Parcours : L2-ElnI

Semestre : 4

AU : 2021-2022

Réf. : GE-083

Abdelbacet Mhamdi

Docteur-Ingénieur en Génie Électrique

Maître Technologue en GE à l'ISET de Bizerte

abdelbacet.mhamdi@bizerte.r-iset.tn

Traitement de Signal

FASCICULE DE TRAVAUX PRATIQUES

Institut Supérieur des Études Technologiques de Bizerte

Disponible @ : https ://github.com/a-mhamdi/shelf_textbook/

Code d'honneur

The University of North Carolina at Chapel Hill
Department of Physics and Astronomy
http://physics.unc.edu/undergraduate-program/labs/general-info/

"During this course, you will be working with one or more partners with whom you may discuss any points concerning laboratory work. However, you must write your lab report, in your own words.

Lab reports that contain identical language are not acceptable, so do not copy your lab partner's writing.

If there is a problem with your data, include an explanation in your report. Recognition of a mistake and a well-reasoned explanation is more important than having high-quality data, and will be rewarded accordingly by your instructor. A lab report containing data that is inconsistent with the original data sheet will be considered a violation of the Honor Code.

Falsification of data or plagiarism of a report will result in prosecution of the offender(s) under the University Honor Code.

On your first lab report you must write out the entire honor pledge :

The work presented in this report is my own, and the data was obtained by my lab partner and me during the lab period.

On future reports, you may simply write "*Laboratory Honor Pledge*" and sign your name."

Table des matières

Pour pouvoir activer l'environnement virtuel et lancer **Jupyter Notebook**, vous devez procéder comme suit :

① Appuyez simultanément sur les touches & R du clavier. Cela ouvrira la boite de dialogue `Exécuter` ;

② Saisissez la commande `cmd` et appuyez sur la touche du clavier ;

③ Tapez ensuite la commande `ats.bat` sur la ligne d'invite de la console de commande ;

```
Command Prompt

C :\Users\admin> ats.bat
```

④ Pressez finalement la touche .

LAISSEZ LA CONSOLE DE COMMANDE DU SYSTÈME ACTIVE.

1 Prise en main de Python

Le code est disponible via https://github.com/a-mhamdi/cosnip/ → Python → sig-proc → init-jupyter.ipynb

Objectifs

1. Apprendre à programmer en **Python** ;
2. Se servir de l'environnement **Jupyter Notebook** ;
3. Utiliser les bibliothèques du calcul scientifique.

1.1 Variables numériques & Types

[1]:
```python
a = 1 # Un entier
print('La variable a = {} est de type {}'.format(a, type(a)))
```

```
La variable a = 1 est de type <class 'int'>
```

[2]:
```python
b = -1.25 # Un nombre réel
print('La variable b = {} est de type {}'.format(b, type(b)))
```

```
La variable b = -1.25 est de type <class 'float'>
```

[3]:
```python
c = 1+0.5j # Un nombre complexe
print('La variable c = {} est de type {}'.format(c, type(c)))
```

```
La variable c = (1+0.5j) est de type <class 'complex'>
```

1.2 Les chaînes de caractères

[4]:
```python
msg = "Mon Premier TP !"
print(msg, type(msg), sep = '\n***\n') # \n : retour à la ligne
print(msg + 3* '\nPython est simple')
```

```
Mon Premier TP !
***
<class 'str'>
Mon Premier TP !
Python est simple
Python est simple
Python est simple
```

[5]:
```
longMsg = """Ceci est un message,
sur plusieurs lignes"""
print(longMsg)
```

```
Ceci est un message,
sur plusieurs lignes
```

Indexation

[6]:
```
# Indexation positive
print(msg, msg[1:5], sep = ' -----> ')
# Indexation négative
print(msg, msg[-5:-1], sep = ' -----> ')
```

```
Mon Premier TP ! -----> on P
Mon Premier TP ! ----->  TP
```

Opérations sur les chaînes

[7]:
```
msg = 'Un message'
print(len(msg))
print(msg.lower())
print(msg.upper())
print(msg.split(' '))
print(msg.replace('mes', 'MES'))
print('a' in msg) # Vérifier si msg contient la lettre 'a'
```

```
10
un message
UN MESSAGE
['Un', 'message']
Un MESsage
True
```

[8]:
```
prix, nombre, perso = 100, 3, 'Un client'
print('{} demande {} pièces pour un prix de {} MDT !'.format(perso, nombre,␣
 ↪prix))
print('{1} demande {2} pièces pour un prix de {0} MDT !'.format(prix, perso,␣
 ↪nombre))
```

```
Un client demande 3 pièces pour un prix de 100 MDT !
Un client demande 3 pièces pour un prix de 100 MDT !
```

1.3 Binaire, octal & hexadécimal

[9]:
```
x = 0b0101 # 0b : binaire
print(x, type(x), sep = '\t----\t') # \t : tabulation
y = 0xAF # 0x : hexadécimal
print(y, type(y), sep = '\t' + '---'*5 + '\t')
z = 0o010 # 0o : octal
print(z, type(z), sep = ', ')
```

```
5       ----    <class 'int'>
175     --------------- <class 'int'>
8, <class 'int'>
```

Boolean

[10]:
```python
a = True
b = False
print(a)
print(b)
```

```
True
False
```

[11]:
```python
print("50 > 20 ? : {} \n50 < 20 ? : {} \n50 = 20 ? : {}\n50 /= 20 ? : {}"
      .format(50 > 20, 50 < 20, 50 == 20, 50 != 20)
     )
```

```
50 > 20 ? : True
50 < 20 ? : False
50 = 20 ? : False
50 /= 20 ? : True
```

[12]:
```python
print(bool(123), bool(0), bool('TP'), bool())
```

```
True False True False
```

[13]:
```python
var1 = 100
print(isinstance(var1, int))
var2 = -100.35
print(isinstance(var2, int))
print(isinstance(var2, float))
```

```
True
False
True
```

1.4 Listes, tuples & dictionnaires

Nous étudierons ici trois types de groupement de données, indexés et non ordonnés, dans le langage de programmation **Python** :

1. *Liste* est une collection modifiable. Elle autorise les termes redondants ;
2. *Tuple* est une collection immuable. Il autorise aussi la redondance ;
3. *Dictionnaire* est une collection modifiable. Un terme en double est interdit.

Lors du choix d'un type, il est utile de comprendre ses caractéristiques et de savoir comment **Python** gère ses définitions.

1.4.1 Opérations sur les listes

[14]:
```
maListe = ['a', 'b', 'c', 1, True] # Une liste mixte
print(maListe)
```

```
['a', 'b', 'c', 1, True]
```

[15]:
```
print(len(maListe)) # Taille de maListe
print(maListe[1:3]) # Accès aux élements de maListe
maListe[0] = ['1', 0] # Liste combinée
print(maListe)
print(maListe[3:])
print(maListe[:3])
```

```
5
['b', 'c']
[['1', 0], 'b', 'c', 1, True]
[1, True]
[['1', 0], 'b', 'c']
```

[16]:
```
maListe.append('etc') # Insertion de 'etc' à la fin de maListe
print(maListe)
```

```
[['1', 0], 'b', 'c', 1, True, 'etc']
```

[17]:
```
maListe.insert(1, 'xyz') # Insertion de 'xyz'
print(maListe)
```

```
[['1', 0], 'xyz', 'b', 'c', 1, True, 'etc']
```

[18]:
```
maListe.pop(1)
print(maListe)
```

```
[['1', 0], 'b', 'c', 1, True, 'etc']
```

[19]:
```
maListe.pop()
print(maListe)
```

```
[['1', 0], 'b', 'c', 1, True]
```

[20]:
```
del maListe[0]
print(maListe)
```

```
['b', 'c', 1, True]
```

[21]:
```
maListe.append('b')
print(maListe)
maListe.remove('b')
print(maListe)
```

```
['b', 'c', 1, True, 'b']
['c', 1, True, 'b']
```

[22]:
```python
# Loop
for k in maListe:
    print(k)
```

```
c
1
True
b
```

[23]:
```python
maListe.clear()
print(maListe)
```

```
[]
```

Méthode	***Description***
copy()	Renvoyer une copie de la liste
list()	Transformer en une liste
extend ()	Ajouter les éléments d'une liste (ou tout autre itérable), à la fin de la liste actuelle
count()	Renvoyer le nombre d'occurrences de la valeur spécifiée
index()	Retourner l'index de la première occurrence de la valeur spécifiée
reverse()	Inverser l'ordre la liste
sort()	Trier la liste

1.4.2 Opérations sur les tuples

[24]:
```python
ce_tuple = (1, 2, 3)
print(ce_tuple)
```

```
(1, 2, 3)
```

[25]:
```python
ce_tuple = (1, '1', 2, 'texte')
print(ce_tuple)
```

```
(1, '1', 2, 'texte')
```

[26]:
```python
print(len(ce_tuple))
```

```
4
```

[27]:
```python
print(ce_tuple[1:])
```

```
('1', 2, 'texte')
```

[28]:
```python
try:
    ce_tuple.append('xyz') # Déclencher une erreur
except Exception as err:
    print(err)
```

```
'tuple' object has no attribute 'append'
```

[29]:
```python
ma_liste = list(ce_tuple)
ma_liste.append('xyz')
print(ma_liste, type(ma_liste), sep = ', ')
```

```
[1, '1', 2, 'texte', 'xyz'], <class 'list'>
```

