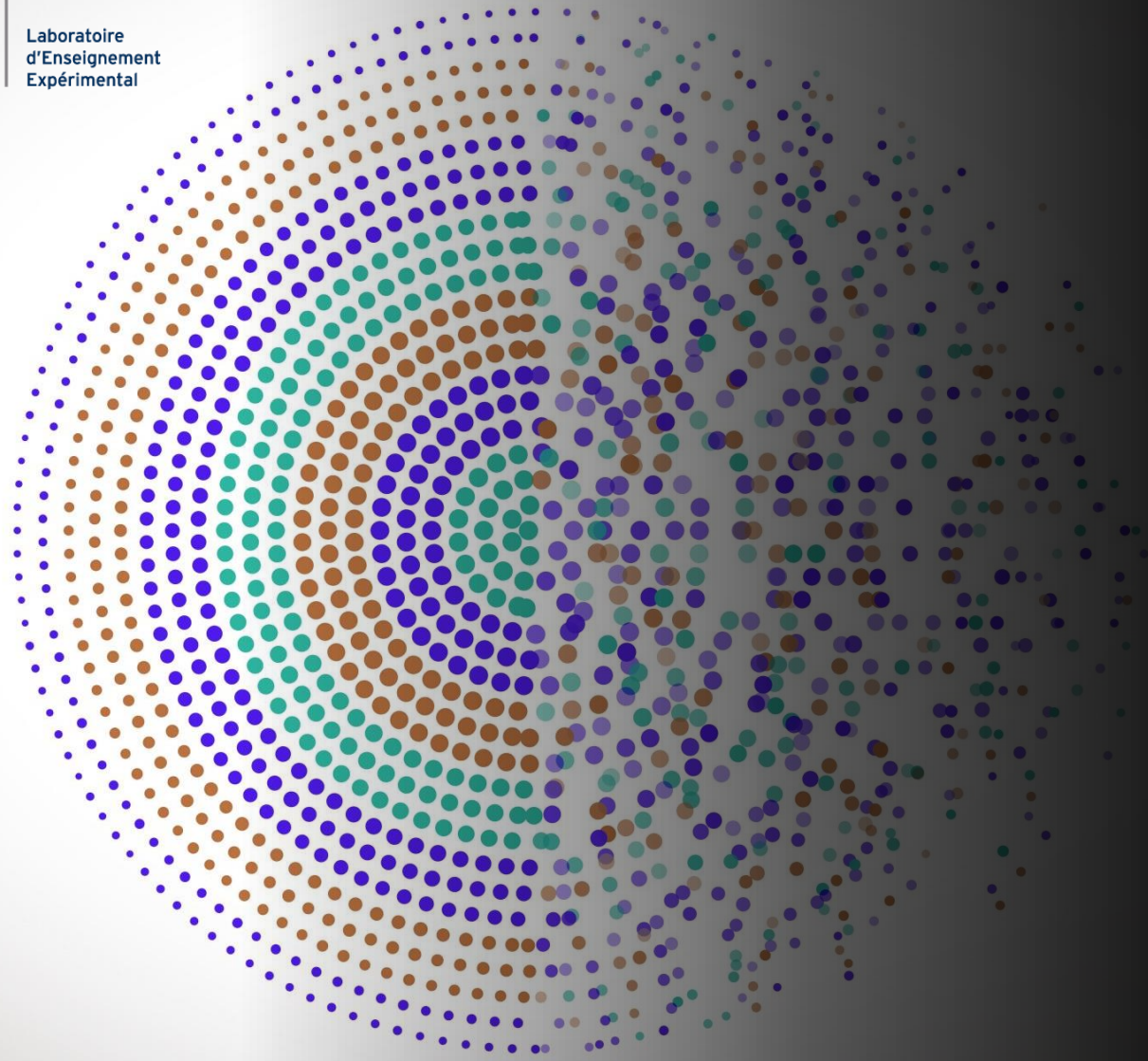
INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech | LEnsE Laboratoire d'Enseignement Expérimental



Traitement 1D

Modulation AM

Outils Numériques / Semestre 5
Institut d'Optique / B3_0

Déroulement du bloc AM

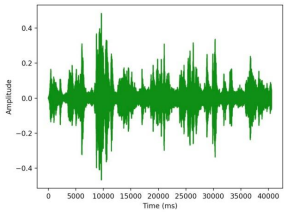
3 séances introductives (2h/séance)

2 blocs de 4 séances (2h/séance)

