1 Exercises on numpy array

1.1 Squared numbers

Create a list (using a loop) and an array (without loop) containing the square of integer number from 0 to N-1. Compare execution speed with $N = 10^6$.

Solution

On can use the time module te precisely measure the speed

```
from time import time
from numpy import *

t0 = time()

list_square = []
for i in range(1000000):
    list_square.append(i**2)

t1 = time()

array_square = arange(1000000, dtype=int64)**2

t2 = time()

print("Duration with a list   :", t1-t0)
print("Duration with an array :", t2-t1)
```

1.2 Calculation of pi

Without any loop, calculates π using the formula :

$$\pi = \sqrt{12}\sum_{k=0}^{\infty}\frac{(-1)^k}{3^k(2k+1)}$$

Take care of the integer division in the 2.7 version of Python.

Solution

```
N = 34
Tk = arange(N, dtype=float64)

print(sqrt(12)*sum((-1)**Tk / (3**Tk*(2*Tk+1))))
```

Which value to choose for N ? The relative precision of 64 bits float is 2×10^{-16}. We can stop the sum if $1/(3^k(2k+1)) < 3 \cdot 10^{-16}$. This is the case for $k > -\log(3 \cdot 10^{-16})/\log(3) \simeq 34$.

1.3 Allan variance

The allan variance of a set of measurements y_k, where each point corresponds to a frequency measured during τ is defined as :

$$\sigma_y^2(\tau) = \frac{1}{2}\left\langle (y_{k+1} - y_k)^2 \right\rangle_k$$

1. Write a function that calculates the Allan variance **without any loop**

Using a data set where the duration of each measurement is τ_0, it is possible to calculate the Allan variance for duration $\tau = n\tau_0$ that are mutiple of τ_0. In this case, the new value is calculated by taking the average value of n consecutive measurements. There is no overlap between the values (there is n times less points compared to the initial set).

2. Write a function `average_frequency(data,n)` that calculate the mean value of `data` with packets of size `n`.

It is possible to make this function without any loop. Let us start with an array of size 10 and take $n = 2$. The shape of the array is initially:

```
+--+--+--+--+--+--+--+--+--+--+
|x0|x1|x2|x3|x4|x5|x6|x7|x8|x9|
+--+--+--+--+--+--+--+--+--+--+
```

Using the rechape method (`x.reshape((5,2))`), the array will look like:

```
+--+--+
|x0|x1|
+--+--+
|x2|x3|
+--+--+
|x4|x5|
+--+--+
|x6|x7|
+--+--+
|x8|x9|
+--+--+
```

The method `mean(axis=1)` will perform the average line by line.

3. Write the function `AllanVariance(data,n)` that calculate the Allan variance for a measurement duration of $n\tau_0$.

Solution

```python
def average_frequency(data, n):
    """ moyenne les valeurs de data par paquets de taille n"""
    databis = data[len(data)%n:]
    return databis.reshape(len(data)//n, n).mean(axis=1)

def AllanVariance(data, n):
    """Variance d'Allan

    Calcul la variance d'Allan du tableau data en regroupant les mesures par
    paquets de taille n
    """
```

```
    y = average_frequency(data, n)
    return  ((y[1:] - y[:-1])**2).mean() /2
```

1.4 Mandelbrot set

The Mandelbrot set is defined as the set of points c of the complex plane such that the following sequence :

$$z_0 = 0$$
$$z_{n+1} = z_n^2 + c$$

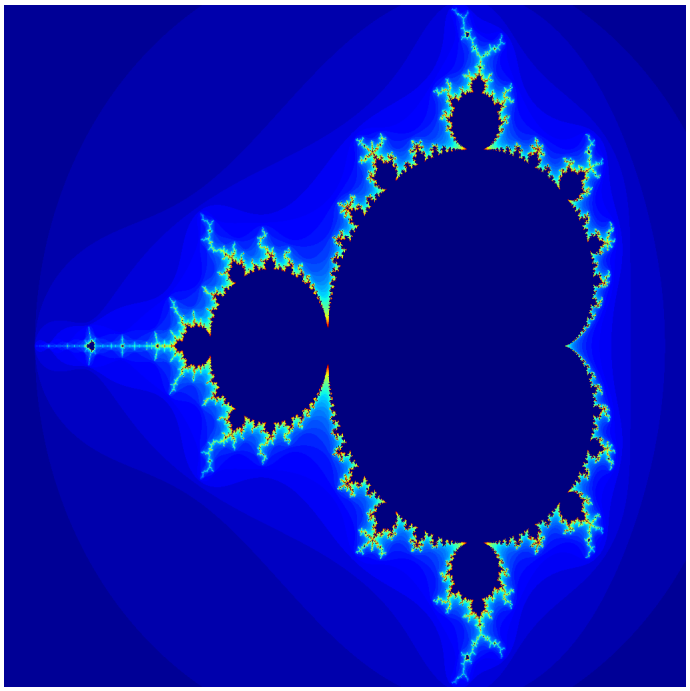
is bounded.

