Exercises for Matplotlib

In today's exercises you will get used to the basics of plotting with matplolib.

Exercise 1: Controlling (interactive updating) in the python shell

The goal of the first exercise is to understand how you can control the interactive updating. In order to complete this exercise you just have to follow the steps indicated below and try to answer the questions.

1. Open a terminal and start ipython.

```
you@yourMachine:~$ ipython
```

2. Import pyplot and check if it is in the interactive mode:

```
import matplotlib.pyplot as plt
plt.isinteractive()
```

What does it mean that pyplot is not in the interactive mode?

3. Try to plot a sinewave using pyplot and numpy.

```
import numpy as np
x = np.arange(0, 10, 0.2)
y = np.sin(x)
plt.plot(x, y)
```

Can you use the terminal? How can you use the terminal again? Is there a way you could have avoid loosing the figure?

4. Switch on the interactive mode and check it.

```
plt.ion()
plt.isinteractive()
```

Now try to plot the sinewave again using the code snippet in 3. What is the difference between interactive and non-interactive modes?

5. Okay, now let's change something, let's increases the limits in the YAxis using a pyplot function.

```
plt.ylim(-2,2)
```

Did it work?

6. That was easy. Let's now do something more difficult, let's revert the change we just did, but this time using the helper functions in the Axes instance.

```
ax = plt.gca() # literally, get-current-axes (gca) function
ax.set_ylim(-1,1)
```

Did it work this time? What do you have to do in order to see the change?

7. Yes, that's right, you have to invoke the draw function!

```
plt.draw()
```

What is the difference between `draw()` and `show()`?

8. Finally, can you guess what would have happened if you would have started the ipython session with the pylab mode? Do you need to import pyplot or numpy if you are the pylab mode?

Exercise 2: Scripting in pyplot-style

The goal of this exercise is to become acquainted with the most commonly used pyplot functions.

For this exercise you are going to write a little python script. So first open your favorite text editor (gedit is highly recommended for beginners) and create an empty file that you are going to save with the name `pyplot_exercise1.py`.

For this exercise, I would recommend you to use the ipython shell together with gedit. First, try out the different commands in the shell and, once you have achieved the desired results, add the line of code to the script. Don't forget to turn on the interactive mode!

You will successfully complete this exercise when the output of your `pyplot_exercise1.py` file looks like this:

```
you@yourMachine:~$ python pyplot_exercise1.py
```

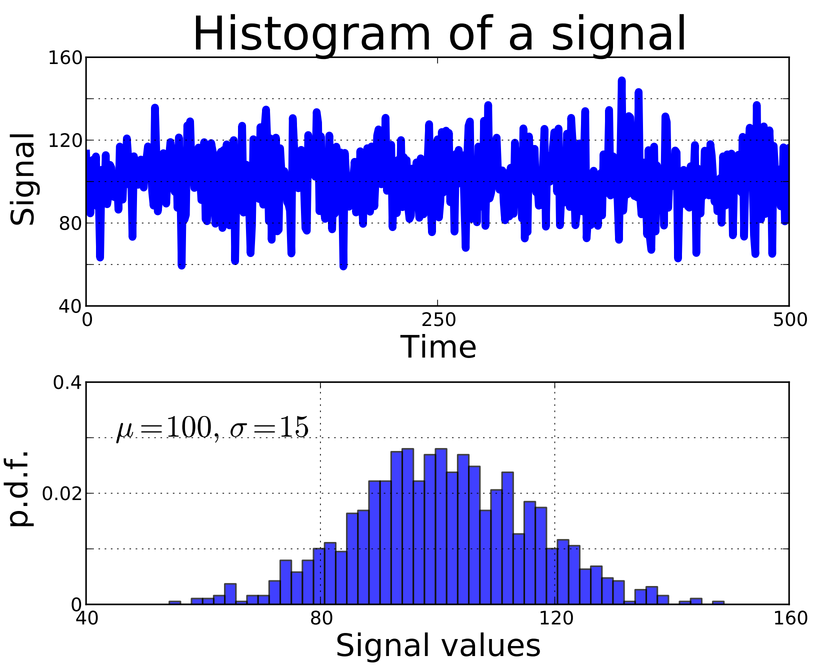

Figure 1: Plot from exercise 1

Remember that for this exercise you are not supposed to use objects but work with pyplot functions only!

1. Import pyplot and numpy libraries.

2. Create a noisy signal using the following code snippet:

```
mu = 100
sigma =  15
t = np.arange(1000)
x = mu+sigma*np.random.normal(size=len(t))
```

3. Create the upper subplot using the `subplot()` function.
4. Plot the signal using the following 2Dline properties:
   - solid
   - blue color
   - width of 5
5. Add the label to the axis with a font size of 20 using `xlabel()` and `ylabel()`.
6. Use the function `xlim()` to select the appropriate range of interest.
7. Choose the position of the ticks and their labels according to what you see in **Figure 1**. Make the font size of the tick labels 12. Use the functions `xticks()` and `yticks()`.
8. Use `grid()` to draw grid lines using the yticks only.
9. Create the lower subplot. This time you can try to use the pyplot function `axes()`.
10. Create a histogram of the signal using `hist()` and with the following characteristics:
    - It contains 50 bins.
    - It is normalized (it has to be a probability density).
    - The bars are blue inside and the color is 25 % transparent.
11. Use the function `text()` to write the values of μ and σ.

