TP 1.2 Montage à diode et non linéarité

1 Matériel

— une diode
— un GBF
— un oscilloscope
— une boite à décade de résistance
— une carte d'acquisition eurosmart
— deux multimètres

2 Énoncé

Dans ce TP nous allons étudier la non-linéarité de circuits électriques. L'élément de base des circuits non linéaire est la diode.

1. Schématiser un filtre tel que :
   — l'entrée et la sortie sont reliées par un pont diviseur de tension comprenant une résistance de 350 Ω et une diode,
   — la sortie est mesurée aux bornes de la diode.
2. Effectuer les branchements de ce circuit et observer la réponse du système à l'oscilloscope pour un signal d'entrée sinusoïdal de fréquence 420 Hz et d'amplitude 2 V.

Nous allons maintenant étudier en représentation fréquentielle la réponse de ce circuit.

Graphes à rendre

4. À l'aide de la carte d'acquisition eurosmart et du logiciel Latis Pro, réaliser une acquisition des signaux d'entrée et de sortie. Vous pouvez réfléchir à quels paramètres d'acquisition il faut choisir.
5. Exporter vos données sous formes de fichier textes. Puis à l'aide d'un script python **tracer les spectres du signal d'entrée et de sortie**. Vous trouverez les fonctions pour calculer une transformée de Fourier, ainsi que pour tracer un graphe en annexe. Pour ce graphe il sera toléré de na pas afficher de barres d'erreur.

   N'oubliez pas d'enregistrer votre graphe et imprimez-le.

Pour comprendre le comportement observé, nous allons étudier la diode en régime continu.

6. Schématiser deux circuits comprenant un voltmètre, un ampèremètre et une source de tension continue pour mesurer la caractéristique courant-tension de la diode. Identifier les montages courte dérivation et longue dérivation.

7. Choisir un des deux montages et utiliser un voltmètre, un ampèremètre, et le GBF en mode DC, avec une résistance de protection en série du GBF. Mesurer la caractéristique courant-tension de la diode.
8. Quelle incertitude doit-on associer aux points de mesure si la notice des multimètres indique une précision de 0,3% L + 3 UR ?
9. En déduire une interprétation des observations faites avec le filtre pont diviseur de tension à l'aide de circuits équivalents.

Nous avons étudié le comportement en régime continu de la diode en fonction de la tension à ses bornes. Nous allons maintenant étudier son comportement en fonction de la fréquence d'excitation.

10. Effectuer à nouveau le montage du filtre à pont diviseur de tension. Choisir une résistance de 100 kΩ et étudier la tension de sortie du filtre en fonction de la fréquence de la tension d'excitation sinusoïdale d'amplitude 2 V.
11. Noter vos observations, proposer une modélisation en termes de dipôles linéaires de la diode dans ses différents régimes de fonctionnement.

3 Annexes

La commande `numpy.fft.fft()`

```
1
2 import numpy as np
3
4 TF_s = np.fft.fft(s)
```

permet de calculer la liste des coefficients complexes de chaque harmonique du signal s.

La commande `numpy.loadtxt()`

```
1
2 import numpy as np
3
4 data = np.loadtxt('datafile.txt')
```

permet d'importer les données écrites dans le fichier texte datafile.txt vers un tableau (numpy.array) data.

avec dans un fichier datafile.txt les nombres décimaux sont notés avec un point, les nombres sont séparés par des tabulations.

Les commandes `matplotlib.pyplot. x ou y label()`

```
1 import matplotlib.pyplot as plt
2
3 plt.ylabel('s(t) (en V)')
4
5 plt.xlabel("t (en s)")
```

permettent de légender les axes d'un graphe.

Les commandes `matplotlib.pyplot. x ou y ticks([liste de valeurs],[liste d'affichage])`

```
1 import matplotlib.pyplot as plt
2
3 plt.xticks([0, tau, T, 2*T],[r'$0$', r'$\tau$', r'$T$', r'$2T$'])
4
5 plt.yticks([0, 0.5, 1],[r'$0$', r'$\frac{A}{2}$', r'$A$'])
```

permettent de placer des valeurs remarquables sur les axes d'un graphe.
La commande `matplotlib.pyplot.savefig()`

```
import matplotlib.pyplot as plt

plt.savefig("graphe.pdf")
```

permet d'enregistrer le graphe comme graphe.pdf
La commande `matplotlib.pyplot.subplot(nbre de lignes,nbre de colonnes, numero)`

```
import matplotlib.pyplot as plt

plt.subplot(2,1,1)
```

permet de faire un tracé comportant plusieurs graphes l'un à côté de l'autre.
Les commandes `matplotlib.pyplot.plot(x,y,label="ma legende")`
et `matplotlib.pyplot.legend(loc='upper right')`

```
import matplotlib.pyplot as plt

plt.plot(x,y,label="ma legende")

plt.legend(loc='upper right')
```

permettent d'ajouter des légendes à un graphe.
Les commandes `matplotlib.pyplot.errorbar(x,y,xerr = , yerr = )`
avec `xerr =` barre d'erreur en x et `yerr =` barre d'erreur en y.
remplace la commande plot pour tracer un graphe avec des barres d'erreur.