One can show that if there is a value of n such that $|z_n| > 2$ the the sequence is divergent. To calculate the Mandelbrot set, for each value of c one calculate 100 iterations. The picture is plotted by giving to each point the smallest value of $n < 100$ such that $|z_n| > 2$ (and 0 if this value doesn't exist).

Calculate and plot the Mandelbrot set for c such that $-2.13 < \mathrm{Re}(c) < 0.77$ and $-1.13 < \mathrm{Im}(c) < 1.13$. One can then zoom on the edge of the set.

Note : to plot an image, use the imshow function of pylab.

The total computation time using numpy efficiently is less than 10 seconds for a 1024x1024 matrix.



Solution

Four different method are used : using numpy functions, using the vectorize function and using the numexpr package (multi CPU).

```
x_min, x_max = -2.13, 0.77
y_min, y_max = -1.13, 1.13

N = 1024

def mandel_numpy(x_min, x_max, y_min, y_max, N):
```

```
    """ Mandelbrot using numpy """
    x = linspace(x_min, x_max, N)
    y = linspace(y_min, y_max, N)*1j

    c = x + y[:,None]

    image = zeros((N,N))

    z = 0
    for i in range(100):
        z = z**2 + c
        ind = (abs(z)>2) & (image==0)
        image[ind] = i

    return image

def _mandel_suite(c):
    z = 0
    for i in range(100):
        z = z**2 + c
        if abs(z)>2:
            return i
    return 0

mandel_suite = vectorize(_mandel_suite)

def mandel_vectorized(x_min, x_max, y_min, y_max, N):
    """ Mandelbrot set using vectorize from numpy """
    x = linspace(x_min, x_max, N)
    y = linspace(y_min, y_max, N)*1j

    c = x + y[:,None]

    return mandel_suite(c)

import numexpr as ne
def mandel_numexpr(x_min, x_max, y_min, y_max, N):
    """ Mandelbrot set using the numexpr package"""
    x = linspace(x_min, x_max, N)
    y = linspace(y_min, y_max, N)*1j

    c = x + y[:,None]

    image = zeros((N,N))

    z = 0
    for i in range(100):
        z = ne.evaluate('z**2 + c')
        ind = ne.evaluate('(z.real**2+z.imag**2>4) & (image==0)')
        image[ind] = i

    return image

import numba
```

```
@numba.vectorize
def _mandel_numba(c, N_iter_max=100):
    z = 0
    for n in range(N_iter_max):
        if abs(z)>2:
            return n
        z = z**2 + c
    return 0

def mandel_numba(x_min, x_max, y_min, y_max, N):
    """ Mandelbrot set using vectorize from numpy """
    x = linspace(x_min, x_max, N)
    y = linspace(y_min, y_max, N)*1j

    c = x + y[:,None]

    return _mandel_numba(c, 100)

t0 = time()
mandel_numpy(x_min, x_max, y_min, y_max, N)
print("Numpy :", time()-t0)
t0 = time()
mandel_vectorized(x_min, x_max, y_min, y_max, N)
print("Vectorize :", time()-t0)
t0 = time()
mandel_numexpr(x_min, x_max, y_min, y_max, N)
print("Numexpr :", time()-t0)
t0 = time()
mandel_numba(x_min, x_max, y_min, y_max, N)
print("Numba :", time()-t0)
```

Benchmark on a Intel(R) Core(TM) i5-3570K CPU @ 3.40GHz Numpy : 2.36773109436 Vectorize : 6.50940179825 Numexpr : 0.899597883224

---

2 Exercises on graphics (and numpy)

2.1 Measurement of pi (Monte Carlo)

This exercise can be solved without any loop.