[30]:
```python
nv_tuple = tuple(ma_liste) # Convertir 'ma_liste' en tuple 'new_tuple'
print(nv_tuple, type(nv_tuple), sep = ', ')
```

```
(1, '1', 2, 'texte', 'xyz'), <class 'tuple'>
```

[31]:
```python
try:
    ce_tuple.insert(1, 'xyz') # Déclencher une erreur
except Exception as err:
    print(err)
```

```
'tuple' object has no attribute 'insert'
```

[32]:
```python
# Loop
for k in ce_tuple:
    print(k)
```

```
1
1
2
texte
```

[33]:
```python
res_tuple = ce_tuple + nv_tuple
print(res_tuple)
```

```
(1, '1', 2, 'texte', 1, '1', 2, 'texte', 'xyz')
```

1.4.3 Opérations sur les dictionnaires

[34]:
```python
# monDict = {"key": "value"}
monDict = {
    "Parcours" : "GM",
    "Spécialité" : "ElnI",
    "Sem" : "4"
}
print(monDict, type(monDict), sep = ', ')
```

```
{'Parcours': 'GM', 'Spécialité': 'ElnI', 'Sem': '4'}, <class 'dict'>
```

[35]:
```python
print(monDict["Sem"])
sem = monDict.get("Sem")
print(sem)
```

```
4
4
```

[36]:
```
monDict["Parcours"] = "GE"
print(monDict)
```

```
{'Parcours': 'GE', 'Spécialité': 'ElnI', 'Sem': '4'}
```

[37]:
```
# Boucle
for d in monDict:
    print(d, monDict[d], sep = '\t|\t')
```

```
Parcours        |       GE
Spécialité      |       ElnI
Sem     |       4
```

[38]:
```
for k in monDict.keys():
    print(k)
```

```
Parcours
Spécialité
Sem
```

[39]:
```
for v in monDict.values():
    print(v)
```

```
GE
ElnI
4
```

1.5 NumPy

La documentation officielle est disponible via ce lien : https://numpy.org/

Commençons d'abord par importer le module de calcul numérique **NumPy**. Nous le référençons désormais par l'acronyme *np*

[40]:
```
import numpy as np
```

NumPy vs Liste

[41]:
```
a_np = np.arange(6) # NumPy
print("a_np = ", a_np)
print(type(a_np))
a_lst = list(range(0,6)) # Liste
print("a_lst = ", a_lst)
print(type(a_lst))
# Comparaison
print("2 * a_np = ", a_np * 2)
print("2 * a_lst = ", a_lst * 2)
```

```
a_np =  [0 1 2 3 4 5]
<class 'numpy.ndarray'>
a_lst =  [0, 1, 2, 3, 4, 5]
<class 'list'>
```

```
2 * a_np =  [ 0  2  4  6  8 10]
2 * a_lst =  [0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5]
```

[42]:
```python
v_np = np.array([1, 2, 3, 4, 5, 6]) # NB : parenthèses puis crochets, i.e,␣
↪([])
print(v_np)
```

```
[1 2 3 4 5 6]
```

[43]:
```python
v_np = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print(v_np)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

[44]:
```python
print(type(v_np))
```

```
<class 'numpy.ndarray'>
```

[45]:
```python
print(v_np[0])
```

```
[1 2 3 4]
```

[46]:
```python
v_np.ndim # Dimensions de v_np
```

[46]: 2

[47]:
```python
v_np.shape # Nombre de lignes et de colonnes
```

[47]: (3, 4)

[48]:
```python
v_np.size # Nombre d'éléments dans v_np = 3 * 4
```

[48]: 12

Une autre démarche pour créer une matrice de taille (3,3) par exemple est :

[49]:
```python
u = np.arange(9).reshape(3,3)
print(u)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

Nous passons maintenant à présenter quelques opérations de base sur les matrices

[50]:
```python
M = np.array([[1, 2], [1, 2]])
print(M)
```

```
[[1 2]
 [1 2]]
```

[51]:
```python
N = np.array([[0, 3], [4, 5]])
print(N)
```

```
[[0 3]
 [4 5]]
```

Addition

[52]:
```python
print(M + N)
print(np.add(M, N))
```

```
[[1 5]
 [5 7]]
[[1 5]
 [5 7]]
```

Soustraction

[53]:
```python
print(M-N)
print(np.subtract(M, N))
```

```
[[ 1 -1]
 [-3 -3]]
[[ 1 -1]
 [-3 -3]]
```

Mutiplication élément par élément (en : Elementwise Product)

$$\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} .\times \begin{bmatrix} 0 & 3 \\ 4 & 5 \end{bmatrix} \quad = \quad \begin{bmatrix} 0 & 6 \\ 4 & 10 \end{bmatrix}$$

[54]:
```python
print(M * N)
print(np.multiply(M, N))
```

```
[[ 0  6]
 [ 4 10]]
[[ 0  6]
 [ 4 10]]
```

Produit matriciel

$$\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} \times \begin{bmatrix} 0 & 3 \\ 4 & 5 \end{bmatrix} \quad = \quad \begin{bmatrix} 8 & 13 \\ 8 & 13 \end{bmatrix}$$

[55]:
```python
print(M.dot(N))
print(np.dot(M, N))
```

```
[[ 8 13]
 [ 8 13]]
[[ 8 13]
 [ 8 13]]
```

Division élément par élément (en : Elementwise Division)

$$\begin{bmatrix} 0 & 3 \\ 4 & 5 \end{bmatrix} ./ \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 0:1 & 3:2 \\ 4:1 & 5:2 \end{bmatrix}$$

[56]:
```
print(N / M)
print(np.divide(N, M))
```

```
[[0.  1.5]
 [4.  2.5]]
[[0.  1.5]
 [4.  2.5]]
```

Calcul du déterminant

[57]:
```
print("Déterminant de M :")
print(np.linalg.det(M))
print("Déterminant de N :")
print(np.linalg.det(N))
```

```
Déterminant de M :
0.0
Déterminant de N :
-12.0
```

1.6 Matplotlib

De plus amples informations peuvent être trouvées à ce site : https://matplotlib.org/

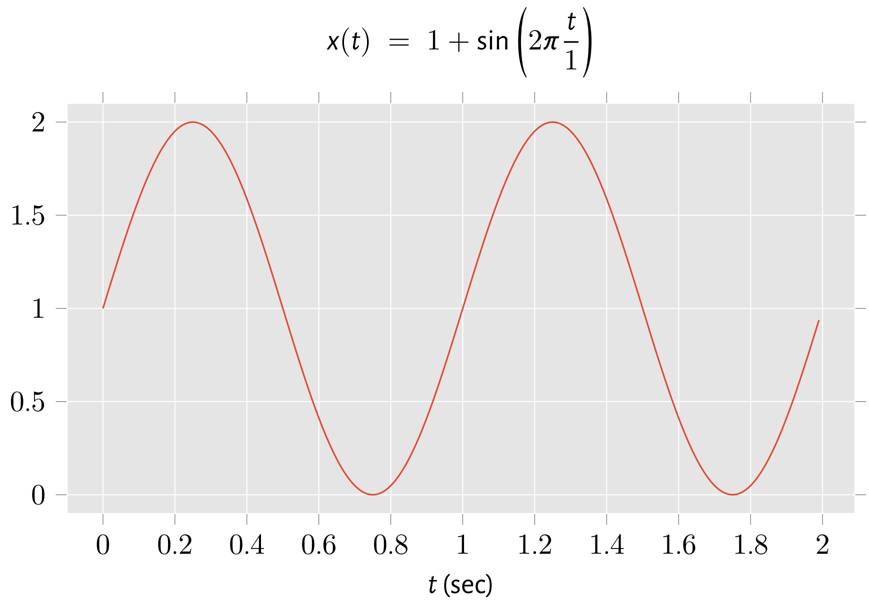
[58]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

Nous créons une fonction sinusoïdale qu'on dénote par x de période 1 sec, rehaussée d'une valeur constante de 1.

[59]:
```
# Fonction continue
t = np.arange(0.0, 2.0, 0.01)
x = 1 + np.sin(2 * np.pi * t) # Fréquence = 1Hz
```

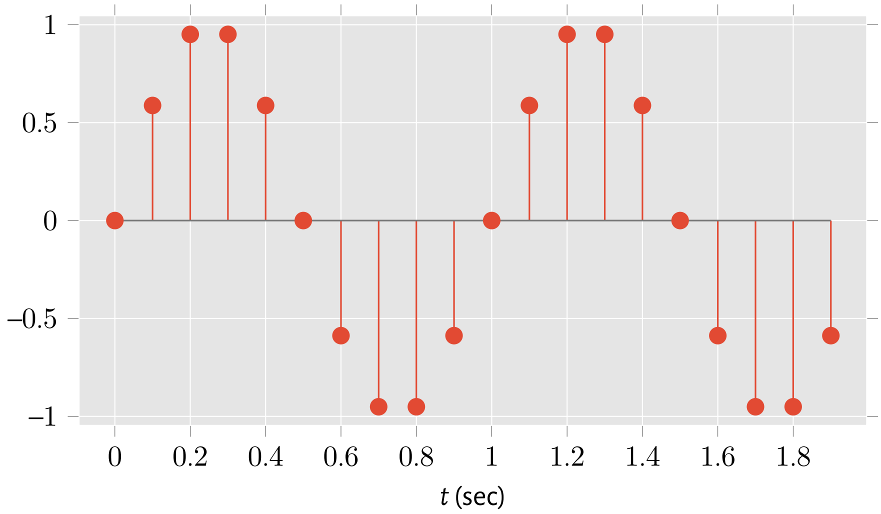
Les commandes qui permettent de tracer le graphique de x sont :