CONCOURS CENTRALE•SUPÉLEC

Oral

Python

MP, PC, PSI, TSI

Réalisation de tracés

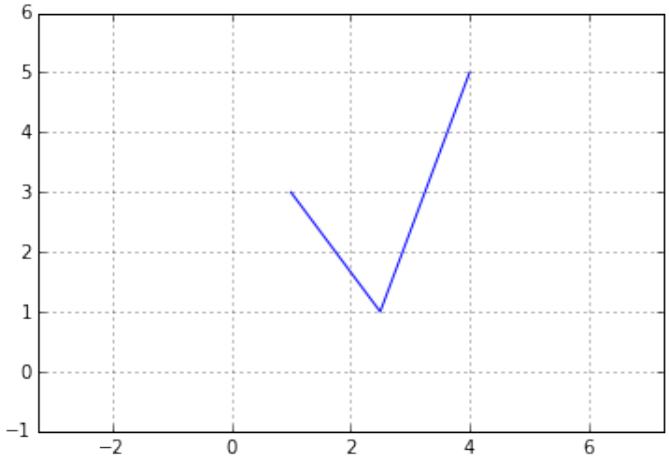
Les fonctions présentées dans ce document permettent la réalisation de tracés. Elles nécessitent l'import du module `numpy` et du module `matplotlib.pyplot`. De plus pour effectuer des tracés en dimension 3, il convient d'importer la fonction `Axes3d` du module `mpl_toolkits.mplot3d`. Les instructions nécessaires aux exemples qui suivent sont listés ci-dessous.

```
import math
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
```

Tracés de lignes brisées et options de tracés

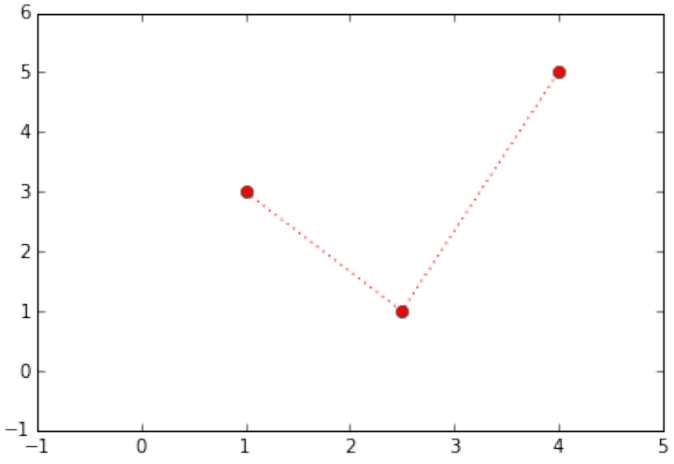
On donne la liste des abscisses et la liste des ordonnées puis on effectue le tracé. La fonction `axis` permet de définir la fenêtre dans laquelle est contenue le graphique. L'option `equal` permet d'obtenir les mêmes échelles sur les deux axes. Les tracés relatifs à divers emplois de la fonction `plot` se superposent. La fonction `plt.clf()` efface les tracés contenus dans la fenêtre graphique.

```
x = [1., 2.5, 4.]
y = [3., 1., 5.]
plt.axis('equal')
plt.plot(x, y)
plt.axis([-1., 5., -1., 6.])
plt.grid()
plt.show()
```



La fonction `plot` admet de nombreuses options de présentation. Le paramètre `color` permet de choisir la couleur (`'g'` : vert, `'r'` : rouge, `'b'` : bleu). Pour définir le style de la ligne, on utilise `linestyle` (`'-'` : ligne continue, `'--'` : ligne discontinue, `':'` : ligne pointillée). Si on veut marquer les points des listes, on utilise le paramètre `marker` (`'+'`, `'.'`, `'o'`, `'v'` donnent différents symboles).