- Using the `rand` function, creates two array X and Y with `N=1000` random samples from a uniform distribution over `[-1,1[`.
- Plot the points
- Plot a circle of radius 1 and plot using a different color the points inside the circle.
- How many points are inside the circle ?
- The number of points is proportional to the area of the circle. Deduce an approximate value of π. One can use 10^6 points (without plotting the figure...)

Solution

```
from numpy import *
from pylab import *
ion()

N = 1000
X = 2*rand(N)-1
Y = 2*rand(N)-1

figure(1)
clf()
plot(X,Y, '.')
cond = (X**2 + Y**2)<1
plot(X[cond],Y[cond], '.')
theta = linspace(0, 2*pi, 201)
plot(sin(theta), cos(theta))

def mesure_pi(N):
    X = 2*rand(N)-1
    Y = 2*rand(N)-1
    Nb_points = sum((X**2 + Y**2)<1)
    return 4*Nb_points/float(N)

print(mesure_pi(1000000))
```

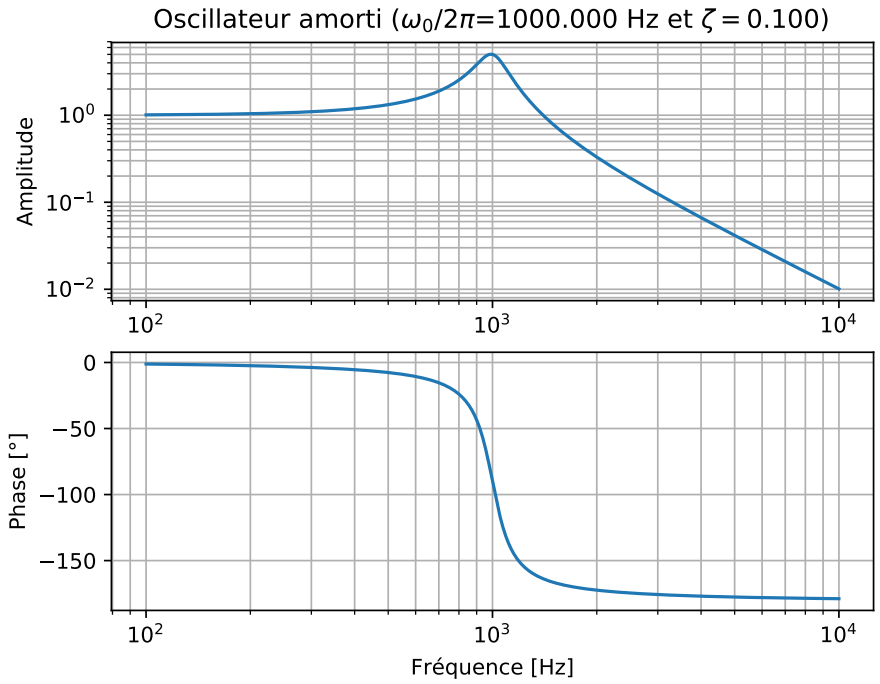
2.2 Bode plot

The transfer function of a damped oscillator can written in the Fourier space using the formula :

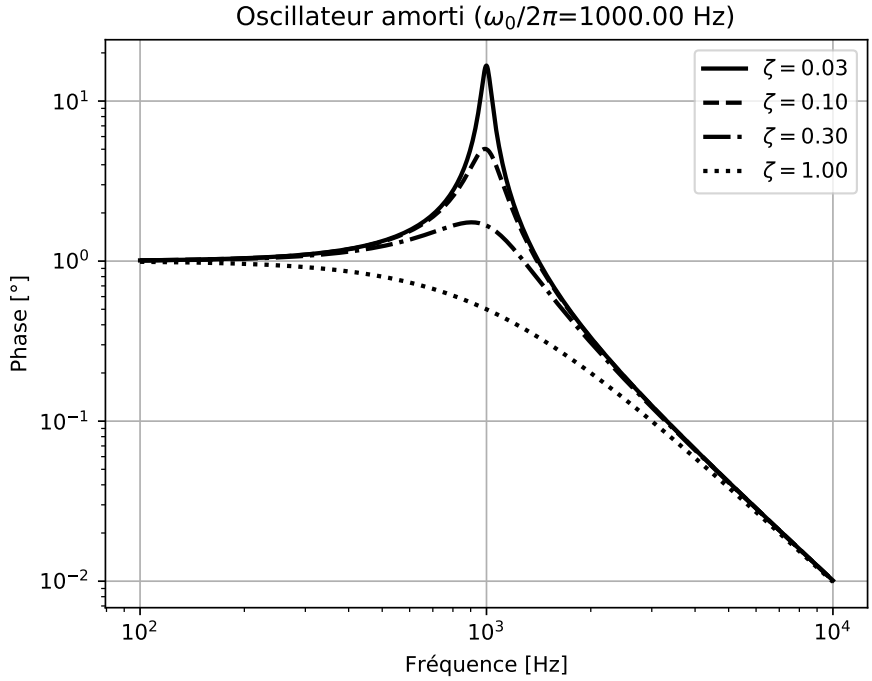
$$H(\omega) = \frac{\omega_0^2}{\omega_0^2 - \omega^2 + 2j\zeta\omega\omega_0}$$

