



Institut Mines-Télécom

Filtrage, transformée de Fourier et échantillonnage sous Python

Roland Badeau



Contexte académique } **sans modifications**

Voir page 9

Master Sciences et Technologies - Parcours ATIAM - UE FpA



1 Introduction à Python et filtrage

1.1 Préliminaires : utilisation de Python

Python est langage de programmation que nous utiliserons en mode interactif dans l'interface Spyder.

On peut exécuter une commande soit en la tapant directement dans le mode interactif soit en exécutant un script qui est une suite de commandes qui se trouvent dans un fichier dont l'extension est ".py". Le mieux est de taper les commandes dans l'éditeur de Spyder, sélectionner puis F9 pour exécuter les commandes sélectionnées. Cela permet de revenir sur les commandes et de les modifier en cas d'erreur. On peut aussi lancer toute une cellule avec Ctrl+Entrée.

Téléchargez le fichier TP_Intro_Python.zip depuis le Moodle du Master ATIAM. Il contient des données et du code Python. Vous travaillerez dans le répertoire résultant de sa décompression. Dans ce répertoire se trouve un fichier scratch_TP_Filtrage.py dans lequel la plupart des commandes sont déjà écrites. Lancez l'environnement Spyder. Dans Spyder ouvrez le fichier scratch_TP_Filtrage.py et exécutez la section de modules à importer.

Ce TP a pour objectif de prendre en main l'utilisation de Python et d'effectuer des opérations de manipulation simple sur les filtres (systèmes linéaires et invariants). Tout le TP peut se faire en mode interactif, Python permettant de définir des fonctions en mode interactif. Pendant la partie 1, vous êtes encouragés à taper les commandes et à en comprendre le résultat. En mode tutoriel le résultat d'une commande est affiché. Exemple

```
3+4; # Vous n'êtes pas obligés de taper les commentaires.
3+4 # ou comment utiliser 10^9 opérations par seconde pour calculer 7
```

1.1.1 Les bases

Sous Python un package indispensable est le package numpy abrégé en np dans ce qui suit (**import numpy as np** et cette commande est à taper dans chaque nouvelle console Python ouverte. Chaque tableau numpy peut avoir une ou plusieurs dimensions. Les matrices seront des tableaux de dimension 2. Les vecteurs de dimension 1. Cependant, pour traiter la multiplication matrice vecteur il est utile de considérer un vecteur comme un tableaux à deux dimension dont l'une vaut 1 (suivant si l'on veut une colonne ou une ligne). Il seront par défaut remplis de valeurs de type float64.

1.1.2 Générer une matrice

Il y a divers moyens de générer une matrice ou un vecteur

- **np.zeros((m,n))** : Génère une matrice de taille $m \times n$ pleine de zéros.
- **np.ones((m,n))** : Cette fois-ci la matrice est pleine de 1.
- **np.arange(a,c,b)** : Génère un vecteur ligne commençant à a et incrémenté de b tant que l'on est encore strictement plus petit que c . Si on tape (a, c) l'incrément par défaut est 1. Essayez

```
np.arange(0,7,3)
np.arange(1,7)
np.arange(0,1.01,0.1)
np.arange(0,1,0.1)
np.arange(5,0,-1)
```
- Entrer directement les valeurs entre crochets :

```
[1, 2 ,3] # ceci est une liste
np.asarray([1,2,3]) # ceci un tableau numpy
```

```

np.asarray([1,2,3]).reshape((1,3)) # tableau numpy sous forme de vecteur ligne
np.asarray([1,2,3]).reshape((1,-1)) # idem ci-dessus
np.asarray([[1,2],[3,4]])
np.asarray([1,2,3,4]).reshape((2,2)) # idem ci-dessus
np.asarray([1,2,3,4]).reshape((-1,2)) # idem ci-dessus

```

1.1.3 Opérations sur les tableaux

- **A+B** : Renvoie la somme des deux tableaux qui doivent être de même taille. **Exception** : Si A ou B est un scalaire, il est ajouté à toutes les entrées de l'autre matrices. L'opération "-" fonctionne de la même manière pour la soustraction.

```

np.ones((2,2))+np.ones((2,2))
np.ones((2,2))+1 # on ajoute 1 à toutes les entrées.