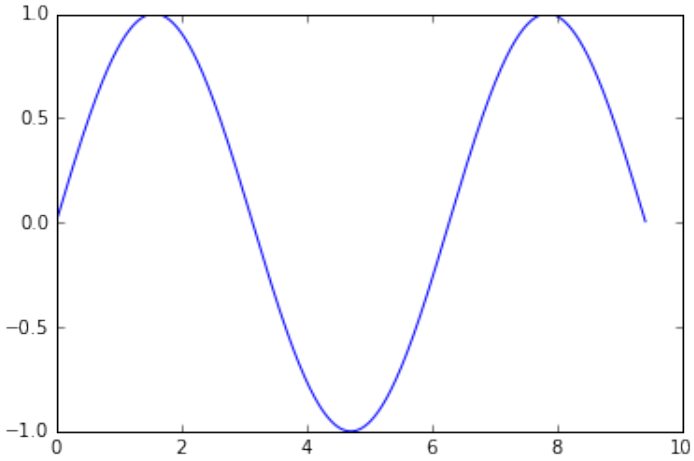
```
x = [1., 2.5, 4.]
y = [3., 1., 5.]
plt.axis([-1., 5., -1., 6.])
plt.plot(x, y, color='r', linestyle=':',
         marker='o')
plt.show()
```



Tracés de fonction

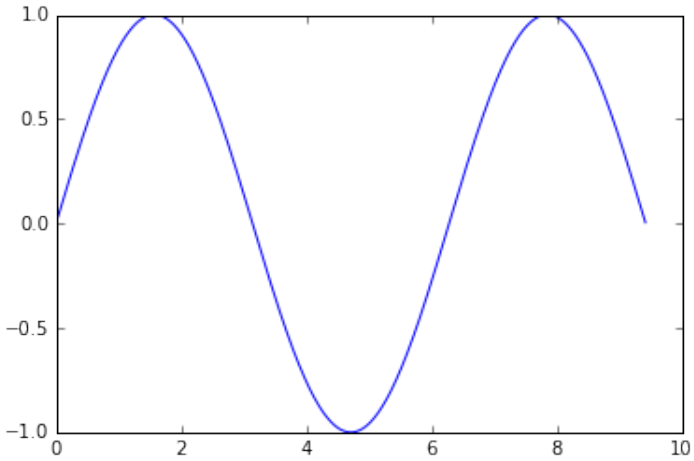
On définit une liste d'abscisses puis on construit la liste des ordonnées correspondantes. L'exemple ci-dessous trace $x \mapsto \sin x$ sur $[0, 3\pi]$.

```
def f(x):
    return math.sin(x)

X = np.arange(0, 3*np.pi, 0.01)
Y = [ f(x) for x in X ]
plt.plot(X, Y)
plt.show()
```



Il est généralement plus intéressant d'utiliser les fonctions du module `numpy`, plutôt que celles du module `math`, car elles permettent de travailler aussi bien avec des scalaires qu'avec des tableaux (on les appelle fonctions vectorisées et *universal function* ou `ufunc` dans la documentation officielle de Python).

```
def f(x):
    return np.sin(x)

X = np.arange(0, 3*np.pi, 0.01)
Y = f(X)
plt.plot(X, Y)
plt.show()
```



Une autre solution consiste à utiliser la fonction `vectorize` du module `numpy` qui permet de transformer une fonction scalaire en une fonction capable de travailler avec des tableaux. Il est cependant beaucoup plus efficace d'utiliser directement des fonctions vectorisées.

```
def f(x):
    return math.sin(x)

f = np.vectorize(f)
```

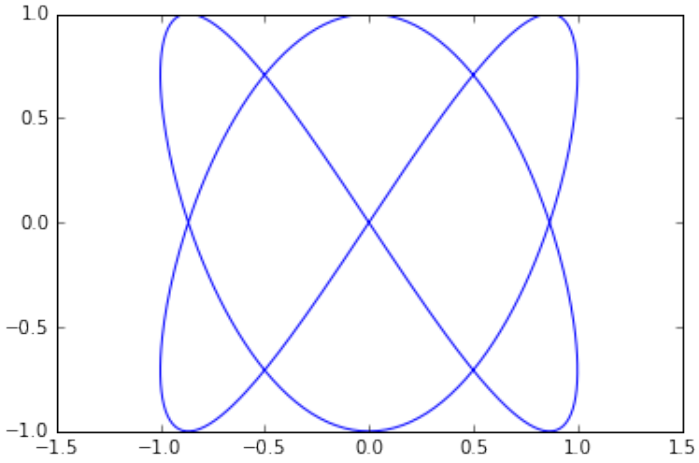
Il est à noter que les opérateurs python (`+`, `-`, `*`, etc.) peuvent s'appliquer à des tableaux, ils agissent alors terme à terme. Ainsi la fonction `f` définie ci-dessous est une fonction vectorisée, elle peut travailler aussi bien avec deux scalaires qu'avec deux tableaux et même avec un scalaire et un tableau.

```
def f(x, y):
    return np.sqrt(x**2 + y**2)

>>> f(3, 4)
5.0
>>> f(np.array([1, 2, 3]), np.array([4, 5, 6]))
array([ 4.12310563,  5.38516481,  6.70820393])
>>> f(np.array([1, 2, 3]), 4)
array([ 4.12310563,  4.47213595,  5.        ])
```

