Numpy et matplotlib

oct. 23, 2019

1 Diagrames de Bode

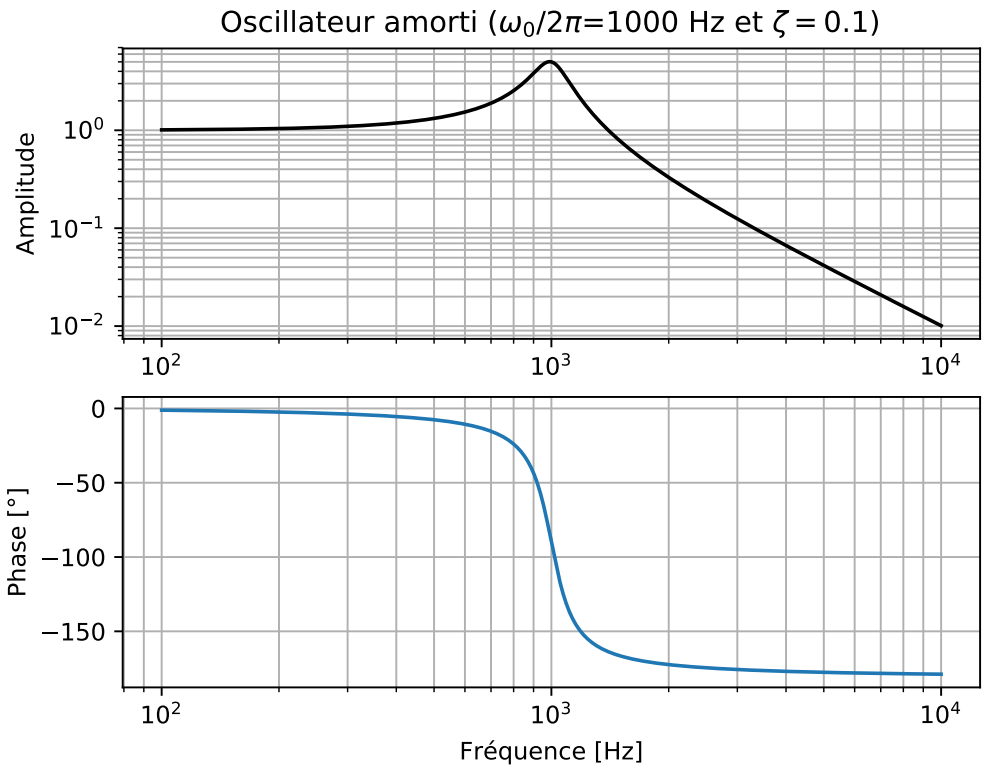
Note : L'objectif de cet exercice est de faire un joli graphique (comme pour une publication).

On considère un oscillateur amorti (mécanique ou électrique). La fonction de transfert de cet oscillateur s'écrit dans l'espace de Fourier de la façon suivante :