We would like to draw the three following graphs :

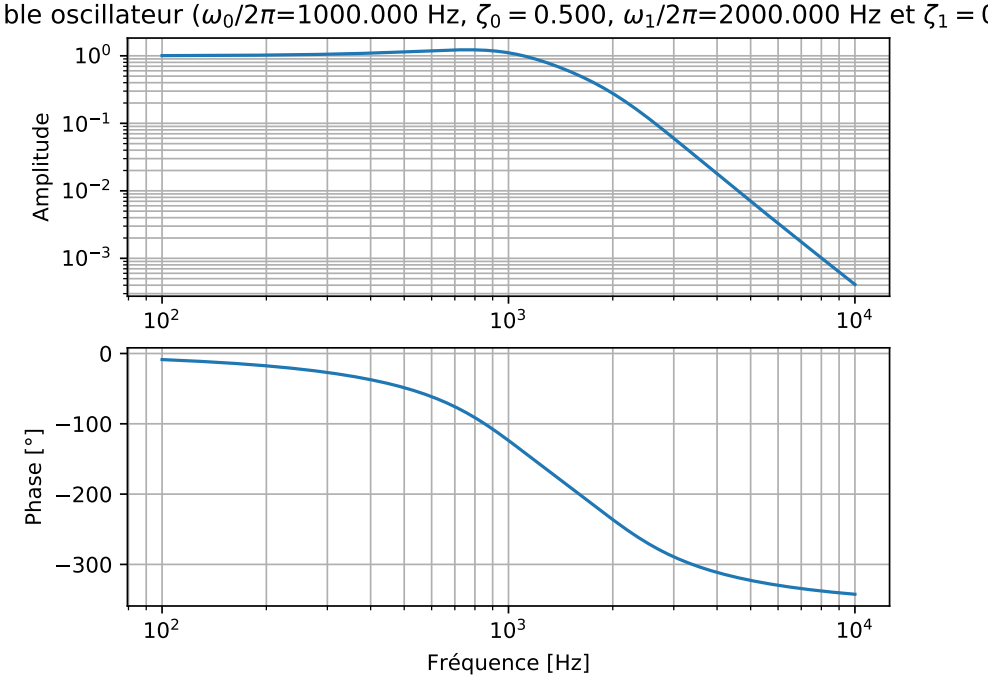
- Bode plot for $\omega_0/2\pi = 1\text{kHz}$ and $\zeta = 0.1$



- Tranfer amplitude function for $\omega_0/2\pi = 1\text{kHz}$ and different values of ζ.



- Bode plot for the product of two tranfer function $\omega_0/2\pi = 1\text{kHz}$, $\zeta = 0.5$, $\omega_1/2\pi = 2\text{kHz}$, $\zeta = 0.5$.



You will uses the following functions :

- `loglog`
- `semilogx`
- `xlabel`, `ylabel` et `title`
- `grid`
- `subplot(ny,nx,n)`

- The `label` optional parameter and the function `legend`.
- The `angle` function of numpy can be used to calculate the phase of a complex number. For the last graph, one should modify the phase in order for the plot to be continuous (do it without any loop!).

You should also know how to

- make a string with accents (for french labels!).
- format a string to insert parameters
- For the greek letters, one can use unicode or latex formula.

