additional materials for SET 2

kacper.topolnicki@uj.edu.pl

The `set_2.py` script contains additional materials for the second set of exercises.

This tutorial is bodged together from material available at:

- https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.html
- https://docs.scipy.org/doc/scipy/reference/tutorial/stats.html

Importing the necessary libraries

First we will import the `numpy` library and the `matplotlib` library [set_2.py line: 19]

```
import numpy
import matplotlib.pyplot as plt
```

just like we did for the previous set of exercises. Using the `as` keyword will allow us to refer to `matplotlib` using the shorter `plt`. This time we will be using additional functions contained in the `scipy.stats`. This library is imported next [set_2.py line: 24]

```
import scipy.stats as sts
```

and we will refer to it as `sts` in the code.

Ploting the normal distribution

Our first task will be to plot the normal distribution, that is the Gaussian probability distribution function (PDF):

$$f(x) = \frac{e^{-x^2/2}}{\sqrt{2\pi}}.$$

where the real number x is the value of a random variable with a normal distribution. The implementation of this function is available in `sts.norm` (`scipy.stats.norm`). Please have a look at the documentation of this function in *ipython* (using the `?` operator) or **online**. Pay special attention to the two optional arguments `loc` and `scale` as they are relevant to the exercises.

One of the thing we have to figure out first is a sensible range for the values of x. Let's say we are interested in the most prominent, central part of the distribution and we are not interested in it's tails. A more formal way of saying the same thing might be that we are interested in values of x such that $a < x < b$ and

$$\int_{-\infty}^{a} f(x)dx = 0.01$$

$$\int_{b}^{+\infty} f(x)dx = 0.01.$$

The second condition can also be written as:

$$\int_{-\infty}^{b} f(x)dx = 1 - \int_{b}^{+\infty} f(x)dx = 0.99.$$

Now all we need to do is work out what the values of a and b are. Luckily, there is a method `sts.norm.ppf` (procent point function) that can help us. Given a number p between 0 and 1 (the probability), it returns the value x_p such that:

$$\int_{-\infty}^{x_p} f(x)dx = p$$

Armed with this function we can determine the plotting boundaries [set_2.py line: 86] :

```
a = sts.norm.ppf(0.01)
print("a = " , a)
b = sts.norm.ppf(0.99)
print("b = " , b)
```

Notice that $a = -b$. Why? How will this change if we add the optional arguments `loc` and `scale`?

Next we will create a `numpy` array that will hold, say, 100 points between a and b. To do this we will use the `linspace` function (please read the documentation for this function in ipython `?numpy.linspace` or **here**) [set_2.py line: 102] :

```
arg = numpy.linspace(a , b , 100)
print("x in : " , arg)
```

We are almost ready to plot the PDF of the normal distribution, all we need is the function values. These can be accessed using the `sts.norm.pdf` method [set_2.py line: 118] :

```
print("f(a) = " , sts.norm.pdf(a))
print("f(b) = " , sts.norm.pdf(b))
```

This function is automatically mapped over `numpy` arrays so we can just write [set_2.py line: 123] :

```
vls = sts.norm.pdf(arg)
print("f(x) : " , vls)
```

Finally we create the plot, give it a title and describe the axis [set_2.py line: 134] :

```
plt.title("NORMAL DISTRIBUTION")
plt.xlabel("x")
plt.ylabel("$\\frac{e^{-x^{2} / 2}}{\\sqrt{2 \\pi}}$")
plt.plot(arg , vls)
plt.show()
```

Histogram of the normal distribution

At this point we would like to draw a number, say a 1000, of random variates from the normal distribution. This can be achieved using the `sts.norm.rvs` function (please read the documentation using the `?` operator in *ipython* or **here**) [set_2.py line: 153] :

```
gen = sts.norm.rvs(size = 1000)
```

The result is an array of 1000 values, without specifying the `size` we would just get a single number.

Let's compare the PDF and the histogram. First we set the title and the label for the horizontal axis [set_2.py line: 179] :

```
plt.title("COMPARE")
plt.xlabel("x")
```

Next we plot the PDF [set_2.py line: 183] :

```
plt.plot(arg , vls , label = "PDF")
```

and give this plot the label "PDF". Finally, we generate a histogram from the values in `gen` [set_2.py line: 186] :

```
plt.hist(gen , density = True , label = "HISTOGRAM")
```

We will call this plot "HISTOGRAM". The value of `density` is crucial for the comparison with the PDF - can you figure out why? Tip: take a look at the documentation for `plt.hist`. The last step is is to draw the plot legend and show the plot [set_2.py line: 189] :

```
plt.legend()
plt.show()
```

Cumulative distribution function

Let's use some of the methods from the previous class and calculate:

$$\int_{-1}^{1} f(x)dx$$

Since the value of the standard deviation in our case is 1 and the distribution is symmetric around 0, we can expect this integral to be roughly 68.3%. The code is almost identical to what was discussed during the previous class [set_2.py line: 211] :

```
import scipy.integrate as integrate
print("integrate.quad, (integral of f(x) from -1 to 1 , estimated error) = " , integrate.qua
```

First we import the `scipy.integrate` library and then we use the `quad` function. The result, printed onto the screen, should roughly match 68%.

The same result can be obtained by using the built in `scipy.stats.norm.cdf` function. [set_2.py line: 227] :

```
print("sts.norm.cdf, (integral of f(x) from -1 to 1 , estimated error) = " , sts.norm.cdf(1
```

This function returns the value of the cumulative distribution function for the normal distribution. That is, for a given z it returns:

$$\int_{-\infty}^{z} f(x)dx$$

Concatenate and `numpy`

We should dedicate a whole separate class to the `numpy` library. For now, in order to implement the exercises in set 2, we will briefly discuss `numpy` array concatenation and methods to calculate the mean and standard deviation. In [set_2.py line: 227] :

```
print("sts.norm.cdf, (integral of f(x) from -1 to 1 , estimated error) = " , sts.norm.cdf(1
```

we first create two `numpy` arrays `vals1`, `vals2`. Next we create a new array, `vals`, by concatenating them. The new array's size $3210 + 1230 = 4440$ is printed onto the screen. The arithmetic mean and the standard deviation of the values in the new array can be calculated using [set_2.py line: 254] :

```
print("numpy.mean(vals) = " , numpy.mean(vals))
print("numpy.std(vals) = " , numpy.std(vals))
```

Is this the result that we expected? # Other distributions

Please have a look at the documentation for the 'scipy.stats' library **here**. A large number of statistical distributions are available (`scipy.stats.uniform` and `scipy.stats.cauchy`). They all have a similar interface.