TSKS37 Lab 1
Getting Started

Fall 2024

Version of this document: January 20, 2025

Examination

The examination for this first lab will focus on:

- The task labeled "Exercise" in `lab1.ipynb`.
- Answering questions on part 4 in `lab1.ipynb`, both written in `lab1.ipynb` and asked by the lab assistant.
- The final LaTeX report from the plotting part, with a focus on readability of the presented data.

The lab assistant may also ask for an explanation of the written code, and stylistic choices regarding the plots.

1 First start

To prepare for the hands-on sessions in the computer lab, or remotely by thinlinc,

1. Create a folder to store all files associated with the labs, a good choice is `~/TSKS37/`. You create it with

   ```
   mkdir ~/TSKS37/
   ```

   You can then navigate into this folder by executing

   ```
   cd ~/TSKS37
   ```

2. Next, we copy the given files from the file server to the folder [1].

   ```
   cp -r /courses/tsks37/ .
   ```

   (Note the period, ".", at the end!) This is the directory you will work on in all the labs.

All files should be in the correct place.

In each terminal session you will need to load the required dependencies

```
module add courses/TSKS37
```

Using a private computer

While everything is setup in the computer halls, there might be situations where you want to use a non-liu computer to complete the labs.

- You can use Thinlinc for graphical remote login. Thinlinc clients exist for Windows, Mac, and Linux.
- You can also make a local installation. This lab use standard modules and there exists many guides online which go into depth on how to install on many different OS. We give a brief overview in Section 5. The given files are also available in lisam.

2 What is Jupyter?

Jupyter is a useful tool for Python development that provide a more interactive experience. Two main features are:

1. Jupyter runs the code in a persistent process, called the *kernel*, which keeps your variables between executions. You can visualize this as having a Python process continuously in the background, which runs the code you send to it. The variables are kept until you exit the development environment or manually restart the kernel. This often makes experimenting much easier and faster, especially when considering lengthy operations.
2. You can seamlessly mix code and its output (including graphs!) with formatted markdown text. This makes Jupyter notebooks a versatile option for presentation and visualization.

[1] All files for the labs are also available in Lisam

Jupyter comes with a native web interface called Jupyter Labs, which offers advanced features like debugging, along with a simpler version known as Jupyter Notebook. Additionally, many popular development environments like Visual Studio Code (VSC), Spyder, or PyCharm offer support for Jupyter, allowing you to integrate Jupyter seamlessly into your existing workflow. When you are done with coding and such you can export the whole document as a PDF!

Another nice feature of Jupyter is its support for multiple programming languages beyond just Python. It supports languages like R, Julia, and more, making it a tool for a wide range of tasks, not just coding python.

2.1 Jupyter Labs Web Interface

By loading the course module you will get all the required packages for Jupyter loaded. We will throughout focus on Jupyter Labs enviroment, but most features are available in other software too.

Let us launch Jupyter Labs, assuming your command line tool has navigated to your `tsks37` directory created in the first section,

```
jupyter-lab lab1.ipynb
```

You should first see some output in the terminal followed by a browser opening the notebook `lab1.ipynb`. Now we can see the jupyter interface with code and text mixed in the middle, as shown in Figure 1.

A few key areas we want to highlight straight away:

1. Run your code. This button will run **only** the selected cell (indicated by the blue line highlight to the left).

    1a. More run options, such as *Run all* or *Run above cells*.

2. Kernel restart button. (*i.e.* clear the variable memory).

3. Add a new block

    3a. Change type of block between code and markdown (text).

4. Common feature for editing code, such as search and replace.

5. Output. Each code cell will output whatever is assigned/returned on the last row, and what else has been printed during execution. A really handy feature is that any plots made will be shown here!

