Python-Exercises
NumPy, SciPy and Matplotlib

October 9, 2012

1 Matrix manipulations

By using miscellaneous constructors, indexing, slicing, and simple operations $(+, -, *, :)$, large arrays with various patterns can be created.

1. Create the following arrays with the simplest solution:

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

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

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

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

   and generate a new array containing only its 2nd and 4th rows.

3. Divide each column of the array

```
a = np.arange(25).reshape(5, 5)
```

elementwise with the array

`b = np.array([1., 5, 10, 15, 20]).`

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

   Helpful functions: `numpy.abs`, `numpy.random.rand`, `numpy.argsort`, `numpy.reshape` `numpy.diag`

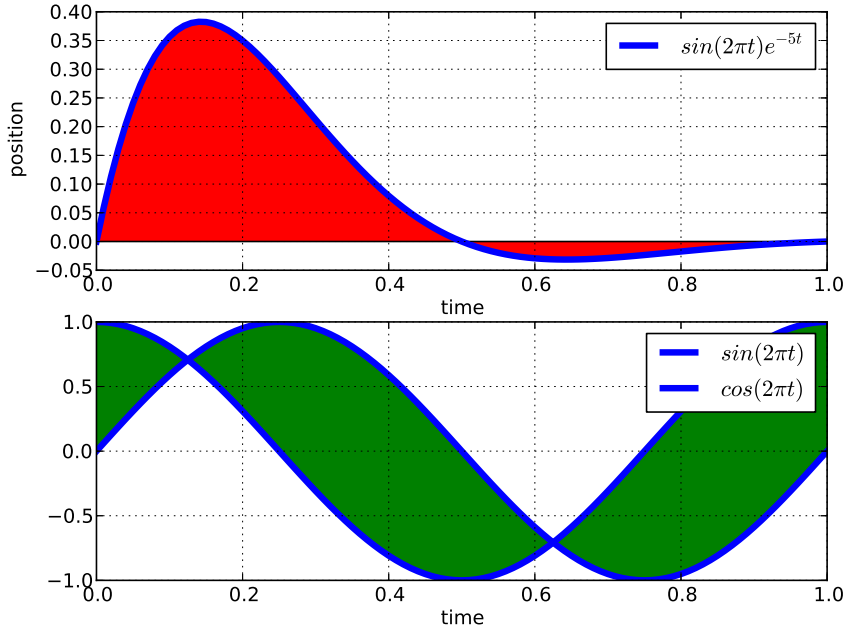
2 Simple plotting

1. Plot a simple graph of the sine function in the range 0 to 3 with a step size of 0.01.
2. Make the line red. Add diamond-shaped markers with size of 5.
3. Add a legend and a grid to the plot.

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

3 Simple plotting II

1. Try to recreate the following plot.



Helpful functions: `pylab.grid`, `pylab.plot`, `pylab.legend`, `pylab.fill`, `pylab.fill_between`

4 Polynomial Fitting (polyfit)

1. Generate data by a polynomial function $f(x) = -3x^3 + 2x - 8$
2. Fit a polynomial (**scipy.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)

5 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.

6 Intergration

1. Define the function $f(x) = e^{-(x+3)^2}$
2. Vectorize the function by using **np.vectorize()** such that the function also accepts lists of xvalues.
3. Calculate the integral $\int_{-20}^{20} f(x)\mathrm{d}x$ by hand, using Riemann sums.
4. Use **scipy.integrate.quad()** to evaluate the integral.
5. Try to calculate $\int_1^\infty \frac{1}{x}\mathrm{d}x$ and $\int_1^\infty \frac{1}{x^2}\mathrm{d}x$

7 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()** to solve the system of equations.
3. Verify the results by using **numpy.dot()**.