```
- **A@B** : Renvoie la multiplication des matrices¹ A et B qui doivent avoir des tailles respectives de $m \times n$ et $n \times l$. Le résultat est de taille $m \times l$. Notez que si $m = l = 1$ alors cette opération est un produit scalaire entre le vecteur ligne A et le vecteur colonne B .

```

A=np.asarray([[1,2],[ 3 ,4]])
B=np.asarray([[ -2,1],[ 1.5 , -0.5]])
A@B
B@A
A@np.asarray([[1, 0],[ 0 ,1]])
np.arange(1,20)@np.arange(1,20) # Bien que arange renvoie un tableau à une seule
                                # dimension, un vecteur prend la forme nécessaire
                                # pour permettre le produit matriciel

A@np.arange(3,5)
np.arange(3,5)@A
A.T
# A.T est la transposée de la matrice A

```
- **A*B** : Multiplication point par point. A et B doivent avoir la même taille. Même exception pour les scalaires.

```

Comparez
A*B
A@B

```
- **Fonctions unaires** : Numpy dispose d'un grand nombre de fonctions usuelles prédéfinies telles que : sin, cos, exp, tan, log ... Les appliquer à un tableau signifie appliquer la fonction à chaque élément du tableau

```

i=np.complex(0,1) # creer le complexe i
pi=np.pi # donner un nom plus commode à pi
np.exp(i*pi)
idx=np.arange(0,11)
np.exp(2*i*pi*0.123*idx) # Onde de Fourier sur Z de fréquence 0.123, entre 0 et 10
np.real(np.exp(2*i*pi*0.123*idx)) # Partie réelle de l'onde
plt.plot(np.real(np.exp(2*i*pi*0.123*idx))) # "plot" affiche son argument
# voir plus bas pour plus de précisions sur plt.plot

```
- **sum()** : Renvoie la somme des éléments d'un tableau. Si on précise un axe, on obtient la somme suivant cet axe

```

np.ones(10).sum()
np.arange(1,101).sum()
np.ones((2,3)).sum(axis=0) # somme colonne par colonne
np.ones((2,3)).sum(axis=1) # somme ligne par ligne
(np.arange(1,10)**2).sum() # somme du carré des entiers entre 1 et 9

```
- **np.concatenate** : Concaténation, par défaut empile les colonnes. L'argument est un n-uplet des tableaux à

1. Seulement valable pour les version de Python supérieures à 3.5.

```

concaténer.
np.concatenate( (np.ones( (2,3) ) , np.ones((2,3)) ) )
np.concatenate( (np.ones( (2,3)) , np.ones((2,3))), axis=1)
np.concatenate( (np.ones( (3,3) ) , np.ones((2,3)) ) )
np.concatenate( (np.ones( (3,3) ) , np.ones((2,3)) ), axis=1 ) # erreur
— A.reshape(-1) : Quelle que soit la forme de A, renvoie un vecteur monodimensionnel qui est la concaté-
  nation verticale des colonnes de A. A est parcourue ligne par ligne. Utile pour s'assurer que deux vecteurs ont
  la même forme.
  A=np.asarray([[1,2],[3,4]])
  A.reshape(-1)
  u=np.asarray([[1, 2, 2, 1]]) # matrice ligne
  v=np.asarray([[1],[2],[3],[4]]) # matrice colonne
  u*v # résultat inattendu
  u.reshape(-1)*v.reshape(-1) #on est s^
  ur de la forme c'est le produit point a point
— x**A : élève le scalaire x à la puissance chacune des entrées de A (fonctionne aussi si A est un scalaire)
  (-1/2)**np.arange(0,11) # les puissances de -1/2 de 0 à 10

```

1.1.4 Les variables

Les variables sont créées lors de leur première affectation, attention aux effets de bord.

```

u=np.ones((2,2))
v=u
v[0,0]=7
u
# u a été modifié par l'action sur v car v n'est qu'une copie de pointeur
v=u.copy() # voici la bonne manière de créer une copie indépendante de u
v[0,0]=8
u

```

Les indices des tableaux commencent à 0 : **h(0)** est le premier élément du vecteur **h**. **A[2,1]** est l'élément situé à la ligne numérotée 2 (troisième ligne) colonne numérotée 1 (deuxième colonne) de la matrice **A**. On peut aussi accéder à une collection de valeurs en indiquant une collection d'indices :

```

h=np.arange(0,10);
h[1:3]
h[1:10].sum() # somme les éléments numéro 1 à 9 de h
h[:-1].sum() #somme tous les éléments de h sauf le dernier
h[0::2].sum() # h[0]+h[2]+...+h[plus grand indice pair possible]
h[::-1] # renvoie le vecteur h à l'envers
h[2:5]=np.ones(3) # remplace certaines entrées de h par des 1

```

1.2 Réaliser une convolution et écouter un écho

Dans cette partie vous allez réaliser une convolution entre deux suites. Vous appliquerez cela à un signal sonore pour simuler un écho et écouterez le résultat. Certes, Python possède des packages capables de faire cela (scipy.signal), mais, pour vous exercer vous en réaliserez une vous même.

