exercises-no-solution

March 24, 2015

```
In [2]: from IPython.core.display import Image
```

1 Numpy Exercises

```
In [3]: # Note that this Document is designed to be executed from top to bottom.
        # General imports, numpy/matplotlib may not be done in all examples.
        import numpy as np

        import warnings  # Just used for some examples in the solutions....
```

1.1 Array creation and manipulation

By using miscellaneous constructors, indexing, slicing, and simple operations (+, −, *, :), large arrays with various patterns can be created. Find a way to create these arrays.

1. Create the following arrays:

a)

$$\begin{pmatrix} 1. & 1. & 1. & 1. \\ 1. & 1. & 1. & 1. \\ 1. & 1. & 1. & 2. \\ 1. & 6. & 1. & 1. \end{pmatrix}$$

b)

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 6 \end{pmatrix}$$

2. Form this 2-D array (without explicitly typing it):

$$\begin{pmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{pmatrix}$$

1.1.1 Solution

```
In [3]:
```

```
In [3]:
```

```
In [3]:
```

1.2 A simple calculation

1. Devide each column of the array `a` elementwise by the array `b` (do you notice some problem?)

In [3]:

1.2.1 Solution

In [3]:

1.3 Sorting and more

a) Generate a 10×3 array of random numbers ($\in [0, 1[$). For each row, pick the number closest to 0.5. Use `abs` and `argmin` to find the column j closest for each row. Use advanced indexing to extract the numbers. (Hint: `arr[i,j]` – the array i must contain the row numbers corresponding to stuff in j.)

b) Find the two points closest using `argsort`. Advanced: Look at `argpartition` can you figure out how to improve the speed of your algorithm? Find the fastest method and test it on larger arrays!

1.3.1 Solution

In [3]:

In [3]:

2 Matplotlib Excercises

In [4]:
```
# Make the IPython notebook matplotlib aware:
%matplotlib inline
import matplotlib
# And fix the default savefig dpi that ipython sets very low
matplotlib.rcParams['savefig.dpi'] = 120

# Import the pyplot interface to matplotlib:
from matplotlib import pyplot as plt
```

2.1 Simple Plotting

1. Plot a the sine function from 0 to 6 (inclusinve) using 50 points.
2. Plot the line again in red and with diamond shaped markers of size 5.
3. Now plot the line, but include a legend.

Helpful functions: `plt.plot`, `plt.legend`, `plt.grid`

2.1.1 Solution

In [4]:

In [4]:

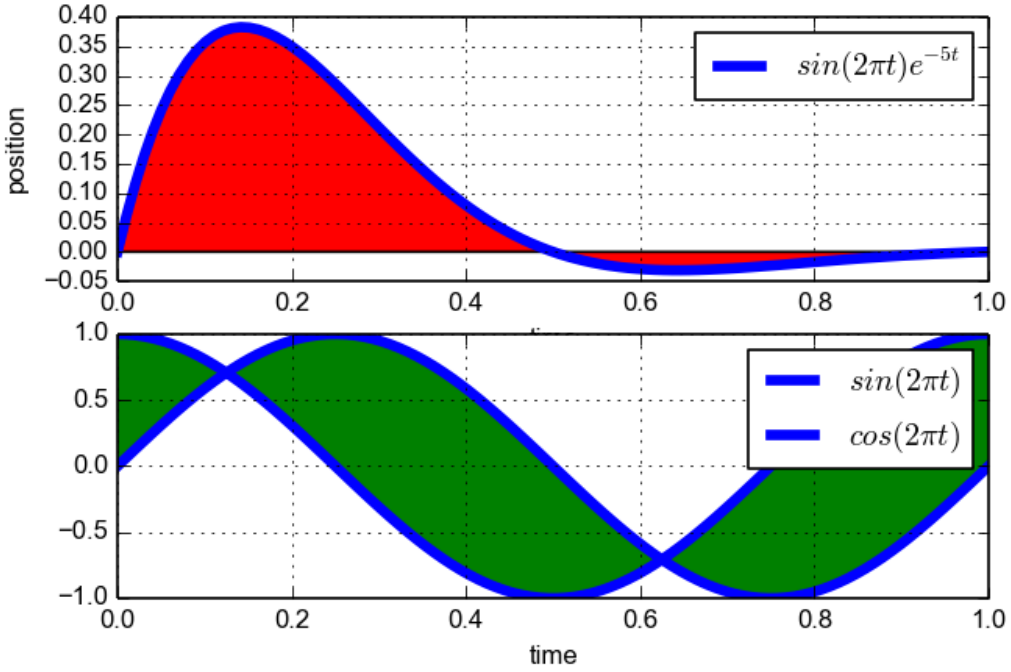
In [4]:

2.2 Simple Plotting II

Try to recreate the following plot:

```
In [5]: Image("pics/simple_plotting_2.png")
```

Out[5]:



Helpful functions: `plt.fill`, `plt.fill between`

2.2.1 Solution

```
In [5]:
```

3 Simple Tasks

3.1 Polynomial Fitting

1. Generate data by a polynomial function $f(x) = -3x^3 + 2x - 8$
2. Fit a polynomial (`np.polyfit()`) and recover the coefficients of the polynomial.
3. Try the same procedure with noisy data. (use np.random.randn() to add noise to the data)

3.1.1 Solution

```
In [5]:
```

3.2 Nonlinear Fitting

1. Define the function $f(x) = e^{-ax} + b$
2. Generate noisy data from that function with parameters $a = 2$, $b = 1.4$
3. Make a simple plot of the data
4. Use `scipy.optimize.curve_fit()` to estimate a and b from the data and plot the results.

3.3 Solution

In [5]:

4 Integration

1. Define the function $f(x) = e^{-(x+3)^2}$. Make sure the function also accepts lists/array-likes.
2. Calculate the integral $\int_{-20}^{20} f(x)\,dx$ by hand using Riemann sums.
3. Use `scipy.integrate.quad()` to evaluate the integral.
4. Try to calculate $\int_1^\infty \frac{1}{x^2}\,dx$ and $\int_1^\infty \frac{1}{x}\,dx$

4.0.1 Solution

In [5]:

4.1 Solving Systems of Linear Equations

You find three shopping bags with following content:

- bag A: 10 kg apples, 5 kg pears, 1 kg oranges (35.35 EUR)
- bag B: 1 kg apples, 8 kg pears, 1 kg oranges (24.91 EUR)
- bag C: 9 kg apples, 3 kg pears, 5 kg oranges (40.38 EUR)

Determine the price of apples, oranges and pears: 1. Formulate a linear system of equations describing the shopping bags. 2. Use `scipy.linalg.solve()` (or `numpy.linalg`) to solve the system of equations. 3. Verify the results by using `np.dot()`.

4.1.1 Solution

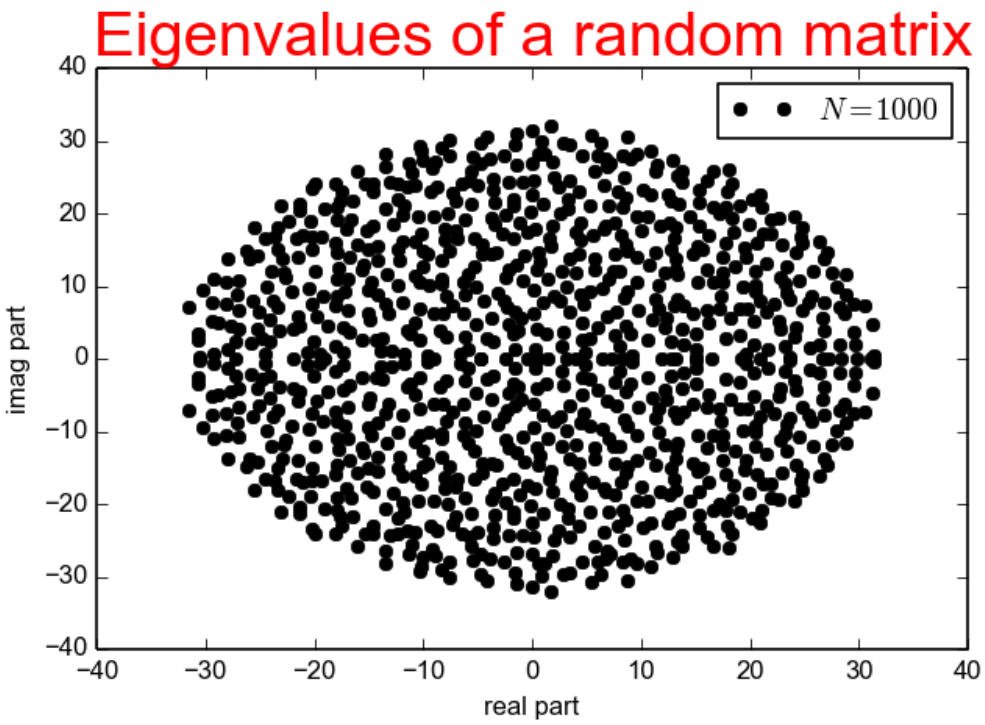
In [5]:

4.2 Some Linear Algebra

1. Create a random matrix, where the entries are randomly chosen from a standard normal (Gaussian) distribution.
2. Plot the (complex!) eigenvalues of that matrix, add labels to the axis and a legend.
3. Add a title for your plot and make it red and increase the size.
4. Store your graph as an `.pdf`. It should look something like this:

In [6]:
```
Image("pics/matrix1.png")
```

Out[6]:



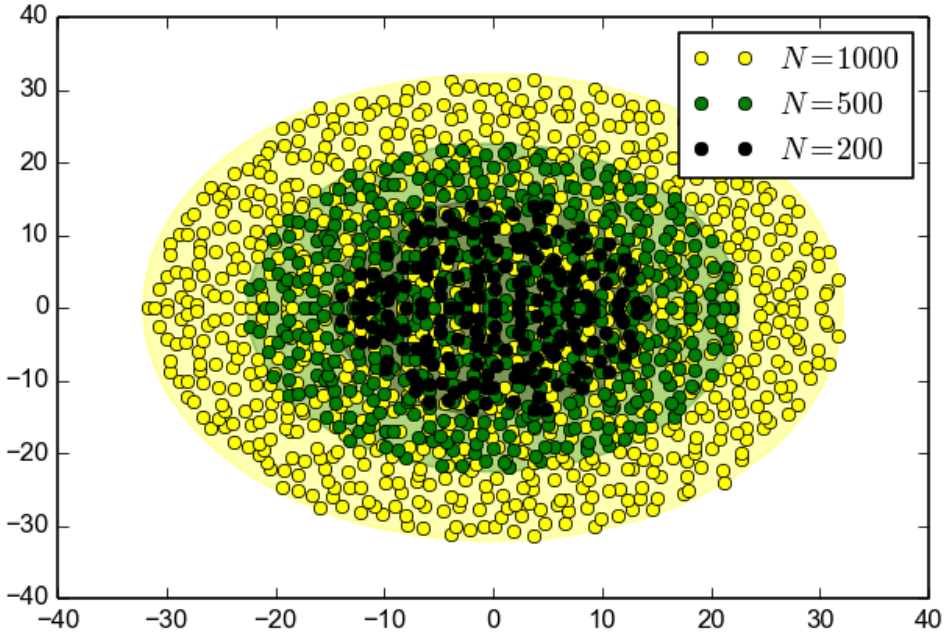
5. Define a function that returns True if a point (x, y) is contained in a circle of radius r around the origin. Make sure it works for array inputs.

6. Define a function to find the smallest circle containing all the eigenvalues of your matrix.

7. Plot the eigenvalues for a matrix of size N = 50, 100, 200, 500 together with the smallest disc containing the eigenvalues. Therefore first import the matplotlib module `matplotlib.patch` and have a look at the classes contained in that mod- ule. Having identified a suitable class create a patch (a circle) and try to add it to your plot. (*Hint:* Patches can be added to axes objects)

```
In [7]: Image("pics/matrix2.png")
```

```
Out[7]:
```



4.2.1 Solution

In [7]:

4.3 Basic Statistics

1. Define a 1D Gaussian probability density function with $\mu = 10$ and $\sigma = 5$ using the formula

$$\frac{1}{\sqrt{2\pi} \cdot \sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

2. Confirm that the integral of this function is approximatly 1.
3. Instead of defining the distribution yourself, now use *only* `scipy.stats.norm` to:
4. Draw 1000 samples. And plot a histogram using matplotlib.
5. Do a maximum likelyhood estimation to find the parameters of the gaussian from the random samples. (For a gaussian these are just the mean and standard deviation, but use `scipy.stats.norm` instead of calculating these yourself)
6. Add both the estimated and the real Gaussian to the plot. If the histogram is not normed correctly already, norm it to fit the lines (see the histogram documentation).
7. Find the points that include 90% around the mean in the original distribution. And confirm that these include 90% using both the comulative distribution function (`cdf`) and the original integration method.