[60]:
```
plt.plot(t, x)

# Donner un titre au graphique
plt.title(r"$x(t) = 1+\sin\left(2\pi\frac{t}{1}\right)$")
plt.xlabel("$t$ (sec)") # Nommer l'axe des abscisses

plt.show()
```



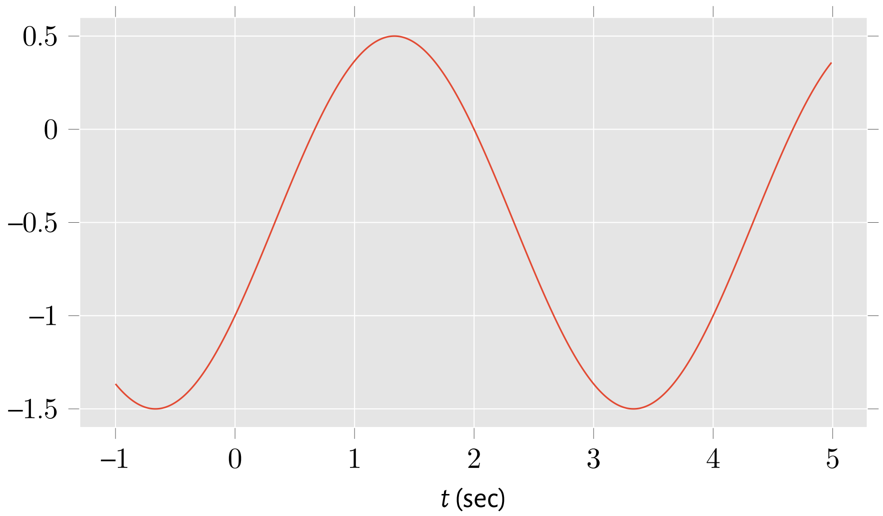
[61]:
```python
# Fonction échantillonnée
t = np.arange(0.0, 2.0, 0.1)
y = np.sin(2*np.pi*t) # Pareil ! Signal sinusoïdal de fréquence 1Hz
```

[62]:
```python
plt.stem(t, y)
plt.xlabel("$t$ (sec)")
plt.show()
```



Manipulation Nº 1 :

La courbe ci-dessous décrit l'évolution d'une fonction h au cours du temps t :



a) Déterminer les mesures de h :

- valeur moyenne ;
- période ;
- amplitude ;
- phase à l'origine.

b) Donner l'expression mathématique de h ;

c) Écrire un code **Python** qui permet de générer exactement le même graphique.

2 Convolution des signaux

2.1 Convolution 1D

Le code est disponible via https://github.com/a-mhamdi/cosnip/ → Python → sig-proc → conv-1d.ipynb

Nous rappelons que le résultat y d'un produit de convolution $u*h = h*u$ entre deux signaux u et h, est défini par la formule :

$$y(t) = \int_0^t h(t-\varsigma)u(\varsigma)d\varsigma$$

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use("ggplot")
```

[2]:
```
def G_tau(t, tau=0):
    return ( t >= tau ).astype(int)
```

[3]:
```
def r_tau(t, tau=0):
    return ( t >= tau ).astype(int)*(t-tau)
```

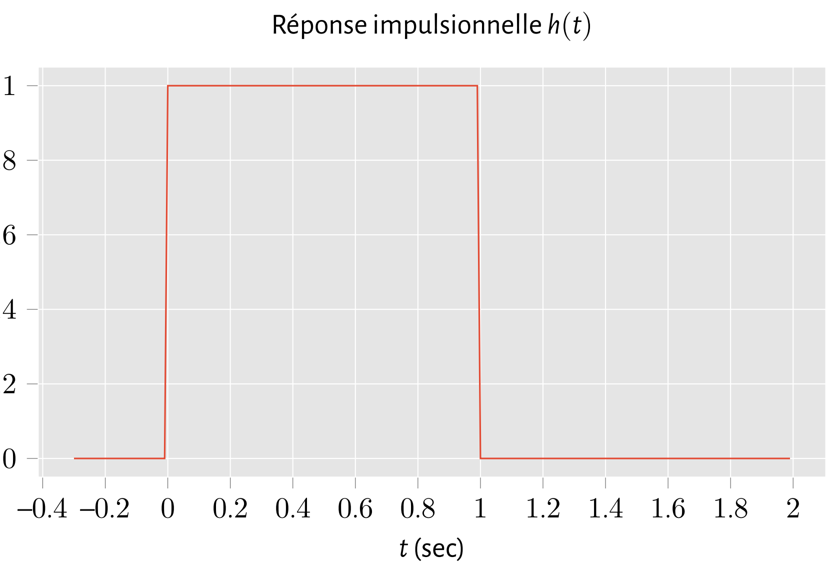
[4]:
```
Ts = .01 # Période d'échantillonnage
```

Soit un système LCIT, dont la réponse impulsionnelle est :

$$h(t) = \Gamma(t) - \Gamma_1(t)$$

[5]:
```
t_h = np.arange(-.3, 2, Ts)
h = G_tau(t_h) - G_tau(t_h, 1)
```

[6]:
```
plt.plot(t_h, h)
plt.title("Réponse impulsionnelle $h(t)$")
plt.xlabel("$t$ (sec)")
plt.grid()
```

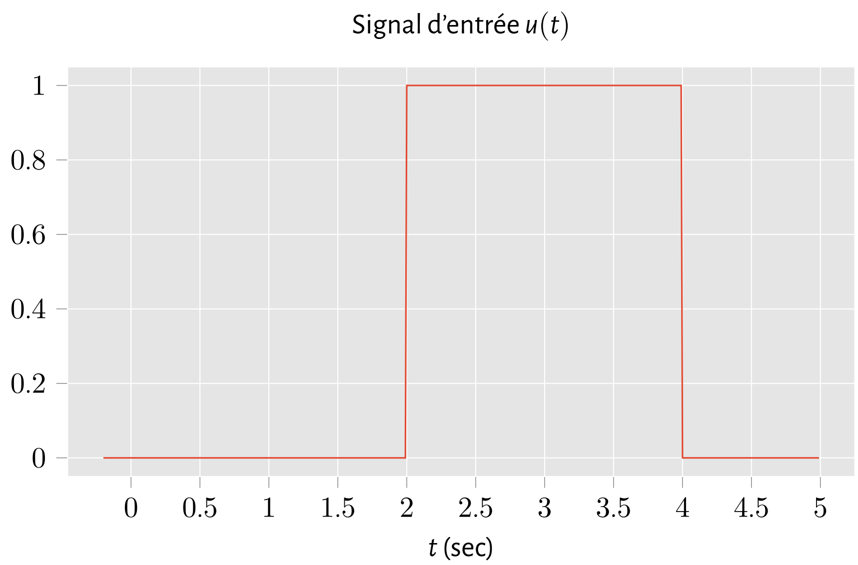


On applique à l'entrée du système la fonction u suivante :

$$u(t) = \Gamma_2(t) - \Gamma_4(t)$$

[7]:
```
t_u = np.arange(-.2, 5, Ts)
u =  G_tau(t_u, 2) - G_tau(t_u, 4)
```

[8]:
```
plt.plot(t_u, u)
plt.title("Signal d'entrée $u(t)$")
plt.xlabel("$t$ (sec)")
plt.grid()
```



Observons que h et u n'ont pas la même taille.

[9]:
```
print(type(h), h.shape)
print(type(u), u.shape)
```