1.2.1 Création d'une fonction

On définit une fonction en Python de la manière suivante

```
def mafonction(a,b,c):
    tmp=a+b
    toto=c-b
    titi=tmp/toto
    return titi
```

Le résultat renvoyé par cette fonction est $(a + b)/(c - b)$. Les variables internes (tmp, toto et titi) seront oubliées après l'appel de cette fonction. Pour ce TP, nous définirons toutes les fonctions dans un seul fichier de travail pour ne pas multiplier les fichiers, mais un gros projet demanderait plus d'organisation.

1.2.2 Modélisation en vue de la convolution

L'opération de convolution entre deux suites h et u donne une suite, que l'on va noter v définie par

$$v_n = \sum_{m \in \mathbb{Z}} h_m u_{n-m}$$

Sur un ordinateur, et dans Python en particulier, on ne peut avoir que des suites à support fini. Ce que nous signifions par "opération de convolution" doit tenir compte de cette limitation. Quitte à décaler les suites, on peut toujours supposer qu'une suite à support fini a son support du type $\{0, \dots, N-1\}$ pour un certain entier N . Si deux suites ont leur support dans $\{0, \dots, N-1\}$ et $\{0, \dots, M-1\}$ respectivement, alors leur produit de convolution a son support dans $\{0, \dots, N+M-2\}$. Ainsi la convolution entre deux vecteurs de taille N et M sera un vecteur de taille $N+M-1$. Nous ferons la convention que, si un vecteur Python \mathbf{h} ² représente une suite h alors le premier élément du vecteur \mathbf{h} (i.e. $\mathbf{h}(0)$) est la valeur de h_0 et le support de la suite h est supposé être dans $\{0, \dots, \text{length}(\mathbf{h}) - 1\}$.

1.2.3 Valeur en un point

Avec ces conventions, écrire une fonction qui prend en entrée deux vecteurs \mathbf{h} et \mathbf{u} et un entier n et renvoie la valeur de leur convolution en n (i.e. $(h * u)_n$). S'il n'y avait pas de problèmes de bord cela s'écrirait

```
def valconv(h,u,n):
    a=len(h) # pour un tableau numpy len(v)==v.shape[0]
    b=len(u)
    u=u.reshape(-1) #s'assurer que u et v son de la meme forme
    h=h.reshape(-1) #
    idx=np.arange(0,a); # les indices pour h
    return (h[idx]*u[n-idx]).sum()
```

Après avoir bien compris l'expression dans `"(...).sum()"` proposer une manière de remédier au problème des bords. On remarquera que pour que $\mathbf{h}(\mathbf{m})$ (m sera un élément du vecteur `idx`) intervienne dans la somme sans poser de problème d'indice, il faut et il suffit d'avoir

$$\begin{aligned} 1 \leq m+1 \leq a & \text{ pour que } \mathbf{h}(\mathbf{m}) \text{ existe} \\ 1 \leq n-m+1 \leq b & \text{ pour que } \mathbf{u}(\mathbf{n-m}) \text{ existe} \end{aligned}$$

2. à partir de maintenant, on écrira en gras des variables Python et en italique leur représentation mathématique

En déduire la formule correcte pour le vecteur **idx** en utilisant `min(x,y)` et `max(x,y)` qui renvoient le minimum et le maximum entre x et y .

Testez votre fonction avec

```
valconv(np.ones(3),np.ones(2),0) # doit renvoyer 1
valconv(np.ones(3),np.ones(2),2) # doit renvoyer 2
```

1.2.4 Convolution complète

Modifiez votre fonction pour qu'elle renvoie le vecteur entier de la convolution (on supprime le paramètre n). Pour cela on utilisera une boucle

```
out = np.zeros(a+b-1)
for n in range(a+b-1): #La boucle va être parcourue avec n parcourant le vecteur 0,1,...
    idx=... # Formule trouvée précédemment
    out[n]= (h[idx]*u[n-idx]).sum()
return out
```

Testez votre fonction avec de petites suites dont vous calculerez la convolution à la main pour vérifier.

1.2.5 Ecoute d'un écho

Charger un son dans Python à l'aide de la fonction

```
[x,Fe]=sf.read('piano.wav')
```

Cela va charger une variable **x** qui contient un son. La valeur de Fe est la fréquence d'échantillonnage. La commande suivante permet d'écouter un son :

```
play(x,Fe)
```

Que donne la commande suivante ?

```
play(x,Fe/2)
```

On peut modéliser un écho par le fait que le son s'ajoute à lui même avec un certain retard, si u représente le signal sonore original, le signal reçu sera du type

$$v_n = u_n + 0.8u_{n-t_1}$$

Ceci signifie que le son rebondit sur un obstacle, perd 20% de son amplitude et arrive retardé de t_1 . On constate que cette modélisation revient à réaliser la convolution du son u avec une réponse impulsionnelle h définie par $h_0 = 1$, $h_{t_1} = 0.8$ (tous les autres termes de h sont nuls).

Créer le vecteur **h** correspondant à un retard approximatif de 0,01 seconde (on rappelle que le son utilisé ici est échantillonné à Fe échantillons par seconde). Convoler, avec votre programme, ce vecteur **h** avec le signal sonore **y** (stockez le résultat dans une variable).

À combien estimez-vous le temps de calcul pris par votre programme ?

Utilisez la commande

```
z=scipy.convolve(h,y);
```

Cette fonction est une fonction Python qui fait ce que fait votre programme de convolution. Que constatez-vous sur le temps de calcul ?

Écoutez le son **z**. (`play(z,Fe)`)

Augmentez le temps de l'écho à 0,1 seconde, quel impression cela donne-t-il ? En termes de distance parcourue par le son, que représentent 0,1 et 0,01 secondes, cela vous aide-t-il à interpréter l'effet de l'écho ?

1.3 Filtre rejecteur

Dans la suite on utilise la commande Python `lfilter` (qui doit être définie par `lfilter=scipy.signal.lfilter`).

Après avoir consulté l'aide par `help lfilter`, dire comment utiliser `lfilter` pour effectuer une convolution d'un signal contre un RIF (réponse impulsionnelle finie).

Dans la suite nous allons voir comment utiliser `lfilter` pour effacer une fréquence parasite dans un son. Pour cela nous créons une version polluée de la variable **y** :

```
n=np.arange(len(y)); #le temps discret que dure le signal y
f0=1261;
yb=y+0.1*cos(2*pi*f0/Fe*n); #on ajoute une onde parasite
```

Écouter le son **y_b** à la fréquence **Fe**. Quelle est la fréquence en Hz et fréquence réduite de l'onde parasite ?

Donner la TZ de la RI du filtrage qu'effectue la fonction `rejette1` ? Écouter le résultat de ce filtrage sur le signal **y_b**. Cela vous paraît-il satisfaisant ?

Donner la TZ de la RI du filtrage qu'effectue la fonction `rejette2` ? Écouter le résultat sur le signal **y_b**.

Pour une valeur de **rho** proche de 1, placer les zéros et les pôles des TZ précédentes dans le plan complexe. Utiliser ces positions pour tracer (approximativement) le module des deux TFtD correspondantes (sur l'intervalle $[-1/2, 1/2]$). Conclure quand à la qualité comparée des deux filtrages.

Pourquoi rho (paramètre de `rejette2`) doit-il toujours être strictement plus petit que 1 ?

En transmettant à la fonction `rejette2` une fréquence **f0=1267** et un paramètre **rho** très très proche de 1, conclure au compromis nécessaire dans le choix de **rho**.

2 Transformation de Fourier et échantillonnage

2.1 Préliminaires

Dans cette partie nous illustrons les propriétés de la transformation de Fourier. Nous manipulons la fonction `fft` pour visualiser des transformées de Fourier et bien en comprendre le fonctionnement. Nous verrons en particulier comment l'algorithme Fast Fourier Transform peut être utilisé pour calculer rapidement des choses qui, a priori, n'ont pas de rapport avec la transformation de Fourier.

Nous abordons ensuite les problèmes que pose l'échantillonnage des signaux, en écoutant l'effet qu'a un sous-échantillonnage sur un son.

Dans le répertoire résultant de la décompression du fichier TP_Intro_Python.zip, se trouve un fichier `scratch_TP_Fourier.py` dans lequel la plupart des commandes sont déjà écrites.

2.2 Transformation de Fourier

Sur ordinateur, on ne manipule que des suites finies de nombres. En cours, nous avons vu diverses transformations de Fourier. A priori, seule la TFD (contrairement à la TFTD, TFTC et coefficients de Fourier) peut être calculée sur ordinateur.

On fera la convention que, si h dénote une suite à support fini, alors \mathbf{h} dénote le vecteur Python qui la représente et on a $\mathbf{h}(0) = h_0$ car Python indexe les éléments d'un tableau à partir de 0.

Si \mathbf{h} est vecteur Python et M un entier. Alors³ :

```
fft(h) # renvoie un tableau de même taille que h et qui en est la TFD
fft(h,M) # Si M est plus grand que la taille de h, renvoie la TFD de h complété par
# des zéros pour atteindre la taille M.
# Si M est plus petit que la taille du vecteur h alors: renvoie la TFD
# du vecteur h[0:M] (i.e. les M premiers échantillons de h)
```

On rappelle que la TFD d'une séquence finie $h_0, \dots, h_{N-1}0\dots$ où l'on a complété la séquence h_n par des zéros pour obtenir un signal de taille M est exactement l'échantillonnée de la TFTD de la séquence h_n , $n \in \mathbb{Z}$ aux fréquences k/M pour k allant de 0 à $M - 1$.

Il ne faut pas confondre FFT et TFD. La FFT est un algorithme rapide de calcul de la TFD. Numpy fait cette confusion, c'est dommage. L'algorithme de la FFT est particulièrement rapide pour des vecteurs de taille une puissance de 2. Il s'exécute en un temps proportionnel à $N \log(N)$ où N est la taille du signal.

- Créer deux vecteurs de longueur 100 qui sont deux ondes de Fourier de fréquence 0.1 et 0.123. Et observer leur TFD par la commande `plot(abs(fft(x)))`;⁴ (\mathbf{x} est le vecteur). Quelles différence remarquez-vous entre les deux cas ? Pourquoi ? (souvenez-vous de la différence entre une onde de Fourier sur \mathbb{Z} et une onde de Fourier sur $\{0, \dots, N - 1\}$)
- Pouvez-vous expliquer la position exacte du pic dans TFD du signal de fréquence 0.1 (=10/100)? (en bougeant le curseur on a en bas de la figure l'affichage de x, y ⁵). Expliquer aussi la position du second pic qui apparaît dans le cas où l'on trace la TFD de la partie réelle de l'onde de fréquence 0.1

3. En fait `fft` est un raccourci pour `numpy.fft.fft`

4. on peut aussi utiliser `stem` à la place de `plot` pour de petits signaux

5. Pour cela il faut configurer Spyder pour effectuer les affichages de figures en extérieur Outils->Préférences->Console iPython->Graphique et relancer une nouvelle console iPython

- (`plot(abs(fft(real(x))))`). Pour un signal réel, il est inutile de tracer toute la TFD, seule la moitié de celle-ci suffit le reste se déduisant par symétrie hermitienne.
- La commande `plot(x,y)` trace la courbe représentant les `y` en fonction de `x`. Pour avoir un axe des abscisses bien indexé lors de la visualisation d'une TFD il convient de faire
`t=np.arange(0,100); #vecteur 0,N-1 pour N=100`
`nu=0.1; #fréquence`
`onde=exp(2*i*pi*nu*t); #création de l'onde de fréquence nu`
`plot(t/100,abs(fft(onde))); #tracé correctement indexé de la TFD`
 - Vérifier sur des exemples simples que le produit de convolution est bien transformé en multiplication en Fourier (exemple, la séquence [1, 2, 3] convoluée avec [1, 1, 0] dont on calcule la convolution circulaire à la main). Vérifier également que la multiplication par une onde de Fourier conduit à une translation de la TFD.
 - Enfin, la fonction `ifft(y)` réalise la TFD inverse de `y`. Que donnent les commandes suivantes ?
`ifft(fft([1, 1, 1]))`
`ifft(fft([1, 1, 1],4))`
`ifft(fft([1, 2, 3]).*fft([1, 1, 0]))-[4, 3, 5]`
 Comme vu en cours la formule de TFD inverse est très proche de celle de la TFD. L'algorithme qui la calcule est aussi rapide que la FFT.
 - **Simuler une convolution sur \mathbb{Z} grâce à une convolution circulaire :** Si h_0, \dots, h_{N-1} et g_0, \dots, g_{N-1} sont deux séquences de taille N et si `h` et `g` sont les vecteur (ligne) Python qui les représentent. Que donne la séquence Python suivante ? (dans la variable `out`)
`N=len(h); # supposé égale à length(g)`
`hz=np.concatenate((h, ...)) # compléter par de zéros`
`gz=np.concatenate((h, ...))`
`out=ifft(fft(hz)*fft(gz)) #TFD inverse de produit de TFD`
`out=out(1:2*N-1)`
 Cette méthode permet en particulier de calculer le produit de deux polynôme de degré $N - 1$ en un temps proportionnel à $N \log(N)$ alors qu'il faudrait de l'ordre de N^2 opérations en appliquant simplement la formule qui donne les coefficients du polynôme produit en fonction des coefficients des polynômes d'origine.