4.3.1 Solution

In [7]:

4.4 Power Spectrum Analaysis

A way to look at a signal is to view its spectral density (i.e., the Fourier transform of the signal). The Fourier transform views the signal as a whole. It swaps the dimension of time with the dimension of frequency. One can think of the Fourier transform as a combination of slow and fast oscillations with different amplitude. A very strong and slow component in the frequency domain implies that there is a high correlation between the large-scale pieces of the signal in time (macro-structures), while a very strong and fast oscillation implies correlation in the micro-structures. Therefore, if our signal $f(t)$ represents values in every single moment of time, its Fourier transform $F(\omega)$ represents the strength of every oscillation in a holistic way in that chunk of Z time. These two signals are related to each other by the following formula:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t}$$

For simplicity, in the following we will consider a signal that is constructed by summing different sine waves and try to get information about the underlying frequencies.

1. Construct a 20 seconds long signal samples at 1kHz by adding 20 sine waves with a frequency randomly choosen from the interval [1Hz, 300Hz] (no phase shift is necessary). Add some gaussian noise to the signal.
2. Plot the signal in the interval t = 0, . . . , 2s.
3. Use numpy to calculate the one-dimensional discrete Fourier Transform.
4. The square of the absolute value of the fourier transform gives you the power carried by each frequency (power spectrum). To calculate the Fourier transform sample frequencies you can also use a numpy function - have a look at the documentation of the `numpy.fft` module. Plot the power spectrum logarithmically and also indicate the frequencies that were used for generating the data.
5. *** Add a plot of the original time series to the plot of the power spectrum. To generate the subplot you can add an axes object directly to the figure. If you don't know how to do that have a look at the documentation of matplotlib or the examples in the gallery (http://matplotlib.sourceforge.net/gallery.html).***

4.4.1 Solution

In [7]:

4.5 Vectorization of a Monte-Carlo Simulation

Recall the dice-simulation from the basic-exercise sheet. The goal of the current exercise is to optimize this program using vectorization.

1. Create a function that draws N-times two uniform random integers from 1 to 6 and counts how many times X at least one of the two takes the value 6. Do this in a single N × 2 matrix.
2. Run this simulation for growing N and plot the estimated probability as a function of N. Try to use linear and logarithmic axis. Include the exact value of 11/36.
3. Repeat the dice throwing Z times for a fixed value of N and plot a histogram of the number of throws with at least one six (e.g. for N = 1000, Z = 10000). The data are integers, so choose a bin width of 1 or use integer functions such as `np.bincount`. Further print out the mean value and the standard deviation.
4. As above repeat the dice throwing for a fixed number N of dice throws Z times and calculate the probability to get at least one six with the two dices. From this data you can calculate the empirical average and standard deviation for every pair of N and Z. Make an error-bar plot for N = 1, . . . , 20 and Z = 5000.

4.5.1 Solution

In [7]:

5 Integration

5.1 Lotka-Volterra System

Consider the Lotka-Volterra equation of predator-prey interactions

$$\frac{\mathrm{d}N_1}{\mathrm{d}t} = N_1 \cdot (\varepsilon_1 - \gamma_1 N_2), \frac{\mathrm{d}N_2}{\mathrm{d}t} = -N_2 \cdot (\varepsilon_2 - \gamma_2 N_1)$$

This is a system of ordinary differential equations, where $N_1(t)$ is the number of preys and $N_2(t)$ the number of predators. $\varepsilon_1, \varepsilon_2, \gamma_1, \gamma_2$ are parameters representing the growth and interaction between preys and predators.

1. Find the fixed points of the system for the parameters $\varepsilon_1 = 1.0, \varepsilon_2 = 1.5, \gamma_1 = 0.1, \gamma_2 = 0.075$.
   - Define a function that returns the growing rates $\frac{\mathrm{d}N_1}{\mathrm{d}t}$ and $\frac{\mathrm{d}N_2}{\mathrm{d}t}$ in an array. In the second part of the exercise, we will use `scipy.integrate.odeint()` to obtain the solution of the ODE system. Therefore, choose the parameters of your function accordingly, $f = f(N, t, \dots)$.
   - Use `scipy.optimize.fsolve` to find the fixed points of the system.
2. Solve the system of equations for the following initial conditions $N_1(0) = 10,\ N_2(0) = 5$.
   - Define a vector containing the time steps of the integration and one for the initial conditions.
   - Solve the system using `scipy.integrate.odeint()`
   - Lower the precision of the integrator until you see a difference after many periods. The integrator has multiple options for precision control. Also the number of steps can have an effect on the result.
3. *Bonus (no solution)*: Also try `scipy.integrate.ode` which provides different integrators but requires manual looping (this is not a big speed issue, integrating in python is pretty slow in any case).

5.1.1 Solution

In [7]: