once.

Jos pakkaat tiedostoja, pakkaa ne zip-muodossa!
Allowed filenames: *.py, *.zip
Submit your files here: Choose Files No file chosen
Send answer You have already answered this task correctly.

Evaluation

In order to pass the course the student or group must undergo a 15-minute evaluation session with the course assistant who reviewed their work. You can book your evaluation time with a form which will be ac to this site at the end of the course.