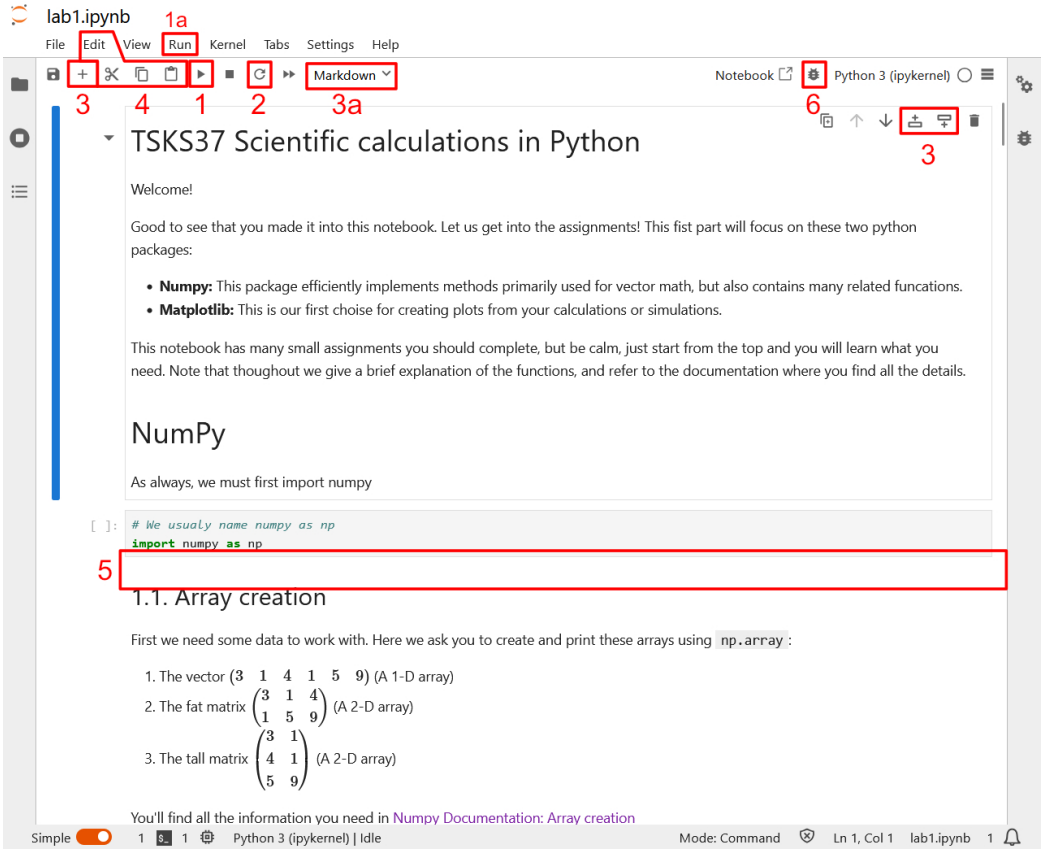

Figure 1: The general interface of Jupyter Labs, with some important buttons marked.

6. Enable debugger button, we will explain this functionality in more detail later in Section 3.

2.2 On using classes or functions

When using classes or functions in Python, it is important to note that functions are essentially variables. Therefore, if you edit a function, you need to rerun the block containing the function (or import of the module) to update it; otherwise, the previous version of the function will still be used. This ensures that your code reflects the most recent changes and operates as expected. While this is always true for python it is more prevalent when using Jupyter.

2.3 Create a new Jupyter notebook

Two options:

1. Create an empty file, give it `.ipynb` as extension and open it as usual.
2. In Jupyter Labs (or Notebook); use File → New to create a notebook.

3 Debugging

While this block divided style makes the "print-based" debugging easier, Jupyter Labs contains a built-in debugger! If you have not used a tool like this before this is a good time to explore its capabilities, which include line by line execution and variable inspection. In Jupyter Labs we must first enable the debugger by pushing the debugger button (in the top right) such that it turns orange.