```
<class 'numpy.ndarray'> (230,)
<class 'numpy.ndarray'> (520,)
```

On dénote par `y_sim` le résultat de la convolution numérique entre u et h.

[10]:
```
y_sim = np.convolve(u, h)/h.sum()
t_y = np.arange(-.5, len(y_sim)*Ts-.5, Ts)
```

La réponse du système est donnée par :

$$y(t) = u(t) * h(t)$$

Soit encore :

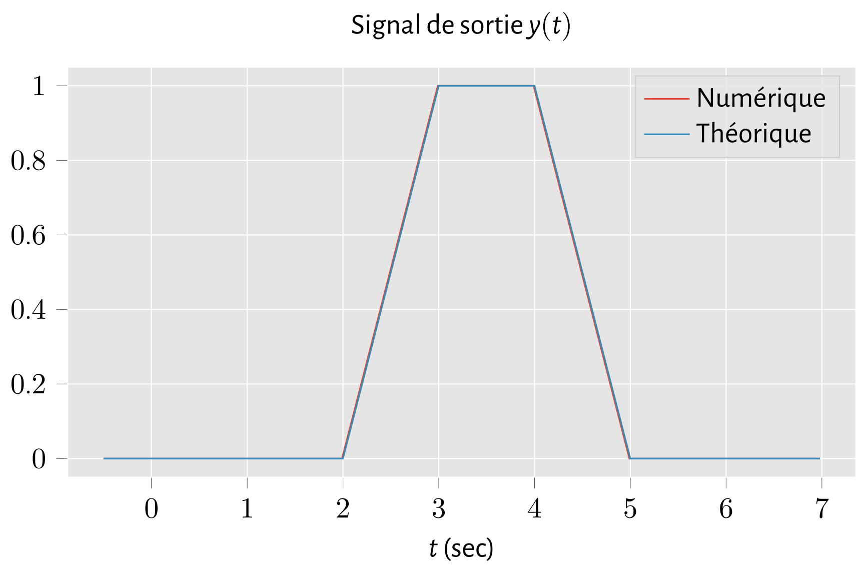
$$y(t) = (\Gamma_2 - \Gamma_4) * (\Gamma - \Gamma_1)$$

La réponse théorique finale est donnée par :

$$y(t) = r_2(t) - r_3(t) - r_4(t) + r_5(t)$$

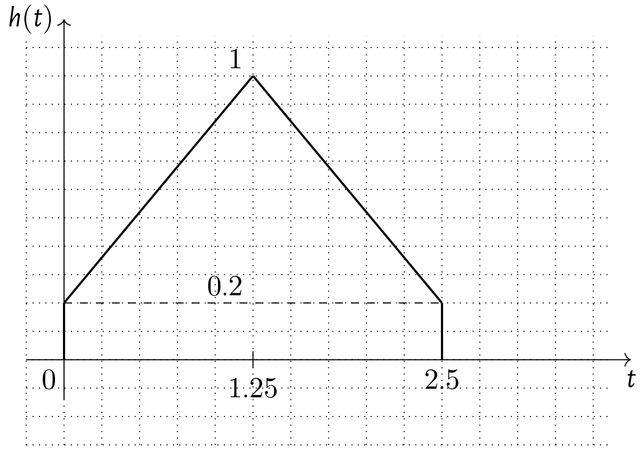
[11]:
```
y = r_tau(t_y, 2)-r_tau(t_y, 3)-r_tau(t_y, 4)+r_tau(t_y, 5)
```

[12]:
```
plt.plot(t_y, y_sim, label='Numérique')
plt.plot( t_y, y, label='Théorique')
plt.title("Signal de sortie $y(t)$")
plt.xlabel("$t$ (sec)")
plt.legend()
plt.grid()
```



Manipulation № 2 :

La fonction h dénote la réponse impulsionelle d'un système LCIT. Son graphe d'évolution est indiqué par le graphique ci-après :



a) Déterminer l'expression du signal h ;

b) Écrire un code **Python** qui permet de tracer ce signal ;

c) Tracer l'évolution du signal d'entrée u, donné par

$$u(t) \quad = \quad 0.8\Gamma(t) - \Gamma_{1.5}(t) + 0.2\Gamma_2(t)$$

d) Calculer et implémenter le produit

$$y(t) \quad = \quad u(t) * h(t)$$

e) Sur une même figure, tracer les deux résultats de convolution, numérique & théorique.

2.2 Convolution 2D

Le code est disponible via `https://github.com/a-mhamdi/cosnip/` → `Python` → `sig-proc` → `conv-2d.ipynb`

Un noyau d'image ou masque est une petite matrice utilisée pour appliquer des effets comme ceux qu'on pourrait trouver dans les applications de traitement d'images, tels que la netteté et la détection de contours, etc.

Ceci est accompli en faisant une convolution entre l'image et le masque.

Commençons par importer l'image "*wb_IsetB.png*" en niveau de gris

[1]:
```python
%matplotlib inline
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from scipy import ndimage
```

[2]:
```python
img_np = plt.imread("wb-IsetB.png")
# Afficher l'image en niveau de gris
ma_fig = plt.imshow(img_np, cmap = 'gray')
plt.axis('off')
```

```
ma_fig.axes.get_xaxis().set_visible(False)
ma_fig.axes.get_yaxis().set_visible(False)
```

Fixons d'abord la structure du noyau à appliquer. Il s'agit par la suite du masque *emboss*. Ce dernier permet de mettre en relief un effet d'ombre 3D à l'image. L'image résultante donne l'illusion de profondeur.

La structure de ce masque est donnée par le code suivant :

[3]:
```
# Structure du masque
emboss_ker = np.array([[-2, -1, 0], [-1, 1, 1], [0, 1, 2]])
print(emboss_ker)
```

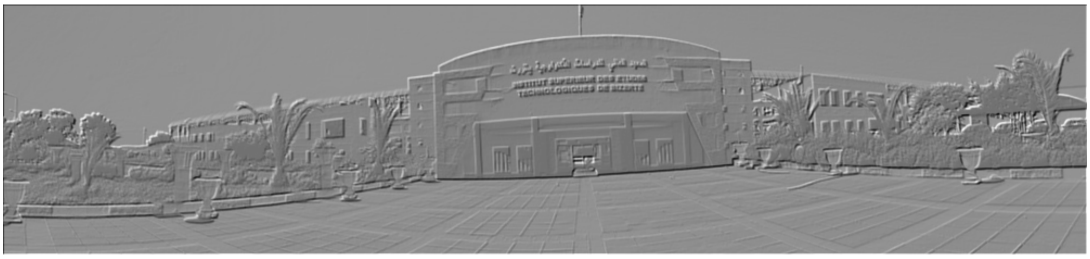
```
[[-2 -1  0]
 [-1  1  1]
 [ 0  1  2]]
```

On peut se référer à la documentation disponible au lien : https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.convolve.html#scipy.ndimage.convolve

[4]:
```
# Convolution 2D
mod_img = ndimage.convolve(img_np, emboss_ker, mode = 'constant', cval = 0.0)
```

Pour visualiser et sauvegarder l'image modifiée suite à l'application du noyau susmentionné, nous écrivons les instructions suivantes :

[5]:
```
mod_fig = plt.imshow(mod_img, cmap = 'gray')
plt.axis('off')
mod_fig.axes.get_xaxis().set_visible(False)
mod_fig.axes.get_yaxis().set_visible(False)
matplotlib.image.imsave('mod-IsetB.png', mod_img, cmap = 'gray')
```



Manipulation № 3 :

Vous pouvez vous référer au site : http://setosa.io/ev/image-kernels/.

a) Convertir l'image *"logo-ISET-Bizerte.png"* en niveau de gris et la charger par la suite dans votre notebook **Jupyter** ;

b) Appliquer deux masques de votre choix ;

c) Justifier à chaque fois la transformation engendrée par application du masque.

3 Décomposition en série de Fourier

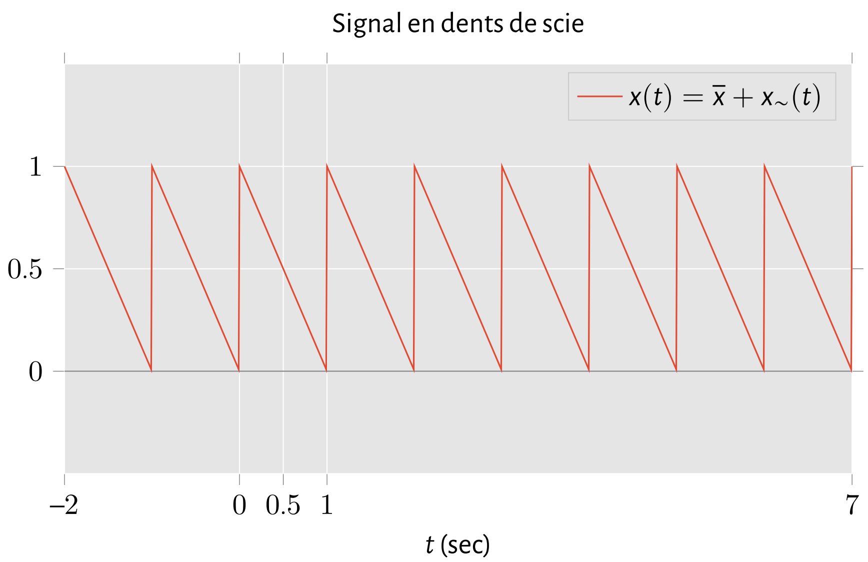
Le code est disponible via https://github.com/a-mhamdi/cosnip/ → Python → sig-proc → fourier-series.ipynb

Soit le code suivant :

[1]:
```
%matplotlib inline
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt
plt.style.use('ggplot')
plt.rc({"figure.figsize": (8, 4), "keymap.grid": "g", "font.serif":
    "Charter", "font.size": 10})
```

[2]:
```
tot_pts = 1000
t = np.linspace(-2, 7, tot_pts) # Vecteur temps  : t
wt = 2*np.pi*t # Fréquence : f = 1Hz
```

[3]:
```
plt.axhline(0, color = 'gray', lw = 1)
plt.plot(t, 0.5 + 0.5 * signal.sawtooth(wt, 0), lw = 1.5, label = r'$x(t) =
    \overline{x} + x_{\sim}(t)$')

plt.yticks([-1, 0, 0.5, 1, 2], ['$-1$', '$0$', '$0.5$', '$1$', '$2$'])
plt.xticks([-2, 0, 0.5, 1, 7], ['$-2$', '$0$', '$0.5$', '$1$', '$7$'])
plt.xlim(-2, 7)
plt.ylim(-0.5, 1.5)