Solution

```
from pylab import *

def H(omega, omega_0, zeta):
    return omega_0**2/(omega_0**2-omega**2 + 2J*omega*zeta*omega_0)

omega_0 = 2*pi*1000
zeta = 0.1

Tomega = 2*pi*10**(linspace(2,4, 1001))
amplitude = H(Tomega, omega_0, zeta)

figure("exo1.1")
clf()
subplot(2,1,1)
title_str = 'Oscillateur amorti ($\omega_0/2\pi$={0} Hz et $\zeta={1}$)'
title(title_str.format(omega_0/(2*pi),zeta))
grid(True, which='both')
loglog(Tomega/(2*pi), abs(amplitude))
ylabel('Amplitude')
subplot(2,1,2)
semilogx(Tomega/(2*pi), angle(amplitude)/(2*pi)*360)
ylabel(u'Phase [°]')
xlabel(u'Fréquence [Hz]')
grid(True, which='both')

savefig('exo1_1.pdf')
savefig('exo1_1.png')

Tzeta = [.03,0.1,0.3,1]
Tlinetype = ['-', '--', '-.', ':']
figure("exo1.2")
clf()
for zeta, linetype in zip(Tzeta, Tlinetype):
    amplitude = H(Tomega, omega_0, zeta)
    loglog(Tomega/(2*pi), abs(amplitude), 'k'+linetype,
    label="$\zeta={0:4.2f}$".format(zeta), linewidth=2)
grid(True)
xlabel(u'Fréquence [Hz]')
ylabel(u'Phase [°]')
legend()
title_str = 'Oscillateur amorti ($\omega_0/2\pi$={0} Hz)'
title(title_str.format(omega_0/(2*pi)))
```

```
savefig('exo1_2.pdf')
savefig('exo1_2.png')

omega_0 = 2*pi*1000
omega_1 = 2*pi*2000

zeta_0 = 0.5
zeta_1 = 0.5

Tomega = 2*pi*10**(linspace(2,4, 1001))
amplitude = H(Tomega, omega_0, zeta_0)*H(Tomega, omega_1, zeta_1)

figure("exo1.3")
clf()
subplot(2,1,1)
title_str='Double oscillateur ($\omega_0/2\pi$={0} Hz,\
 $\zeta_0={1}$, $\omega_1/2\pi$={2} Hz et $\zeta_1={3}$)'
title(title_str.format(omega_0/(2*pi),zeta_0, omega_1/(2*pi), zeta_1))
grid(True, which='both')
loglog(Tomega/(2*pi), abs(amplitude))
ylabel('Amplitude')
subplot(2,1,2)

phase = angle(amplitude)
phase[phase>0] -= 2*pi
semilogx(Tomega/(2*pi), phase/(2*pi)*360)
ylabel(u'Phase [°]')
xlabel(u'Fréquence [Hz]')
grid(True, which='both')
savefig('exo1_3.pdf')
savefig('exo1_3.png')
```

---

3 Exercises on fit

3.1 Fit of inteference fringes

We would like to fit the fringes from an atom interferometer. The data are in the file (`data/fit_sinus.dat`). The first column of the file (x axis) represents a frequency in Hz. The second column (y axis) represents the population measured for the given frequency. The goal is to find the position of the central fringe.

The fit function is a cosine function with adjustable offset, amplitude, position and width.

- Load and plot the data.
- Write the fit function called `fringe(x,....)`
- Find the initial parameter for the fit.
- Calculate the optimal parameters
- What is the position and uncertainty of the central fringe ?

Solution

```
from numpy import *
from pylab import *
ion()

from scipy.optimize import curve_fit

data = loadtxt('data/fit_sinus.dat')

Freq = data[:,0]
Y = data[:,1]

def fringe(x,offset, amplitude, center, Delta_f):
    "y = offset + amplitude* (1+cos(2*pi*(x-center)/Delta_f))/2"
    return offset + amplitude* (1+cos(2*pi*(x-center)/Delta_f))/2

param_ini = [0.12, .2, 0., 200.]

clf()
plot(Freq, Y, 'o')

x_fit = linspace(min(Freq), max(Freq))
#plot(x_fit, fringe(x_fit, *param_ini))

popt, cov = curve_fit(fringe, Freq, Y, p0=param_ini)

offset, amplitude, center, Delta_f = popt
sigma_offset, sigma_amplitude, sigma_center, sigma_Delta_f = sqrt(diag(cov))

plot(x_fit, fringe(x_fit, *popt))
```

3.2 Fit of a picture

We have a 64x64 picture of a double star. Using the example given in the lecture, fit the picture by the sum of two gaussians and get the distance between the two stars.