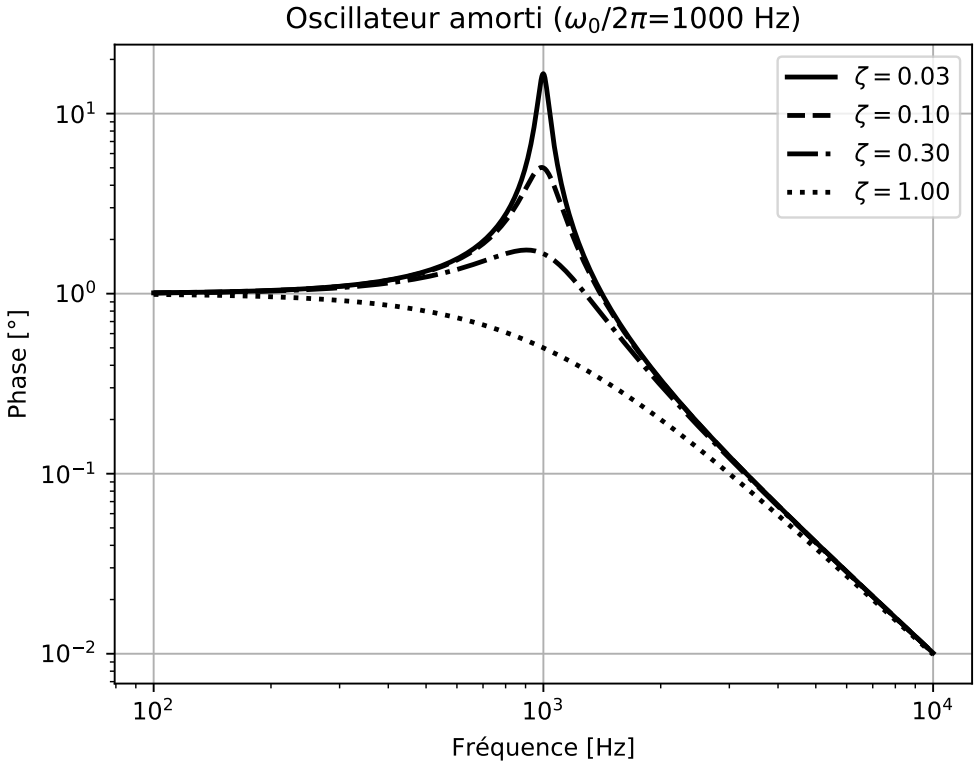
$$H(\omega) = \frac{\omega_0^2}{\omega_0^2 - \omega^2 + 2j\zeta\omega\omega_0}$$

Voici les deux graphs que l'on veut obtenir :

— Diagramme de Bode, pour $\omega_0/2\pi = 1\text{kHz}$ et $\zeta = 0.1$



— Amplitude de transfert pour $\omega_0/2\pi = 1\text{kHz}$ et différentes valeurs de ζ.



Les fonctions dont on va avoir besoin sont :

- `loglog` pour tracer une fonction en échelle logarithme
- `semilogx` pour tracer une fonction en échelle semi-logarithme
- `xlabel`, `ylabel` et `title`
- `grid` pour afficher la grille
- `subplot(ny, nx, n)` pour tracer le nième graph (en partant de 1) d'une grille de ny x nx graphs
- L'option `label` et la fonction `legend` pour mettre une légende.
- La fonction `angle` de numpy permet d'avoir la phase d'un nombre complexe.

Vous aurez aussi besoin de savoir :

- comment formater une chaîne de caractère pour y mettre des paramètres
- (Facultatif) En ce qui concerne les lettres grecs, on peut utiliser l'unicode. Cependant, il existe une autre methode propre à matplotlib permettant de mettre des formules mathématique. Cette méthode s'inspire de la syntaxe de Latex. Par exemple pour mettre la formule $\omega_0/2\pi$, on utilise `$\omega_0/2\pi$` dans la chaine de caractère, de même pour ζ on utilise `$\zeta$`.

Quelques remarques supplémentaires :

- Evidement, il faut commencer par créer une fonction Python qui calcul H. Cette fonction devra fonctionner avec des tableaux numpy.
- Les points pour le graph doivent être distribués uniformement en echelle logarithmique (`np.logspace`).
- Faites attention aux facteurs 2π. On fait les calculs avec des pulsations, mais on trace avec la fréquence !

2 Formule de Simpson

On rappelle la formule de Simpson pour le calcul approché d'une intégrale :

$$\int_a^b f(x)\mathrm{d}x \simeq \Delta_x \sum_{i=0}^{N-1} \frac{f(x_i) + 4f(x_i + \frac{\Delta_x}{2}) + f(x_i + \Delta_x)}{6}$$

où $\Delta_x = \frac{b-a}{N}$ et $x_i = a + i\Delta_x$

1. Ecrivez une fonction en Python qui calcule l'intégrale en utilisant la méthode de Simpson et sans utiliser de boucle (on suposera que la fonction `f` est vectorisée).
2. Calculez l'intégrale de $\frac{1}{1+x^2}$ entre 0 et 1. Comparez à sa valeur théorique en fonction de N (on pourra faire un graph en échelle log/log). Qu'elle est la vitesse de convergence ?

3 Loi de Poisson

Une source lumineuse illumine un photomultiplicateur. Ce dispositif envoie un pulse digital d'environ 20 ns à chaque photon qu'il détecte. La sortie du photomultiplicateur est connectée à un dispositif informatique qui permet de compter le nombre de pulses reçu pendant une durée determinée.

Le nombre de photons qui arrive pendant une durée donnée suit une loi de Poisson, c'est à dire que la probabilité de détecter k photons est donnée par :

$$p(k) = \frac{\lambda^k}{k!} e^{-\lambda}$$

où λ est le nombre moyen de photons. Le paramètre λ sera proportionnel à la durée t_0 pendant laquelle on mesure le nombre de photons : $\lambda = \Gamma t_0$.

On rappelle que l'écart type de la loi de Poisson vaut $\sqrt{\lambda}$

1. Le fichier de données est enregistré sour forme d'un fichier texte. Chaque point correspond à une mesure de durée $t_0 = 200\mu$s. Lire le fichier et le convertir en entier :

   ```
   fichier = "100secondes_200us_count.txt"
   data = np.loadtxt(fichier, dtype=int)
   ```

   Quel est le nombre moyen de photons reçu par seconde ? Quelle est approximativement la puissance lumineuse ?
2. Calculez l'écart type et vérifiez qu'il vaut $\sqrt{\lambda}$.
3. En utilisant la fonction `numpy.unique`, avec l'option `return_counts=True`, tracez la distribution de probabilité.
4. Tracez les points représentants $p(k)/p(k+1)$.
5. Soit x_i le nombre de photons détectés au cours de la mesure numéro i. On veut simuler un jeu de données correspondant à des mesures prise pendant un temps $t = Nt_0$. On défini la variable x_j^N par

$$x_j^N = \sum_{i=0}^{i=N-1} x_{Nj+i}$$

Écrivez une fonction python `somme_par_paquet` qui effectue cette opération. Si la taille du tableau d'entrée n'est pas un multiple de N, on suprimera des points au début du tableau. Evidement les boucles `for` sont interdites ! On transformera le tableau en un tableau 2D et on fera une somme par ligne (argument axis de la méthode `sum`).

Tracez au cours du temps le flux de photons pris en intégrant sur 100 ms.

6. On peut créer un générateur de bits aléatoires à partir de cette séquence : si $x_{2j} > x_{2j+1}$ alors on prend 1, si $x_{2j} < x_{2j+1}$ on prend 0, sinon on élimine le point j.
   Créez cette suite de bits aléatoire que l'on appellera a_i
7. On peut ensuite créer une suite de nombre aléatoire entre 0 et 1 en regroupant les bits pour écrire le nombre en binaire. On prendra par exemple N=11.

$$b_j = \sum_{i=0}^{i=N-1} \frac{a_{Nj+i}}{2^{i+1}}$$

8. Si X et Y sont deux variables aléatoires ayant une distribution uniforme entre 0 et 1, alors on a

$$P(X^2 + Y^2 < 1) = \frac{\pi}{4}$$

   Déterminez π en utilisant notre générateur de nombre aléatoire.

4 Ensemble de Mandelbrot

L'ensemble de Mandelbrot est une fractale définie comme l'ensemble des points `c` du plan complexe pour lesquels la suite de nombres complexes définie par récurrence par :

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

est bornée.

On peut montrer que si il existe n tel que $|z_n| > 2$ alors la suite diverge (le réciproque est triviale). Pour calculer l'ensemble de Mandelbrot, pour chaque point c, on effectue au plus 100 itérations. On construit un image en associant à chaque point la plus petite valeur de $n < 100$ telle que $|z_n| > 2$ si cette valeur existe et 0 sinon.

Tracer l'ensemble de Mandelbrot pour des point c tels que $-2.13 < \mathrm{Re}(c) < 0.77$ et $-1.13 < \mathrm{Re}(c) < 1.13$. On pourra ensuite effectuer un zoom sur le bord de la structure.

Dans un premier temps, on pourra créer une fonction simple `mandel(c)` que l'on vectorisera à l'aide de la commande `vectorize`. On pourra ensuite créer une fonction qui utilise directement un tableau.