plt.legend(fontsize = 13, fancybox = True, framealpha = 0.3, loc = 'best')
plt.title('Signal en dents de scie')
plt.xlabel('$t$ (sec)')
plt.show()
```



Rappelons d'abord la définition du signal x :

$$x(t) \;=\; \overline{x} + x_{\sim}(t)$$

La valeur moyenne du signal est :

$$\overline{x} \;=\; 0.5$$

Il reste maintenant à calculer la partie alternative de $x_{\sim}$. Par définition, l'expression de $x_{\sim}$ est donnée par :

$$x_{\sim}(t) \;=\; \sum_{k=1}^{+\infty} a_k \cos\left(2k\pi\frac{t}{\mathsf{T}}\right) + \sum_{k=1}^{+\infty} b_k \sin\left(2k\pi\frac{t}{\mathsf{T}}\right)$$

Le terme a_k est :

$$a_k \;=\; \frac{2}{\mathsf{T}}\int_0^{\mathsf{T}} x(t)\cos\left(2k\pi\frac{t}{\mathsf{T}}\right)dt, \qquad \text{avec} \quad \mathsf{T} = 1\,\text{sec.}$$

Le signal x est de période T = 1 sec. Il est de nature impair. Il en résulte que :

$$a_k \;=\; 0, \qquad \forall k \;=\; 1,\,2,\,\cdots$$

Chaque coefficient b_k se calcule de la façon suivante :

$$b_k \;=\; \frac{2}{\mathsf{T}}\int_0^{\mathsf{T}} x(t)\sin\left(2k\pi\frac{t}{\mathsf{T}}\right)dt, \qquad \text{avec} \quad \mathsf{T} = 1\,\text{sec.}$$

La troncature de x sur une période est :

$$x(t) \;=\; \Gamma(t) - r(t)$$

L'équation de b_k se transforme ainsi en

$$b_k \;=\; \frac{2}{\mathsf{T}}\int_0^{\mathsf{T}} (\Gamma(t) - r(t))\sin\left(2k\pi\frac{t}{\mathsf{T}}\right)dt, \qquad \text{avec} \quad \mathsf{T} = 1\,\text{sec.}$$

Soit encore

$$b_k \;=\; -\frac{2}{\mathsf{T}}\int_0^{\mathsf{T}} t\sin\left(2k\pi\frac{t}{\mathsf{T}}\right)dt$$

Après intégration, on obtient :

$$b_k = \frac{1}{k\pi}\left[t\cos\left(2k\pi\frac{t}{T}\right)\right]_0^T$$

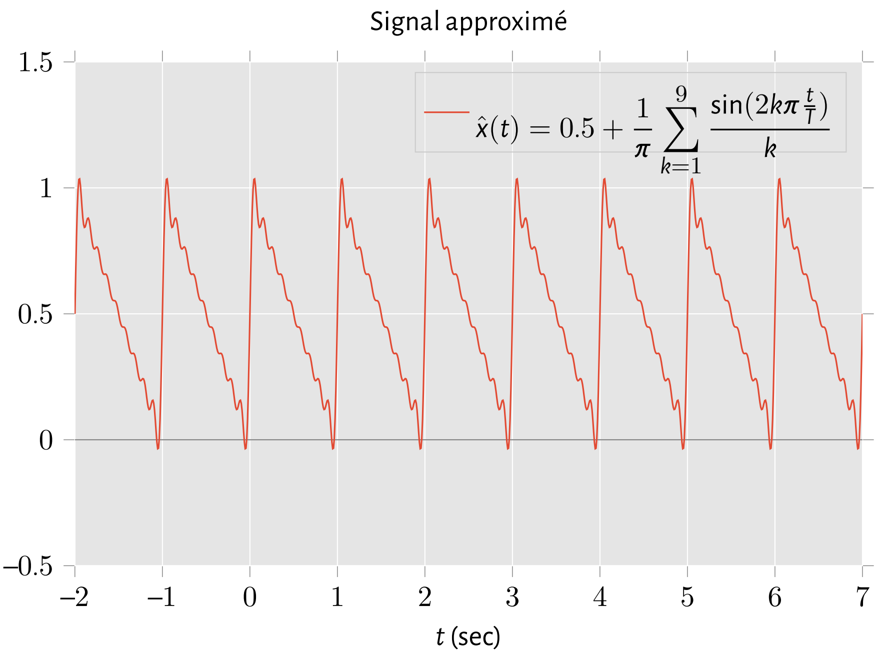
Finallement :

$$b_k = \frac{1}{k\pi}$$

Compte tenu de ce qui précède, l'expression finale de x est

$$x(t) = 0.5 + \frac{1}{\pi}\sum_{k=1}^{+\infty}\frac{\sin\left(2k\pi\frac{t}{T}\right)}{k}$$

[4]:
```
plt.axhline(0, color = 'gray', lw = 1)
# Itérer sur une liste
x_lst = [1/(k*np.pi) * np.sin(k * wt) for k in range(1, 10)]
# Convertir en np
x_np = np.asarray(x_lst, dtype=np.float32)
# Transformer le vecteur x_np en une matrice de 9 colonnes
x_np.reshape(tot_pts, 9)
# Valeur moyenne = 0.5
x_app = 0.5 + np.sum(x_np, axis = 0)

plt.plot(t, x_app, label = r'$\hat{x}(t) = 0.5 +␣
↪\frac{1}{\pi}\sum_{k=1}^{9}\frac{\sin(2k\pi\frac{t}{T})}{k}$')

plt.xlim(-2, 7)
plt.ylim(-0.5, 1.5)

plt.legend(fontsize = 13, fancybox = True, framealpha = 0.3, loc = 'best')
plt.title("Signal approximé")
plt.xlabel('$t$ (sec)')
plt.show()
```



Manipulation № 4 :

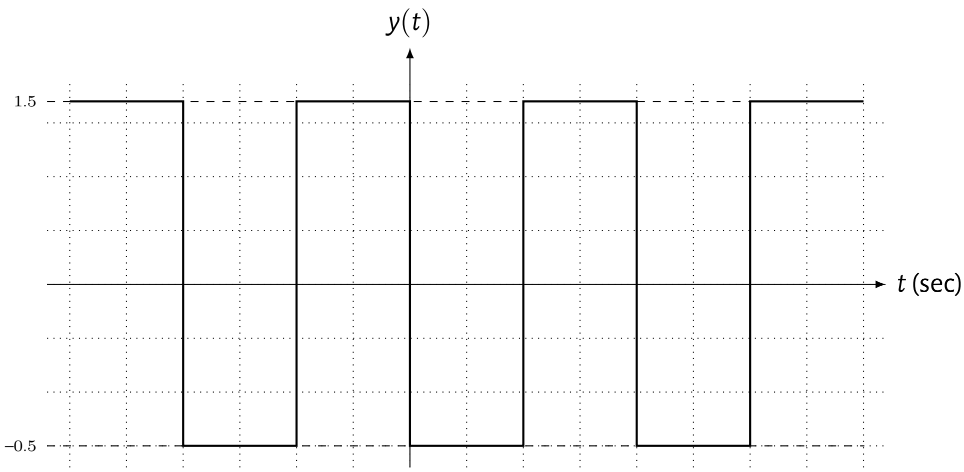
Il s'agit de faire les activités suivantes :

a) Déterminer le code qui permet de créer la fonction x ;

b) Quelles lignes permettent de représenter la courbe de x ;

c) À partir du code de la cellule 4, expliquer comment construire le signal approché x_{app} ;

d) Le graphe ci-dessous représente l'évolution d'un signal carré y, de période T = 2 sec et de rapport cyclique égal à 50 %.



- Écrire un code **Python** qui permet de générer et d'afficher le signal y ;
- Décomposer y en sa série de Fourier ;
- Implémenter vos résultats théoriques en **Python** *(10 harmoniques !)* ;
- Afficher le signal approché.

4 Transformée de Fourier

Le code est disponible via `https://github.com/a-mhamdi/cosnip/` → `Python` → `sig-proc` → `fourier-transform.ipynb`

Rappelons d'abord la définition de la transformée de Fourier d'un signal x, soit encore $\mathfrak{F}\left\{x(t)\right\}$ qu'on dénote par $\mathcal{X}(f)$:

$$\mathcal{X}(f) \;=\; \int_{-\infty}^{+\infty} x(t)\mathrm{e}^{-2\jmath\pi ft}\,dt$$

Par examen de cette transformation, nous observons qu'il est impossible d'implémenter cette intégrale en temps continu sur un calculateur. Ce dernier ne travaille que sur des valeurs discrètes, nous ferons recours à la *Transformée de Fourier Discrète*. Elle consiste d'abord à discrétiser et à tronquer x en une série $x_0, \cdots, x_{n-1}$. Les coefficients discrets de $\mathcal{X}(f)$ sont calculés par la suite conformément à la formule suivante :

$$\mathcal{X}_l \;=\; \sum_{p=0}^{n-1} x_p \mathrm{e}^{-\dfrac{2\jmath\pi pl}{n}}, \quad \text{avec} \quad l \;=\; 0, \cdots, n-1$$

Néanmoins, le calcul des coefficients $\mathcal{X}_l$, pour $l \;=\; 0, \cdots, n{-}1$, à partir de la définition est souvent gourmand en temps. Un autre algorithme très répandu dans les applications d'ingénierie est la *Transformée de Fourier Rapide*, souvent abrégée *FFT*. Cette approche de calcul permet de réduire énormément la complexité du calcul des termes susmentionnés.

Une explication détaillée avec une implémentation en **Python** est accessible via le lien suivant : `https://towardsdatascience.com/fast-fourier-transform-937926e591cb`

Commençons d'abord par importer les modules requis

[1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

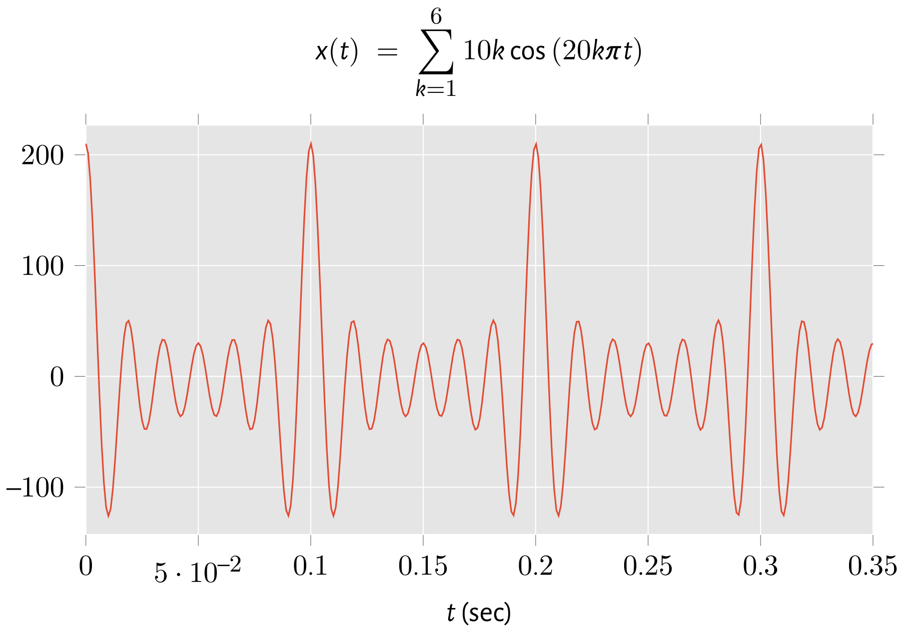
Nous générons maintenant un signal x défini par :

$$x(t) \;=\; \sum_{k=1}^{6} \mathcal{A}k \cos\left(2\mathcal{A}k\pi t\right) \quad \text{avec} \quad \mathcal{A} = 10\,\text{Hz}.$$

[2]:
```
nb_pts = 1000 # Nombre de points
Delta_t = 0.001 # Période d'échantillonnage
t = np.linspace(0.0, nb_pts * Delta_t, nb_pts) # Vecteur temps
```