Data are in the file (`data/double_star`).

Solution

```
image = loadtxt('data/double_star.txt')

ny, nx = image.shape
X,Y = meshgrid(range(nx), range(ny))
# two column matrices with X and Y
XY = array([X.flatten(), Y.flatten()]).transpose()

def gauss(XY, amplitude, center_x, center_y, diameter):
    x = XY[:,0]
    y = XY[:,1]
    return amplitude*exp(-((x-center_x)**2 + (y-center_y)**2)/diameter**2)

def model(XY, *param):
    return gauss(XY, *param[:4]) + gauss(XY, *param[4:8]) + param[8]
```

```
figure(0)
imshow(image, interpolation = 'nearest')

# Measured position of the center using the imshow figure.
x0, y0 = 31, 25
x1, y1 = 31, 38

p0 = [200, x0, y0, 2, 200, x1, y1, 2, 0.2]

# One can look at the initial parameters
image_test = model(XY, *p0).reshape(X.shape)
imshow(image_test, interpolation = 'nearest')

popt, pcov = curve_fit(model, XY, image.flatten(), p0)
print("Distance : ", hypot(popt[1] - popt[5], popt[2] - popt[6]))

figure(1)
image_fit = model(XY, *popt).reshape(X.shape)
imshow(image_fit, interpolation = 'nearest')
```

---

4 Exercises on differential equation

4.1 The pendulum equation

We consider the equation

$$\theta'' = -\sin\theta$$

Write a function that returns an array containing the position and the angular velocity of the pendulum for N instants t_i between 0 and T.

The initial position is $\theta = 0$. Plot the phase space trajectory for different values of the initial velocity. Angle will be represented between -π and π.

Solution

```
from scipy.integrate import ode
from numpy import *
from pylab import *
ion()

def f(t, Y):
    y, yprime=Y
    return array([yprime, -sin(y)])

def pendule(theta_0, vit_ang, T, N=1000):
    r = ode(f).set_integrator('vode')
    r.set_initial_value(array([theta_0, vit_ang]),0)
    TT = linspace(0,T,N+1)
    output = [[0, theta_0, vit_ang]] # Output is a list of list
    for i, t in enumerate(TT[1:]):
        r.integrate(t)
```

```
        output.append([t, r.y[0], r.y[1]])
    return array(output) # This is a 2D array

figure(1)
clf()
Tvit_ini = linspace(-3,3, 61)
for vit_ini in Tvit_ini:
    a = pendule(0, vit_ini, 100)
    a[:,1] = ((a[:,1]+pi)%(2*pi) - pi) # Tricks to be between -pi and pi
    plot(a[:,1], a[:,2], '+')

xlim(-pi,pi)
```

4.2 Solving the Schrödinger equation using the finite element method

Let us consider the Schrödinger equation with hbar = m = 1

$$\frac{d\psi}{dt} = -i\left(-\frac{1}{2}\frac{d^2\psi}{dx^2} + V(x)\psi\right)$$

The potential is $V(x) = \frac{1}{2}kx^2$ with $\kappa = .5$.

To solve the equation, we will truncate the x-axis to values between $x_{\min}$ and $x_{\max}$. We will also discretize the x-axis with small steps (Δx).

The term $\frac{d^2\psi}{dx^2}$ will be approximated using $\frac{\psi(x+\Delta x)-2\psi(x)+\psi(x-\Delta x)}{\Delta x^2}$.

For the initial state, we will take a Gaussian distribution $e^{-\alpha(x-x_0)^2}$. We will use $\alpha = \frac{1}{2}$ and $x_0 = 1$.

Calculate and plot $|\psi(x,t)|^2$ as a function of x for $t = 1$ using the `zvode` solver with an absolute precision of 10^-3.

