MPS115/MPS116 PRESENTATION LAB 11

On this sheet you will combine LaTeX and HTML with Python. First you will use SciPy to integrate numerically, then learn how to export data from Python by writing to files which can then be read by other software.

1. More integration and SciPy

Recall from Introduction to Probability and Data Science that the probability density function (pdf) of the standard normal distribution $N(0,1)$ is given by

$$\phi(z) := \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-z^2}{2}\right),$$

so that for Z a normally distributed random variable $P(a \leq Z \leq b) = \int_a^b \phi(z)\,\mathrm{d}z$. The cumulative distribution function (cdf) is given by

$$\Phi(z) := P(Z \leq z) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-t^2}{2}\right)\,\mathrm{d}t.$$

This integral has no simple expression so needs to be calculated numerically. You saw previously how to calculate approximations to definite integrals. However, in this case we want to define an integral from $-\infty$ so the previous method wouldn't work. We will use the integration functions from the SciPy (short for Scientific Python) module. To integrate a function `f` from `a` to `b` you use `scipy.integrate.quad(f, a, b)` as in the following.

```python
import numpy as np
import scipy.integrate as integrate


def pdf(z):
    """Calculate the probability density function."""
    return np.exp(-z**2 / 2) / np.sqrt(2 * np.pi)

integral = integrate.quad(pdf, 0, 2)
print("The integral of phi from 0 to 2 is:", integral)
```

The output is a pair of numbers in round brackets: `(result, error)`. The calculated `result` is within `error` of the actual integral. Note that this expression is not a list as it is in round brackets rather than square brackets. This expression is a type called a tuple; it is a bit like a list, but you can't change a tuple. The important point here is that you access the individual entries in a tuple in the same way that you do with a list, so to obtain just the approximation to the integral and not the error, you should change the final line to the following.

```python
print("Prob(0 <= Z <= 2) =", integral[0],
      "plus or minus", integral[1])
```

This integration method will allow us to use ∞ and $-\infty$ as limits. We do this with `np.inf` and `-np.inf`. For instance, add the following lines.

```python
integral = integrate.quad(pdf, -np.inf, 0)
print("Prob(Z <= 0) =", integral[0],
      "plus or minus", integral[1])
```

We will ignore the error term in this sheet, as you can often do unless you are interested in precise numerics. We can now get Python to plot the cdf within a range and to tabulate its values. Download the following program `cdf.py` from the module website (right-click, save-as). Run the program.

```
 1 import numpy as np
 2 import scipy.integrate as integrate
 3 import matplotlib.pyplot as plt
 4
 5
 6 def pdf(x):
 7     """Calculate the probability density function."""
 8     return np.exp(-x**2 / 2) / np.sqrt(2 * np.pi)
 9
10
11 MAX = 3.5
12
13 z = np.linspace(-MAX, MAX, int(2*MAX*16 + 1))
14
15 cdf = np.zeros_like(z)
16 for i in range(len(z)):
17     cdf[i] = integrate.quad(pdf, -np.inf, z[i])[0]
18
19 plt.plot(z, pdf(z))
20 plt.plot(z, cdf)
21 plt.axis([-MAX, MAX, 0, 1.05])
22 plt.show()
23
24 print("\n{0:^5}  {1:^6}".format("z", "cdf(z)"))
25 print("-"*13)
26 for i in range(0, len(z), 8):
27     print("{0:5.2f}  {1:.4f}".format(z[i], cdf[i]))
```

Let's have a look at some of the lines.

Line 11	defines the constant `MAX`. We will be calculating the cdf from `-MAX` to `MAX`.
Line 13	defines the numpy array of points at which we will calculate the cdf. There will be 16 points calculated between each pair of integers.
Line 15	initialized the array in which we will store the corresponding cdf values. It makes an array with the same number of entries as `z` but with all the entries zero.
Lines 16–17	calculates the cdf for the various values and stores it in the array.
Lines 19–22	plot the graph of the cdf and the pdf.
Lines 24–27	tabulate the values of the cdf. Note that the 8 on line 26 means that we only tabulate every 8th value. This is just so as not to display a massive table. Change the 8 to a 1 and see what happens.

Now in the rest of this lab we will look at ways of exporting the data we have just created, so that it can be used in other software.

2. Writing a file that pgfplots can read

A graph in a LaTeX document looks a lot better if it is drawn by pgfplots than if it is just an imported picture; for one thing, the fonts will actually match. We will get Python to export the function values to a file and then get pgfplots to read in the values from the file to draw the graph.

Add the following line to the above program and run it.

```
29 with open("cdf.data", mode='w') as cdf_file:
30     for i in range(0, len(z)):
31         cdf_file.write("{0:.2f}  {1:.4f}\n"
32                        .format(z[i], cdf[i]))
```

The file `cdf.data` should have been created in your working directory. Open the file in a text editor. You should see a table of z-values and the corresponding cummulative density function values.

Let's analyse what was done here. It is very similar to the table printing at lines 26 and 27 of the program above. The main difference is that it is part of a `with` program block and we use the `file_object.write()` method rather than the `print()` function. To get python to write a file, the general form is as follows.

```
with open(file_name, mode='w') as file_object:
    indented program block
    containing statements of the form
    file_object.write(string)
```

The first line creates a Python *file object* which refers to the file with the name `file_name`. The `mode='w'` part means that we will be 'w'riting to this file. When this command is issued, if the file already exists then its contents are deleted. If you wanted to add some data to an already existing file then you use `mode='a'`, which stands for 'append'.

The command `file_object.write(string)` will just output the string `string` to the file with name `file_name` instead of printing it to the screen.

The file `cdf_graph_demo.tex` is available on the module webpage (right click to save). Save it in the same directory that your file `cdf.data` is in, and LaTeX it.

You should see the graphs of the pdf and the cdf. Much of the code in `cdf_graph_demo.tex` is setting up the plot. The relevant lines here are lines 27–31 We have an explicit formula for the pdf, so that is plotted by pgfplots on lines 27–30. Line 31 is the following.

```
31 \addplot [mark=none, thick] file {cdf.data};
```

This plots the cdf from the data that is in the file `cdf.data`; the keyword `file` tells pgfplots to look in the given file.

3. Writing LaTeX and HTML with Python

The new piece of Python you will need here is the idea of *raw triple quotes* to define a string. Note that on the first line of the following the 'r' stands for 'raw' and that you need three quotation marks at the beginning and at the end. Try the following.

```
string = r"""This string appears as typed.
Things
like \t and \n are not converted.
```

```
It is like using \begin{verbatim} in LaTeX.
"""
print(string)
```

To get the middle pair of lines using the ordinary construction for a string you would have to use something like the following.

```
string_2 = "Things\nlike \\t and \\n are not converted."
print(string_2)
```

So it is sometimes useful to use this construction if your string will have lots of backslashes and newlines, like in a tex file, as you will see below.

Now add the following code to your Python program `cdf.py`.

```
HEADER = r"""\begin{center}
  \begin{tabular}{rr}
    \toprule
    $z$ & $\Phi(z)$ \\
    \otoprule
"""

FOOTER = r"""    \bottomrule
  \end{tabular}
\end{center}"""

with open("cdf_table.tex", mode="w") as tex_file:
    tex_file.write(HEADER)
    for i in range(0, len(z), 8):
        tex_file.write("    ${0:.2f}$ & ${1:.6f}$ "
                              "\\\\ \n"
                               .format(z[i], cdf[i]))
    tex_file.write(FOOTER)
```

Run the program. It should have created a file `cdf_table.tex` in your working directory. Open it and examine it. This is a ready-built piece of LaTeX that you can input into any LaTeX file.

Add the following lines to the end of the LaTeX file `cdf_graph_demo.tex`.

```
Here is the table.
\input{cdf_table.tex}
```

Now LaTeX the file `cdf_graph_demo.tex` and see the table appear.

Exercise 1. Copy your Python program from above and save it with a different name, such as `cdf_html.py`. Now edit the program so that it saves the graph as an SVG image file `cdf_graph.svg` and writes an HTML file `cdf.html` which displays the table of data of cdf values and displays the SVG graph. You can base it on the HTML file `cdf_incomplete.html` which you can find on the module webpage (right click to save).