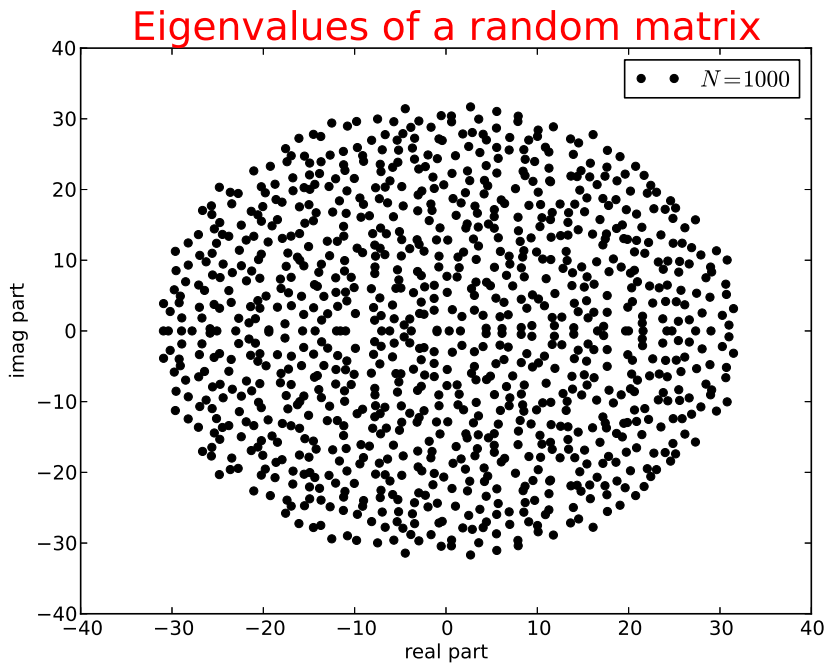
8 Some linear algebra

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.

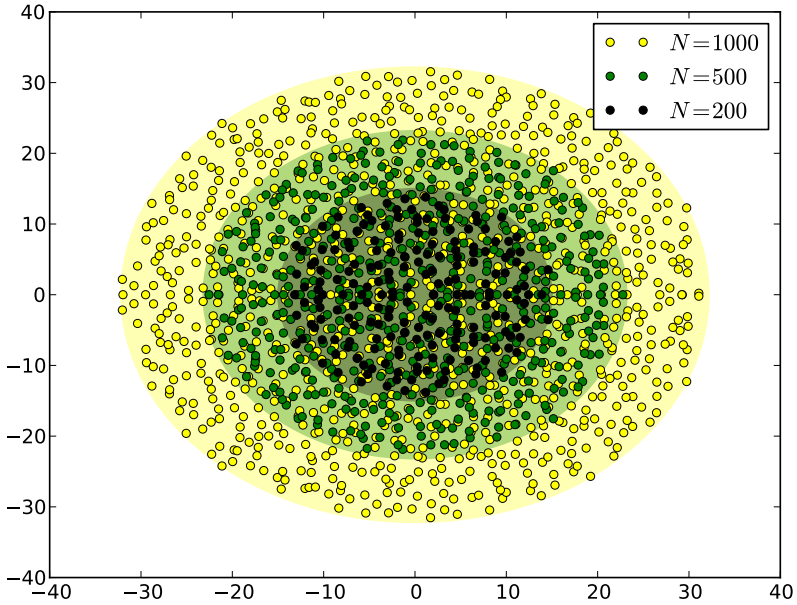
4. Store your graph as an `.eps` file:



5. Define a function that returns `True` if a point (x, y) is contained in a circle of radius r around the origin.

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 module. 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)



Helpful functions: `scipy.linalg.eigenvals`, `numpy.real`, `numpy.imag`

9 Calculating a Histogram

1. Take the histogram function `histgram( data, numbins )` from the basic-exercises sheet 2.
   Modify the function, such that its output is now a tuple of numpy arrays. it returns an array containing the left border of each bin, an array with the number of data points in each bin and the bin width.

2. Use this function to create a graphical histogram using `pylab.bar()`. Compare your results with the plot generated by `pylab.histogram()`.

10 1D Arrays and Plotting

1. Plot a 1D Gaussian probability density function with $\mu = 10$ and $\sigma = 5$ using the formula
   $$G(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$$
   Choose a suitable x-range.

2. Use a numerical integration method to calculate the integral of $G(x)$ on the interval $[\mu - 50, \mu + 50]$ and verify that it is approximately equal to 1

3. Integrate the function with varying window sizes w_i, such that the integration is on the interval $[\mu - w_i, \mu + w_i]$. Plot the integral as a function of the used window-size.

Helpful functions: `numpy.arange, pylab.plot, scipy.integrate.quad`

11 2D Arrays and Plotting

1. Calculate a bivariate standard normal density

$$G_2(x, y) = \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-(x^2 + y^2)}{2}\right)$$

   on the interval $[-3, 3]$ and save it to a matrix `G`.

2. Plot the result in an image with a colorbar-legend.

3. The bivariate gaussian has a hat-like shape. Use index-slicing to get 4 separate, nonoverlapping matrices `Q1,Q2,Q3,Q4` (quadrants) of equal dimensions that cover the whole matrix `G`. Create a new matrix `G2` in which `Q1` and `Q3` changed place and `Q2` $\leftrightarrow$ `Q4`. Plot this result as well.