2.3 Fréquence en Hz, fréquence réduite

Si $h_n = f(nT_e)$ est le résultat de l'échantillonnage d'une fonction f à la fréquence $F_e = 1/T_e$ alors une onde de Fourier sur \mathbb{R} de fréquence ν va donner

$$h_n = e^{2i\pi\nu n T_e} = e^{2i\pi \frac{\nu}{F_e} n}$$

soit une onde de Fourier à la fréquence $\frac{\nu}{F_e}$. Ainsi, la bonne normalisation pour l'axe des abscisses, si on connaît la fréquence d'échantillonnage F_e , est

```
Fe=1000; Te=1/Fe; t=np.arange(0,300)*Te; #fréquence d'échant. 1000Hz. t qui représente le t
# pour 300 échantillons
xi=100; #fréquence en Hertz
h=np.cos(2*pi*xi*t); # équivalent à cos(2*pi*(nu/Fe)*(0:299))
fh=fft(h)
fh=fh[1:151] # comme h est réel inutile de garder plus de la moitié de la TF
plot(np.arange(0,151)/300*Fe, abs(fh)); # on indexe en vraies fréquences.
# la position 150 dans la TFD donne Fe/2=500Hz, ici.
```

Générer une onde de fréquence 0,01 et longue de 10000 échantillons ($n \mapsto e^{2i\pi\nu n}$, ν est la fréquence). Écoutez sa partie réelle(`np.real(u)`) par

```
play(real(u),44100) # Le son est joué à 44100 échantillons par seconde  
play(real(u),22050)# le son est joué à 22050 échantillons par seconde
```

Quelles fréquences en Hz ($1\text{Hz}=1/\text{seconde}$) percevez-vous dans les deux cas ?

On dit que la fréquence en Hz est la fréquence réelle (ce qui nécessite de connaître la fréquence d'échantillonnage) alors que la fréquence 0,06 est la fréquence réduite (qui est toujours, pour un signal défini sur \mathbb{Z} , entre $-1/2$ et $1/2$).



Contexte académique } sans modifications

Par le téléchargement ou la consultation de ce document, l'utilisateur accepte la licence d'utilisation qui y est attachée, telle que détaillée dans les dispositions suivantes, et s'engage à la respecter intégralement.

La licence confère à l'utilisateur un droit d'usage sur le document consulté ou téléchargé, totalement ou en partie, dans les conditions définies ci-après, et à l'exclusion de toute utilisation commerciale.

Le droit d'usage défini par la licence autorise un usage dans un cadre académique, par un utilisateur donnant des cours dans un établissement d'enseignement secondaire ou supérieur et à l'exclusion expresse des formations commerciales et notamment de formation continue. Ce droit comprend :

- le droit de reproduire tout ou partie du document sur support informatique ou papier,
- le droit de diffuser tout ou partie du document à destination des élèves ou étudiants.

Aucune modification du document dans son contenu, sa forme ou sa présentation n'est autorisée.

Les mentions relatives à la source du document et/ou à son auteur doivent être conservées dans leur intégralité.

Le droit d'usage défini par la licence est personnel et non exclusif. Tout autre usage que ceux prévus par la licence est soumis à autorisation préalable et expresse de l'auteur : sitepedago@telecom-paristech.fr