```
wt = 2.0 * np.pi * t
x_lst = [10 * k * np.cos(10 * k * wt) for k in range(1, 7)]
xmat_t = np.asarray(x_lst, dtype = np.float32)
x_t = np.sum(xmat_t, axis = 0)
```

[3]:
```
plt.plot(t, x_t)
plt.title(r'$x(t) \;=\; \sum_{k=1}^{6} 10 k \cos\left(20 k \pi t\right)$')
plt.grid()
plt.xlim(0, 0.35)
plt.xlabel("$t$ (sec)")
plt.show()
```



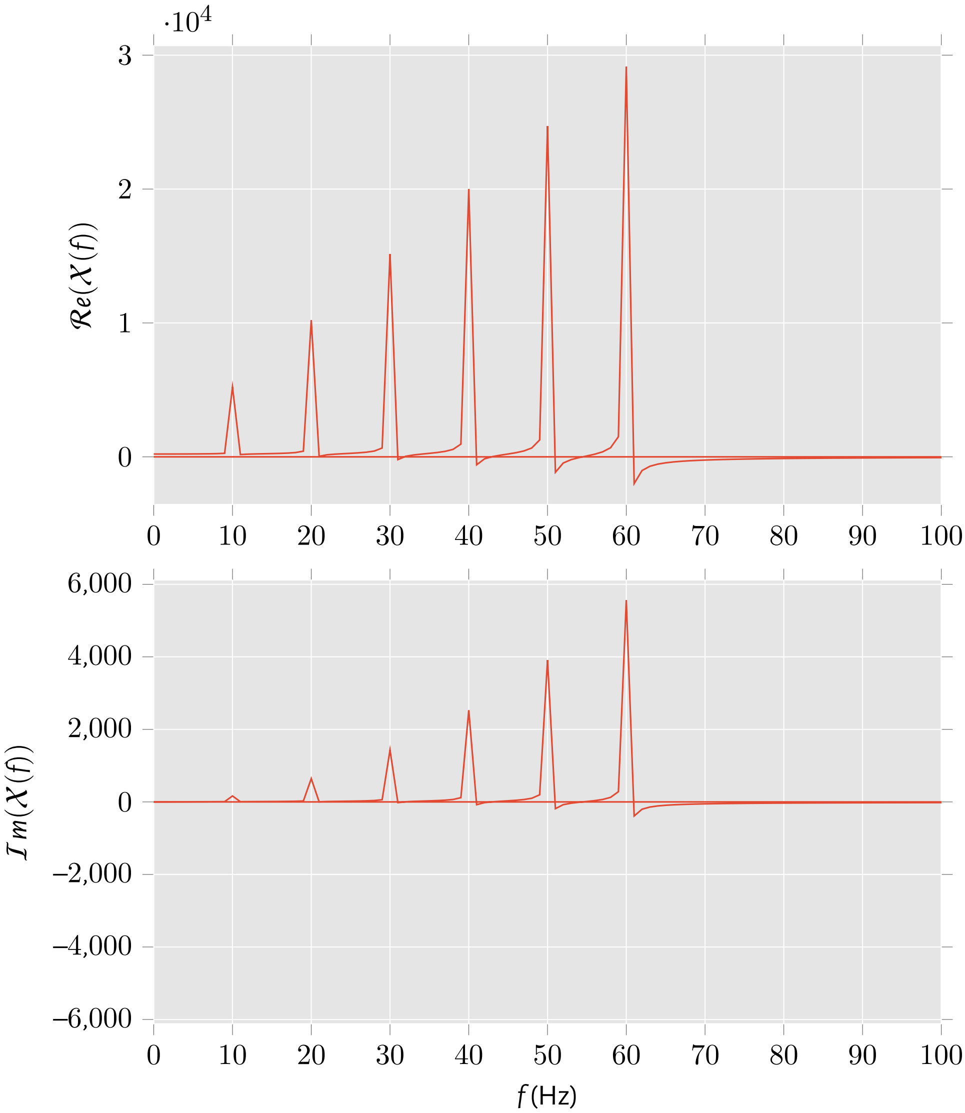
Appliquons la transformée Fourier rapide *(en : FFT ou Fast Fourier Transform)* de $x(t)$. La quantité **x_f** dénote $\mathcal{X}(f)$.

[4]:
```
x_f = np.fft.fft(x_t)
freqs = np.fft.fftfreq(nb_pts, Delta_t)
```

Traçons par la suite les parties réelle et imaginaire de $\mathcal{X}(f)$

[5]:
```
plt.subplot(2, 1, 1)
plt.plot(freqs, x_f.real)
plt.xlabel("$f$ (Hz)")
plt.ylabel(r"$\mathcal{R}e(\mathcal{X}(f))$")
plt.xlim(0, 100)
plt.grid()
plt.subplot(2, 1, 2)
plt.plot(freqs, x_f.imag)
plt.xlabel("$f$ (Hz)")
plt.ylabel(r"$\mathcal{I}m(\mathcal{X}(f))$")
```

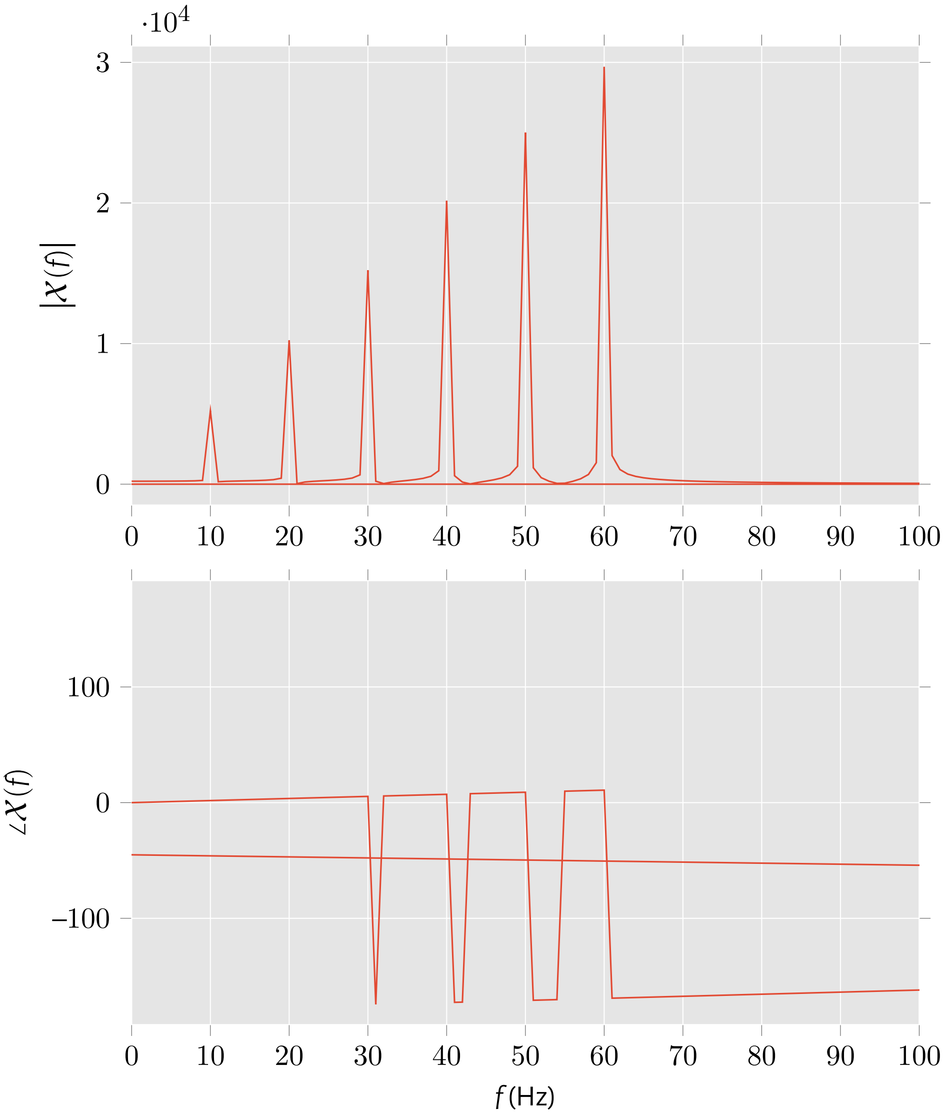
```
plt.xlim(0, 100)
plt.grid()
plt.show()
```



Une autre manière de présentation de $\mathcal{X}(f)$ est de tracer les graphes de $\left|\mathcal{X}(f)\right|$ et $\angle\mathcal{X}(f)$.

[6]:
```
plt.subplot(2, 1, 1)
plt.plot(freqs, np.abs(x_f))
plt.xlabel("$f$ (Hz)")
plt.ylabel(r"$\left|\mathcal{X}(f)\right|$")
plt.xlim(0, 100)
plt.grid()
plt.subplot(2, 1, 2)
plt.plot(freqs, np.angle(x_f, deg = True))
plt.xlabel("$f$ (Hz)")
plt.ylabel(r"$\angle{\mathcal{X}(f)}$")
plt.xlim(0, 100)
```

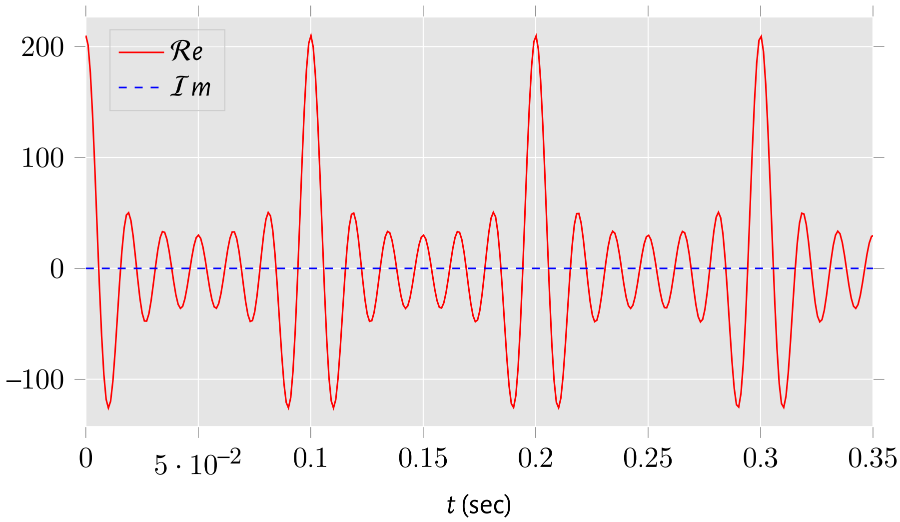
```
plt.grid()
plt.show()
```



Essayons de reconstruire $x(t)$ par application de la transformée de Fourier inverse **ifft** *(en : Inverse Fast Fourier Transform)*.

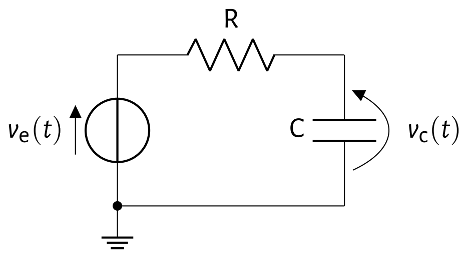
[7]:
```
x_t_app = np.fft.ifft(x_f)
plt.plot(t, x_t_app.real, '-r', t, x_t_app.imag, '--b')
plt.title(r'$\hat{x}(t) \;=\;␣
 ↪\mathfrak{F}^{-1}\left\{\mathcal{X}(f)\right\}$')
plt.legend(['$\mathcal{R}e$', '$\mathcal{I}m$'], fancybox = True, framealpha␣
 ↪= 0.3, loc = 'best')
plt.grid()
plt.xlim(0, 0.35)
plt.xlabel("$t$ (sec)")
plt.show()
```

$$\hat{x}(t) = \mathfrak{F}^{-1}\{\mathcal{X}(f)\}$$



Manipulation Nº 5 :

On considère le circuit RC suivant :



On rappelle qu'un tel circuit pour un condensateur initialement déchargé, c.-à-d. $v_c(t=0) = 0$, est régit par l'équation différentielle suivante :

$$\tau\frac{dv_c(t)}{dt} + v_c(t) = v_e(t), \quad \text{avec} \quad v_c(t=0) = 0,$$

où $\tau = 1$ sec désigne la constante de temps du montage.

a) On opère d'abord dans le domaine temporel :

- Calculer sa réponse impulsionnelle $h(t)$;
- Tracer la courbe de h pour $0 \leq t \leq 10$ sec;