4. Plot the difference of `G` and `G2`.

   Functions to use: `numpy.arange`, `numpy.meshgrid`, `pylab.imshow`, `pylab.colorbar`, `numpy.empty`

12 Calculate a Power spectrum

Away 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 time. These two signals are related to each other by the following formula: $F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt$.
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.

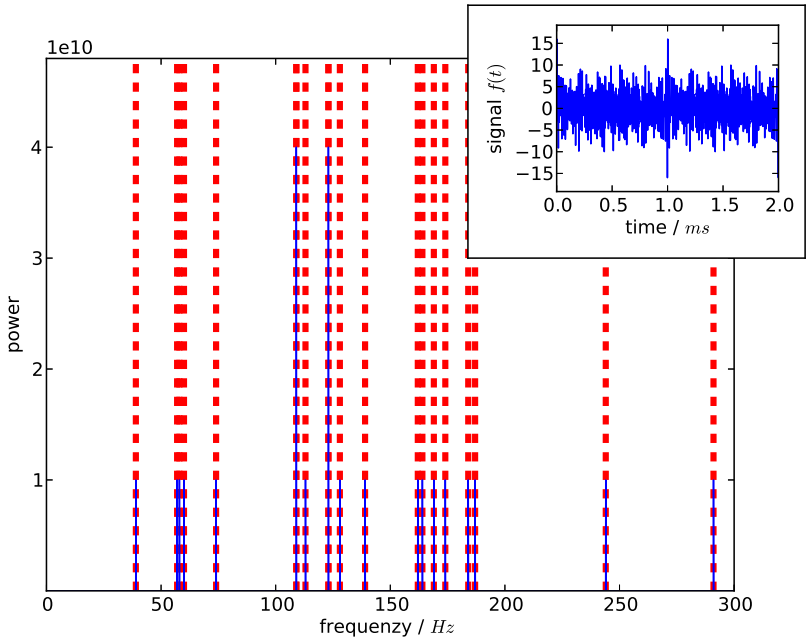
1. Construct a signal by adding 20 sine waves with a frequency randomly choosen from the interval $[1Hz, 300Hz]$.

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 numpy function - have a look at the documentation of the numpy.fft module. Plot the powerspectrum 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).



13 Vectorization of 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 script that N-times draws 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 $N \times 2$ matrix. Write out the estimated probability X/N together with the exact result 11/36.

2. Run this simulation for growing N and plot the estimated probability as a function of N. Try to use linear and logarithmic axis.

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 $N = 1000$, $Z = 10000$).

4. The data are integers, so choose a bin width of 1. Further print out the mean value and the standard deviation.

5. 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 $N = 1, \ldots, 20$ and $Z = 5000$.

   Helpful functions: `numpy.random.randint`, `numpy.sum`, `numpy.vectorize`, `numpy.mean` `numpy.std`, `pylab.plot`, `pylab.semilogx`, `pylab.hist`

14 ODE integration

Recall the Neuron Class from last week. There you approximated the time derivative of the membrane potential by

$$\dot{V} \approx \frac{V(t+h) - V(t)}{h} \ .$$

1. Now use the methods from `numpy.integrate` to provide a more accurate integral of the equation determining the membrane potential

$$c_M \dot{V} = -g_L \left(V - V_L\right) + i_{ext}$$

15 Lotka-Voltera System

Consider the Lotka-Voltera equation of predator-prey interactions

$\frac{\mathrm{d}N_1}{\mathrm{d}t} = N_1(\epsilon_1 - \gamma_1 N_2)$,
$\frac{\mathrm{d}N_2}{\mathrm{d}t} = -N_2(\epsilon_2 - \gamma_2 N_2)$

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. ϵ_1, ϵ_2, γ_1, γ_2 are parameters representing the growth and interaction between preys and predators.

1. Find the fixed points of the system for the following parameters
   $\epsilon_1 = 1.0,\ \epsilon_2 = 1.5,\ \gamma_1 = 0.1,\ \gamma_2 = 0.075$
   - Define a function that returns the growing rates $\frac{\mathrm{d}N_1}{\mathrm{d}t}$ and $\frac{\mathrm{d}N_1}{\mathrm{d}t}$ 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, $f = f(N, t, ...)$.
   - 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()**.
   - Plot the results for two different time step sizes.