- Sur machine
- En binôme
- 2 encadrant.es par séance

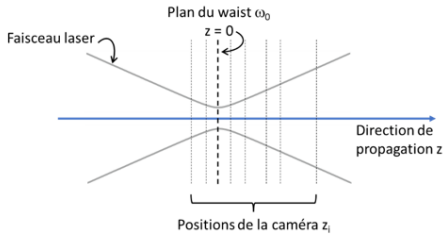
Bloc AM : Traitement de données 1D

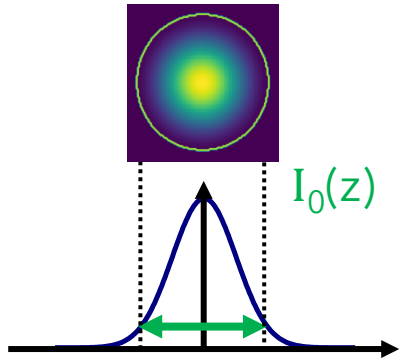
Problème 1 : signal modulé en amplitude / acquisition numérique



Bloc Laser : Traitement de données 2D

Problème 2 : images d'un faisceau LASER en différents points d'un chemin optique

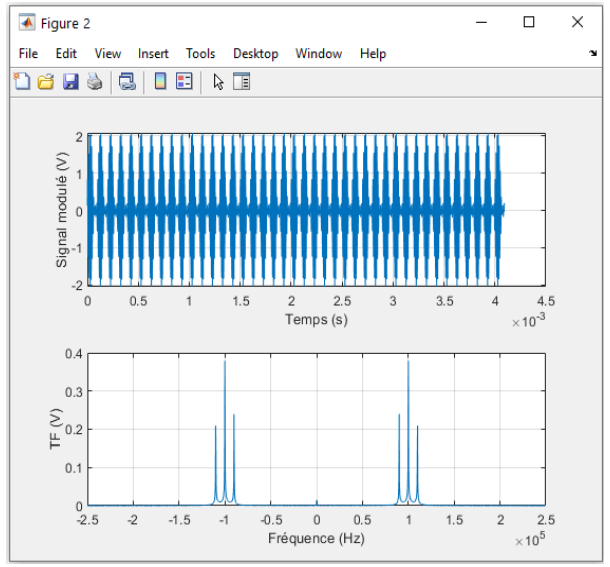




INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech | LEnsE Laboratoire d'Enseignement Expérimental

Contexte

- **Instrumentation numérique**
  - Acquisition de données
  - Sauvegarde de données
  - Analyse des données
  - Traitement des données