- Écrire le code **Python** pour le calcul et le traçage du module et de l'argument de $\hat{\mathcal{H}}(f)$ dans un repère semi-logarithmique :

$$\hat{\mathcal{H}}(f) = \mathfrak{F}\{h(t)\};$$

b) On considère par la suite le domaine fréquentiel. L'opérateur de la dérivation dans l'espace temporel est équivalent à une multiplication par $\jmath\omega = 2\jmath\pi f$ dans le domaine spectral :

$$\frac{dv_c}{dt} \longrightarrow 2\jmath\pi f\mathcal{V}_c(f), \quad \text{où} \quad \mathcal{V}_c(f) = \mathfrak{F}\{v_c(t)\}.$$

- Partant de l'équation descriptive du système, écrire $\mathcal{H}(f)$:

$$\mathcal{H}(f) = \frac{\mathcal{V}_c(f)}{\mathcal{V}_e(f)}, \quad \text{où} \quad \begin{cases} \mathcal{V}_c(f) &= \mathfrak{F}\left\{v_c(t)\right\}, \\ \mathcal{V}_e(f) &= \mathfrak{F}\left\{v_e(t)\right\}. \end{cases}$$

- Proposer les instructions **Python** qui permettent de tracer le module et l'argument de $\mathcal{H}(f)$ sur une échelle semi-logarithmique pour 10^{-3} rad/sec $\leq 2\pi f \leq 10^3$ rad/sec;
- Utiliser la fonction `ifft` pour trouver et représenter la fonction $\hat{h}(t)$;
- Justifier la similitude entre les graphes de $h(t)$ et $\hat{h}(t)$.

5 Filtrage des signaux

Le code est disponible via https://github.com/a-mhamdi/cosnip/ → Python → sig-proc → filtering.ipynb

La convolution est une intégrale qui exprime le degré de chevauchement d'une fonction h lorsqu'elle est décalée sur une autre fonction x.

Par définition, une convolution $x * h$ se mesure par l'équation suivante :

$$x * h = \int_0^t h(t-\varsigma)x(\varsigma)d\varsigma.$$

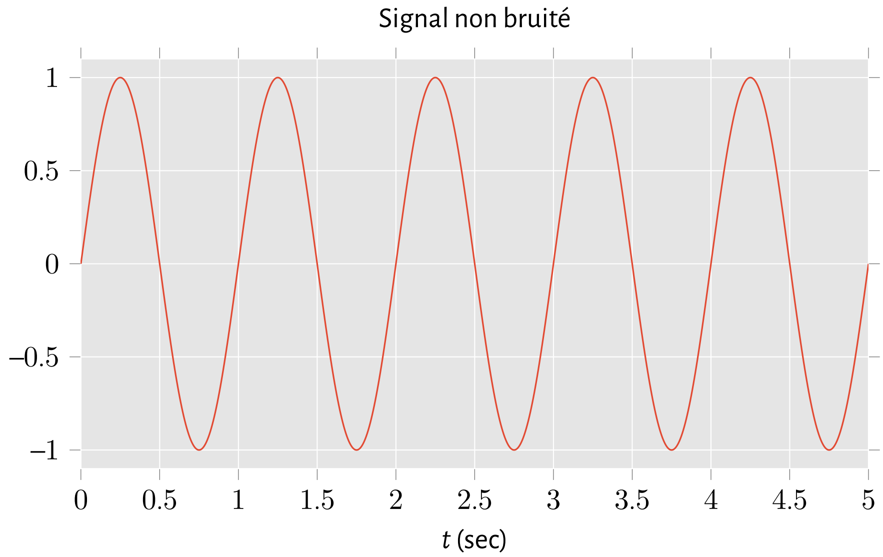
Ce produit est utilisé fréquemment pour le filtrage d'un signal contaminé par du bruit gênant ainsi la perception correcte de l'information. Un produit de convolution peut être vu comme une technique de calcul de moyenne à un instant t d'une fonction x pondérée par la fonction h et vice-versa.

On se propose de générer un signal sinusoïdal d'amplitude 1 et de fréquence 1 Hz que l'on note $x(t)$.

$$x(t) = \sin(2\pi f t), \quad \text{avec} \quad f = 1\,\text{Hz}.$$

[1]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

[2]:
```
t = np.linspace(0, 5.0, 1000)
x = np.sin(2 * np.pi * t)
plt.plot(t, x)
plt.title("Signal non bruité")
plt.xlabel("$t$ (sec)")
plt.grid()
plt.show()
```



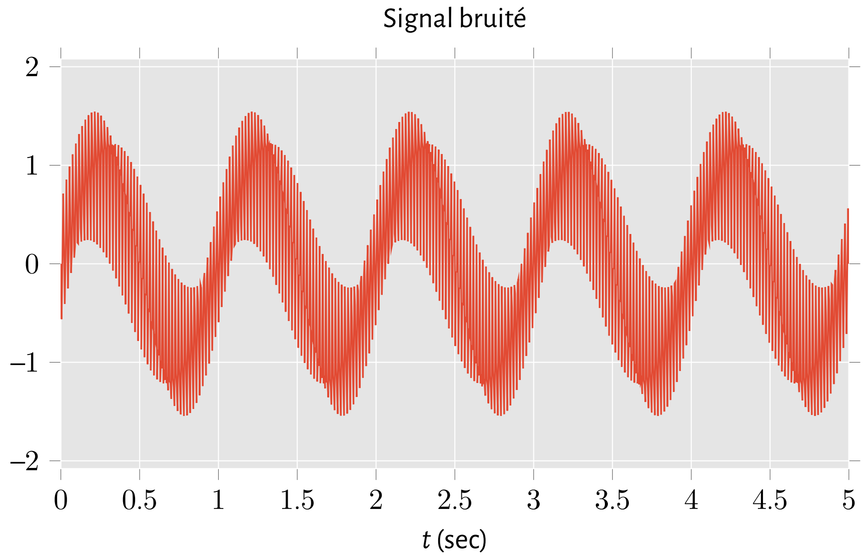
Nous synthétisons ici un exemple de bruit :

$$b(t) = -0.4 \sin\left(2\pi f_{b_1} t\right) + 0.6 \sin\left(2\pi f_{b_2} t\right), \quad \text{avec} \quad \begin{cases} f_{b_1} = 500\,\text{Hz}, \\ f_{b_2} = 750\,\text{Hz}. \end{cases}$$

Nous le rajouterons par la suite au signal d'origine $x(t)$ comme suit :