Tracés d'arcs paramétrés

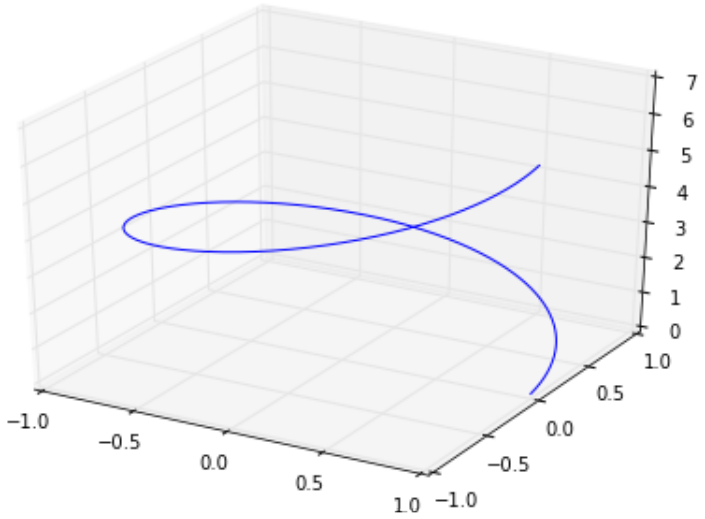
Dans le cas d'un arc paramétré plan, on définit d'abord la liste des valeurs données au paramètre puis on construit la liste des abscisses et des ordonnées correspondantes. On effectue ensuite le tracé.

```
def x(t):
    return np.sin(2*t)

def y(t):
    return np.sin(3*t)

T = np.arange(0, 2*np.pi, 0.01)
X = x(T)
Y = y(T)
plt.axis('equal')
plt.plot(X, Y)
plt.show()
```



Voici un exemple de tracé d'un arc paramétré de l'espace.

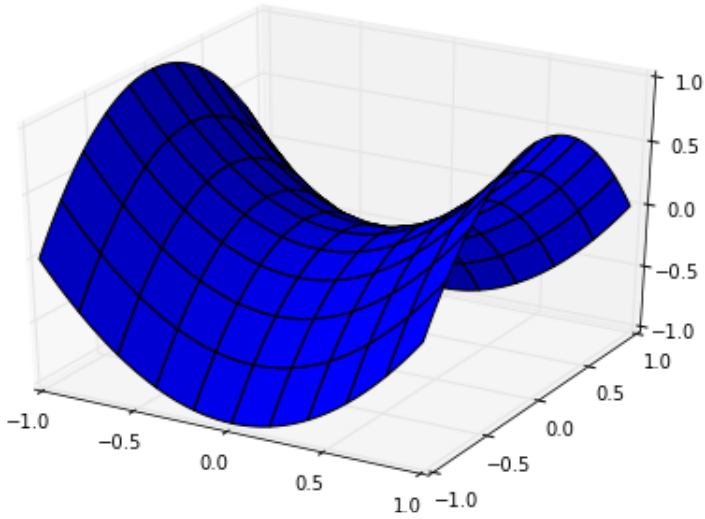
```
ax = Axes3D(plt.figure())
T = np.arange(0, 2*np.pi, 0.01)
X = np.cos(T)
Y = np.sin(T)
Z = T
ax.plot(X, Y, T)
plt.show()
```



Tracé de surfaces

Pour tracer une surface d'équation $z = f(x, y)$, on réalise d'abord une grille en (x, y) puis on calcule les valeurs de z correspondant aux points de cette grille. On fait ensuite le tracé avec la fonction `plot_surface`.

```
ax = Axes3D(plt.figure())

def f(x,y):
    return x**2 - y**2

f=np.vectorize(f)
X = np.arange(-1, 1, 0.02)
Y = np.arange(-1, 1, 0.02)
X, Y = np.meshgrid(X, Y)
Z = f(X, Y)
ax.plot_surface(X, Y, Z)
plt.show()
```



Tracé de lignes de niveau

Pour tracer des courbes d'équation $f(x,y) = k$, on fait une grille en x et en y sur laquelle on calcule les valeurs de f. On emploie ensuite la fonction `contour` en mettant dans une liste les valeurs de k pour lesquelles on veut tracer la courbe d'équation $f(x,y) = k$.

```
def f(x,y):
    return x**2 + y**2 + x*y

f=np.vectorize(f)
X = np.arange(-1, 1, 0.01)
Y = np.arange(-1, 1, 0.01)
X, Y = np.meshgrid(X, Y)
Z = f(X, Y)
plt.axis('equal')
plt.contour(X, Y, Z, [0.1,0.4,0.5])
plt.show()
```