Solution

```
from scipy.integrate import ode
from numpy import *

Deltax = 0.1
X_MIN = -10
X_MAX = 10

alpha = .4
x0 = 1

kappa = .5
T = 10
N = 1000

x = linspace(X_MIN, X_MAX, (X_MAX-X_MIN)/Deltax + 1)

psi_0 = exp(-alpha*(x-x0)**2)
psi_0 = psi_0 / sqrt(sum(abs(psi_0**2)))

V = -.5*kappa*x**2
```

```
psi_dot = zeros(len(x), dtype="complex128")

def f(t, psi):
    energy = .5*(psi[2:] - 2*psi[1:-1] + psi[:-2])/Deltax**2
    pot = V[1:-1]*psi[1:-1]
    psi_dot[1:-1] = -1j*(energy + pot)
    return psi_dot

r = ode(f).set_integrator('zvode', atol=1E-3)
r.set_initial_value(psi_0,0)

psi_f = r.integrate(1)

figure(0)
plot(x, abs(psi_f)**2)

# Calculate the mean position and relative width.
figure(1)
r = ode(f).set_integrator('zvode', atol=1E-3)
r.set_initial_value(psi_0,0)

dT = linspace(0,T,N+1)
out = []
for i, t in enumerate(dT):
    if t>r.t:
        r.integrate(t)
    print(i)
    out.append([t, sum(x*abs(r.y)**2)/sum(abs(r.y)**2), (mean(x**2*abs(r.
↪y)**2) - mean(x*abs(r.y)**2)**2)/mean(abs(r.y)**2)])
```

5 Exercises on Fourier analysis

5.1 Simple example of Fourier transform

Plot the Fourier transform of $1/(1+0.99\cos(2\pi t))$ for frequency below 50 Hz. using the *fft* function of the module *numpy.fft*.

- Which sample rate do you use ?
- On which duration should you calculate f ?
- What happens if this duration is 10 time too long ?
- What happens if it is not a multiple of the period of the signal ?

Solution

The sample rate should be at least 100 sample/s f is periodic. It should be calculated over one period

```
from numpy import *
from numpy.fft import *
from pylab import *
ion()
```

```
def function(t):
    return 1/ (1+.99 * cos(2*pi*t))

Taux = 100 # Sample rate
T = 1. # duration
N = int(Taux * T) # number of points

X = linspace(0,T,N, endpoint=False)
Y = function(X)

figure('exo 1A')
clf()
plot(X,Y)
Ytilde = fft(Y)

Freq = arange(N)
# The frequency is defined modulo N.
# We take values between -N/2 and N/2
Freq[Freq>=N/2] = Freq[Freq>=N/2] - N
Freq = Freq/T

figure('exo 1B')
clf()
loglog(Freq[1:N//2-1], abs(Ytilde)[1:N//2-1])
```

If the duration is 10 times the period, intermediate frequencies will have a zero amplitude.

The discrete fourier transform calculates the continuous Fourier transform of the function which is extrapolated by periodic juxtaposition of the initial one. It is the initial function only if it was truncated to a multiple of the initial period. If this not done, then we do not obtain the right Fourier transform

---

5.2 Power spectral density

The power spectral density (PSD) is defined as the square of the modulus of the Fourier transform of a signal divided by the integration time. It is defined only for positive frequencies.

$$S(\omega) = \frac{1}{T}\left(\int_{-T/2}^{T/2} f(t)e^{-i\omega t}\mathrm{d}t\right)^2$$

- Write a function in Python that calculates the PSD for a given signal (numpy array). This function will return two arrays : the frequencies and the corresponding PSD.
  - What is the unit of the PSD ?
  - Which parameter should we add in order to get the correct freuqencies ?
  - What is the minimal (non zero) and maximal frequency ?

If you don't want to solve this question, use the `periodogram` function of the `scipy.signal` module.

Solution

If A is the unit of the signal, the DSP unit is A^2T, where T is the unit if time. We need the sample rate. The minimal frequency is 1/T. The maximal is N/(2T)

```
def DSP(data, sample_rate):
    """ Calculate the power spectral density

    output : (freq, PSD)

    """
    N = len(data)
    T = N/float(sample_rate)
    freq = arange(N)/float(T)
    dsp = abs(fft(data))**2/(N*sample_rate)
    return freq[:N//2], dsp[:N//2]
```

5.3 Sound analysis

In the given file, I have recorded the noise of my washing machine (`son/machine_a_laver.wav`).

- Load the file and get its sample rate using the function `read` from the `scipy.io.wavfile` module.

Solution

```
from scipy.io.wavfile import read
samplerate, amplitude = read('son/machine_a_laver.wav')
```

- Plot the PSD of the noise. What is the frequency of the rotation of the machine ?

Solution

There is a peak at 20 Hz.