    Note that matplotlib accepts TEX equation expressions in any text expression. For example to write the expression $\sigma_i = 15$ in the title, you can write a TEX expression surrounded by dollar signs:

```
plt.title(r'$\sigma_i=15$')
```

12. Finally, modify the lower subplot, adding labels and selecting ticks, in the same way you did it before.

Exercise 3: Scripting in object-oriented (pythonic) style with pyplot

The goal of this exercise is to use pyplot in a more object-oriented (pythonic) way. This is the recommended way of plotting in general, although it is a bit more wordy than using pyplot functions directly, it is more powerful and flexible. You will learn how to make functions that take an Artist instance as argument and use its helper methods to modify it in a very general and reusable (pythonic) way.

Create a python script called `pyplot_exercise2.py`. This time, your plot should look something like this nice colorful one down here!

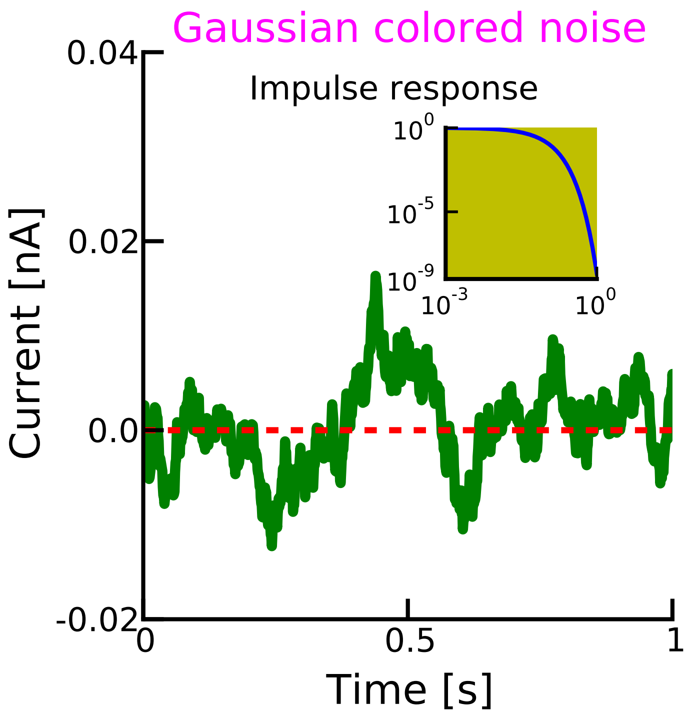

Figure 2: Plot from exercise 2

1. Import pyplot and numpy libraries.
2. Create a filtered random variable using the following code snippet:

```
dt = 0.001
t = np.arange(0.0, 10.0, dt)
r = np.exp(-t[:1000]/0.05)              # impulse response
x = np.random.randn(len(t))
s = np.convolve(x,r)[:len(x)]*dt        # colored noise
```

3. Create a figure object with following characteristics:
    - figure number
    - size 5 inches width and 4 inches height

- resolution of 100 dot-per-inch (pixel per inch; 500X400 pixels)

4. Make sure that you don't redraw in the same figure multiple times without erasing the previous plot using the figure `clf()` method (clear-figure).
5. Create the main Axes instance using the figure's function `add_axes()`
6. Plot the colored noise. You should access the Axes instance via the `fig.axes` list (even if it doesn't seem the most intelligent thing to do in this case). Don't set other properties yet!
7. Similarly, plot a dashed line at 0 nA.
8. Access both lines via `ax.lines` list and use the set methods to make them look like the lines in **Figure** 2.
9. Create a python function that takes an Axes instance as an argument together with three integer values. Pretty much like this one:

   ```
   def create_beautiful_axis(ax,lw=2,ms=10,mew=3):
       pass
   ```

   This function should remove the right and top spines, make the left and bottom spines thicker and increase the size of the ticks. Note that the spines are the longer lines that you see in the Xaxis and Yaxis and ticks are the small ones. In order to implement this wonderful function, follow this steps:

   - Set the color of the top and right spines to `none`.
   - Set the position of the ticks to the bottom and the left.
   - Use the first argument to make bottom and left spines thicker.
   - Loop through the xticklines and yticklines using the Axes's `get_xticklines()` and `get_yticklines()` methods and make those ticks bigger using the last two arguments (ms=`markersize` and mew=`markeredgewidth`).
10. Use your brand new function to make your Axis look nice!
11. Congrats, you're almost done with this Axes! Now, the rest of things that you need to do in order to make the figure look like the one in **Figure** 2 are the same as in the previous exercise. But remember to use fig.axes and axes.list!
12. Create the inset plot. The only difference between this plot and the main plot is its size and location in the figure, the rest is the same. Use the `plot()` method (please).
13. Set the scale of both Axis to logarithmic instead of linear using the Axes' `set_xscale()` and `set_yscale()` methods.
14. Make the inset plot look like the one in **Figure** 2 with the exception of the ticklabels and the text on the top, which we'll leave for the last two steps.
15. For plotting the text use the Axes' function `annotate()`.
16. Wrap the whole thing up in a function (leaving the `create_beautiful_axis()` function and the snipped to generate the data out). Include an option to save the figure instead of plotting it. Use pyplot's `save()` method and save it as a pdf (vector graphics publication quality format).

Extra bonus!

- Use `locator` and `formatter` objects to make the logarithmic ticklabels look more scientific.
- Draw a fancy arrow connecting the text with the inset plot.

You're done (for today)! Hope that you have enjoyed it! Now you're a matplotlib master! For more info don't forget to check the matplotlib website. `http://matplotlib.org/index.html`