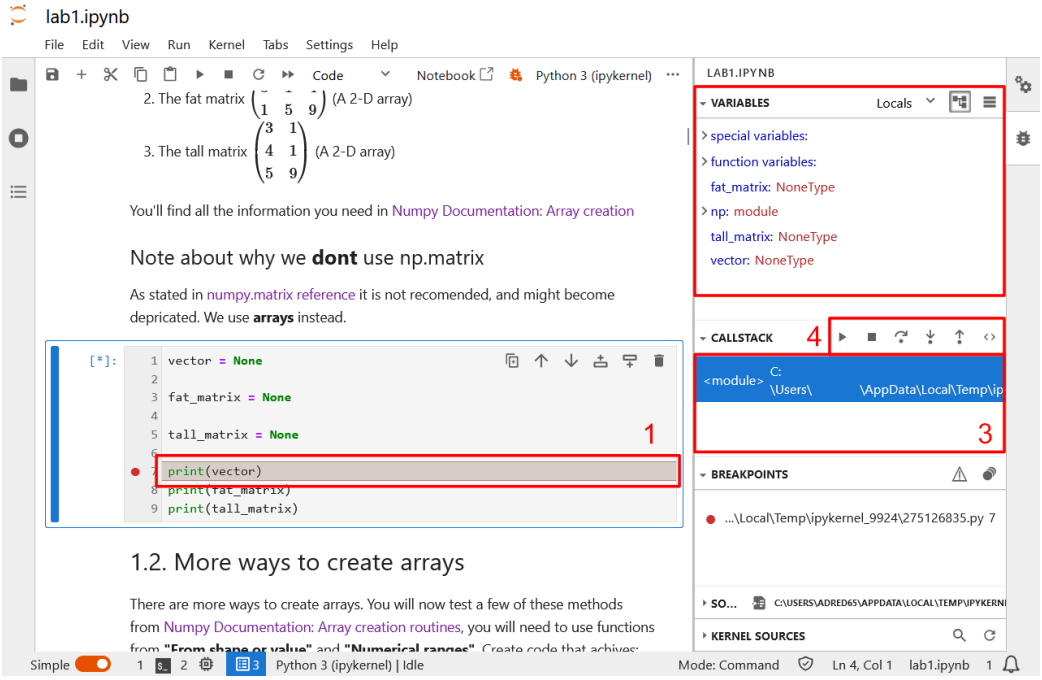

Figure 2: Look of jupyter lab with the debugger enabled.

3.1 Using the debugger

Now, when running the code as usual the debugger is enabled, allowing for the program to pause on *breakpoints*, which you place by pressing to the left of a row number in a code block. Each breakpoint is listed on the right side and indicated in the code by the little red dot next to the line number.

Now, when your code halts at a breakpoint we illustrate ways to inspect it in Figure 2. A few places has been marked in the image and we explain them below:

1. The line where execution halted (paused).
2. The variables in their current state.
3. The call stack, see how your program got to this point. You can also inspect the variables in the outer scopes.
4. When you have examined the current state of your program you have a few option on how to continue:
   (a) **Continue:** Continue execution.
   (b) **Step over:** This will execute the current line and step to the next line.
   (c) **Step into:** If the current line calls a function the debugger steps into the start of the function.
   (d) **Step out:** This will execute the rest of the function and return to the outer scope. If you are not in a function it will run the rest of the file/block and exit.

Note that this is a brief overview, and you can continue reading in the official documentation to learn more: `https://jupyterlab.readthedocs.io/en/stable/user/debugger.html`

3.2 Interactive terminal

Being able to test functionality directly is a good way to quickly test things, or debug. In Jupyter Labs simply navigate: File → "New console for notebook", which opens a console where you have direct access to all your variables stored in the kernel.

4 Lets get started!

Now, you have all the prerequisites to get started with the first lab. The majority of the assignments are located in the Jupyter Notebook `lab1.ipynb`.

There is also extra material provided for the plotting assignments:

- `plot.pdf`: Which contains a complete description of how to create the plots and a explanation of the data.
- `lab-report.tex`: A LaTeX source file containing a template in which you will insert some of the plots you have created. This is further explained in `plot.pdf`

`lab1.ipynb` should already have copied in the first step. Open the notebook to get started with the assignments!

5 Setup on a private linux computer

We present this as a brief guide, including the most important information (and good practices) when setting up the environment.

While this *should* work for any modern python 3 installation, the version used on the lab computers is 3.10. We assume that `python` refers to this version. Using a virtual environment (venv) is a good practice, and we recommend using one. Assuming you have navigated to your lab folder, you create the venv using

```
python -m venv TSKS37-venv
```

In each terminal session when you want to use the venv you must activate it. Assuming you have navigated to your lab folder, you active it by

```
source TSKS37-venv/bin/activate
```

With the venv activated you can now install the packages:

```
numpy
matplotlib
jupyter
ipykernel
scikit-learn
pyaudio
pydub
```

using pip.

Note that `ffmpeg` should **not** be downloaded through pip. You can download prebuilt binary files from `https://ffmpeg.org/`. The folder with the downloaded binaries (often called `bin`) should the be added to the path. `ffmpeg` is also often available though common package manager such as `apt`.

5.1 Running the labs locally on a Mac or Windows computer at home

The process is similar as compared to installing on a linux machine. We unfortunately cannot offer technical support for Windows or Mac.

5.2 Troubleshooting

- The way LiU's system is configured, the directory `/courses/tsks37/` is not always visible in the file browser or via autocomplete (Tab) on the command line. To reach the directory from the command line, write

```
cd /courses/tsks37
```

- For support on LiU's computer system, call 013-282828 or write to `helpdesk@liu.se`.