$$x_b = x(t) + b(t)$$

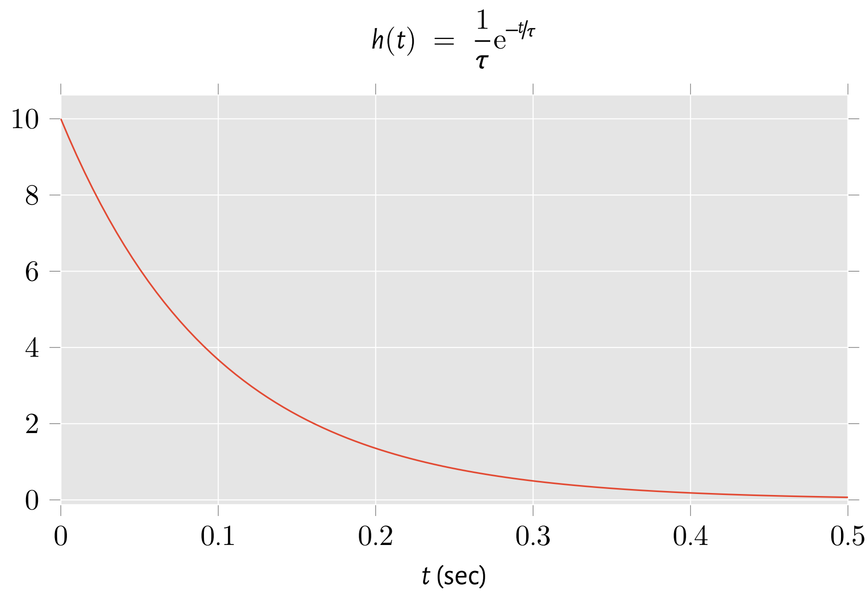
[3]:
```
# Générer un bruit
b = -0.4 * np.sin(1000 * np.pi * t) +0.6 * np.sin(1500 * np.pi * t)
x_b = x + b
plt.plot(t, x_b)
plt.title("Signal bruité")
plt.xlabel("$t$ (sec)")
plt.grid()
plt.show()
```



Le filtre à appliquer s'agit d'un passe-bas de réponse impulsionnelle :

$$h(t) \;=\; \dfrac{1}{\tau}\mathrm{e}^{-\dfrac{t}{\tau}}, \quad \text{avec} \quad \tau \;=\; 0.1 \text{ sec.}$$

[4]:
```
tau = 0.1; t_h = np.linspace(0, 5*tau, 100)
h = 1/tau * np.exp(-t_h/tau)
plt.plot(t_h, h)
plt.title(r"$h(t) \;=\; \dfrac{1}{\tau}\mathrm{e}^{-\dfrac{t}{\tau}}$")
plt.xlabel("$t$ (sec)")
plt.grid()
plt.show()
```

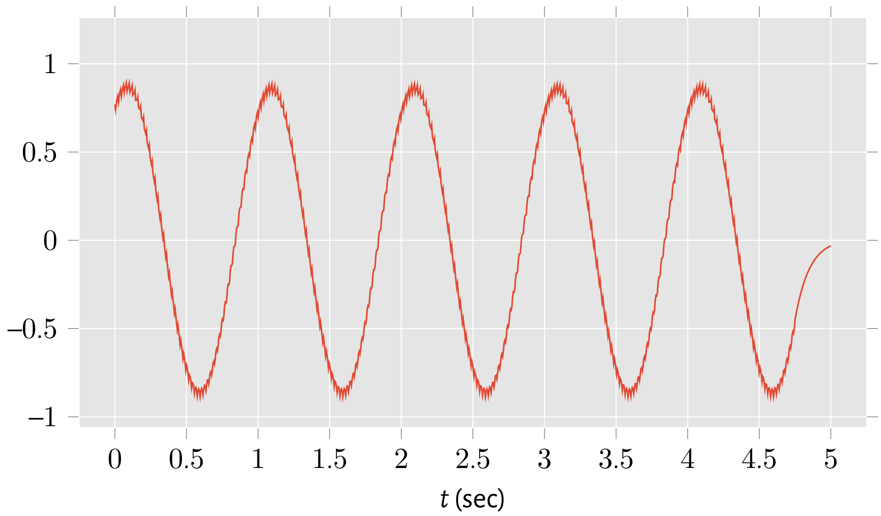


La sortie $\tilde{x}$ du filtre est le résultat du produit de convolution suivant :

$$\tilde{x}(t) = \int_0^t h(t-\varsigma)x(\varsigma)d\varsigma$$

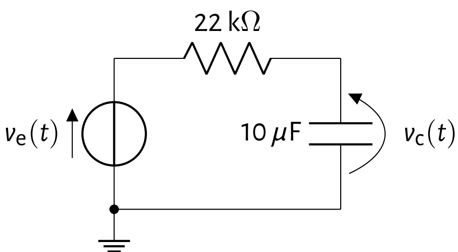
[5]:
```python
# # Normaliser le filtre : filt = h/Sigma_h
filt = h/h.sum()
x_f = np.convolve(x_b, filt, 'same')
plt.plot(t, x_f)
plt.title("Signal filtré")
plt.xlabel("$t$ (sec)")
plt.grid()
plt.show()
```

Signal filtré



Manipulation Nº 6 :

On se propose d'étudier la charge et la décharge dans le condensateur du circuit suivant :



La fonction h désigne sa réponse impulsionnelle.

a) Montrer que $h(t)$ s'écrit comme suit :

$$h(t) = {}^1\!/_\tau \mathrm{e}^{-t/\tau}, \quad \forall t \geq 0.$$

b) On applique en entrée de ce montage une entrée v_e décrite par :

$$v_e = \Gamma(t) - 2\Gamma_3(t) + \Gamma_5(t),$$

où Γ_α dénote la fonction d'Heaviside retardée, c.-à-d. $\Gamma(t-\alpha)$;

c) Pour $0 \leq t \leq 15$, représenter l'évolution de l'entrée $v_{\mathrm{e}}(t)$;

d) Calculer et tracer la sortie $v_{\mathrm{c}}(t)$.

Le présent fascicule s'adresse aux étudiants de la spécialité **Génie Électrique**, parcours **Électronique Industrielle**.

Nous traitons essentiellement les parties suivantes :

① Prise en main de `Python`

S'initier avec la programmation d'un calcul scientifique sous `Jupyter`.

② Convolution 1D & 2D

Combiner deux fonctions pour en former une nouvelle, exprimant ainsi comment les variations d'une fonction sont modifiées par l'autre.

③ Décomposition en série de Fourier

Décomposer un signal périodique en ses composants sinus et cosinus.

④ Transformée de Fourier

Représenter un signal non périodique dans le domaine fréquentiel.

⑤ Filtrage des signaux

Purifier un signal contaminé par du bruit via un produit de convolution.

`Python`; `Jupyter`; `NumPy`; `Matplotlib`; calcul scientifique ; convolution ; série de Fourier ; transformée de Fourier ; filtrage.