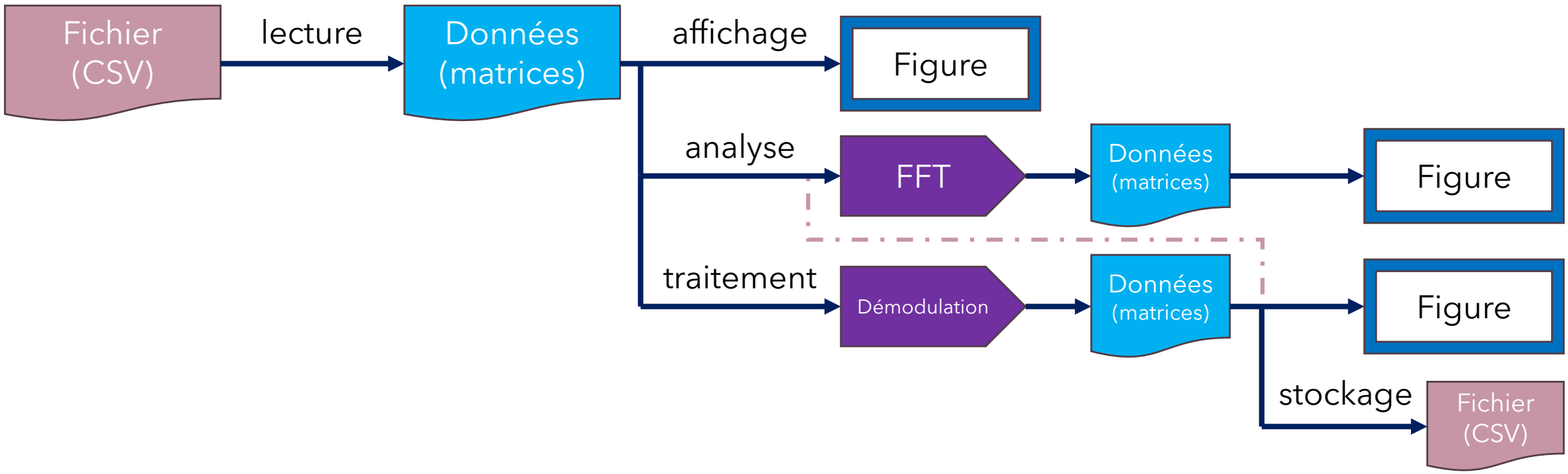
Signaux modulés en amplitude

Transformée de Fourier

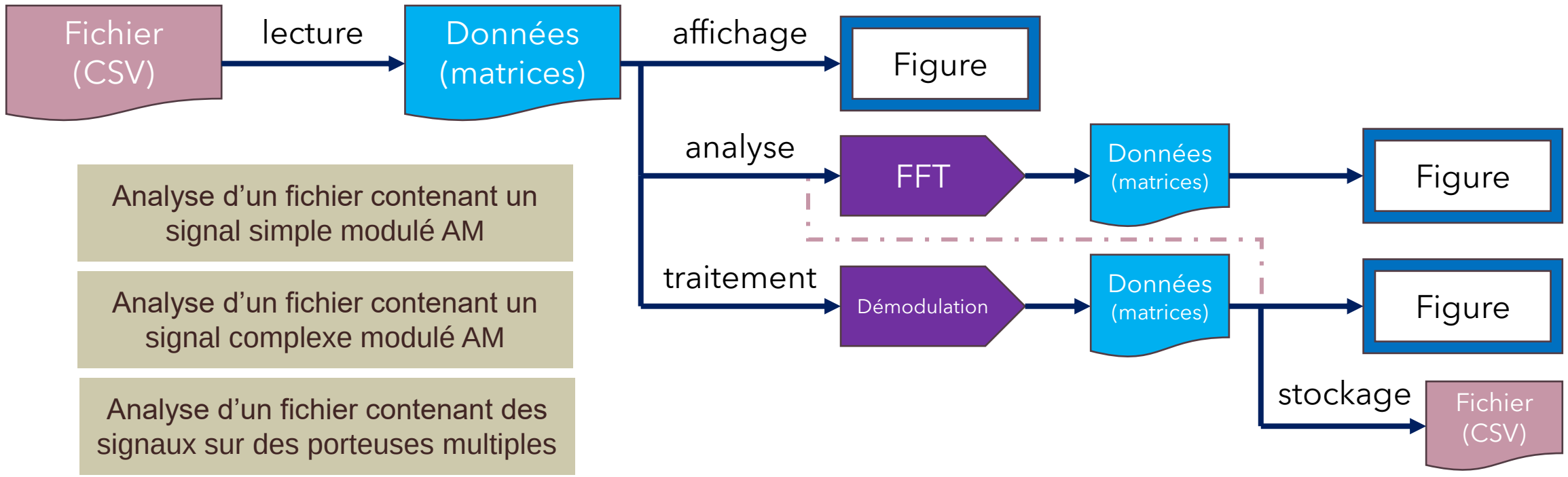
Données initiales / Démarche

Fichier (CSV)

Etapes pour l'analyse



Etapes pour l'analyse



Analyse d'un fichier contenant un signal simple modulé AM

Analyse d'un fichier contenant un signal complexe modulé AM

Analyse d'un fichier contenant des signaux sur des porteuses multiples

Travail à réaliser

- **Etape 1 : Afficher des données provenant d'un fichier**
  - Lire un fichier texte / tableur
  - Afficher les signaux contenus dans le fichier
- **Etape 2 : Calculer, afficher et analyser le spectre du signal**
  - Comprendre les données obtenues par le calcul
  - Afficher le spectre en recréant l'axe fréquentiel
  - Déterminer les informations utiles
- **Etape 3 : Démoduler un signal modulé et l'afficher**
- **Etape 4 : Générer de nouveaux signaux modulés et les démoduler**

Quelques fonctions intéressantes

- lire des fichiers CSV
  - numpy .genfromtxt
  - pandas .read_csv
- créer de vecteurs / matrices
  - numpy .linspace .logspace
  - numpy .ones .zeros
- afficher des figures
  - pyplot .figure .plot .title .xlabel .ylabel .legend
- calculer la FFT
  - numpy .fft.fft .fft.fftshift .fft.fftfreq
- transcodage / Numpy types
  - numpy .frombuffer .astype
- encodage B64
  - base64 .b64encode .b64decode
- encodage WAV
  - scipy.io .wavfile.read .write

Fichiers à analyser

- B3_data_01.csv
  - Issu d'un oscilloscope VoltCraft
  - Modulante sinusoïdale
- B3_data_02.txt
  - Format de données binaire 64
  - Modulante sinusoïdale
  - Fichier sonore / 24 kHz / 16 bits
- B3_data_03.txt
  - Format de données binaire 64
  - Multi-porteuses sinusoïdales
  - Fichier sonore / 160 kHz / 16 bits

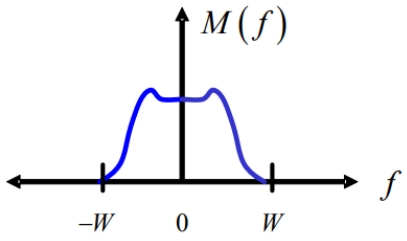
```
with open('B3_data_03.txt', 'rb') as file_to_decode:
    binary_file_data = file_to_decode.read()
    data = np.frombuffer(base64.b64decode(binary_file_data), np.int16)
```

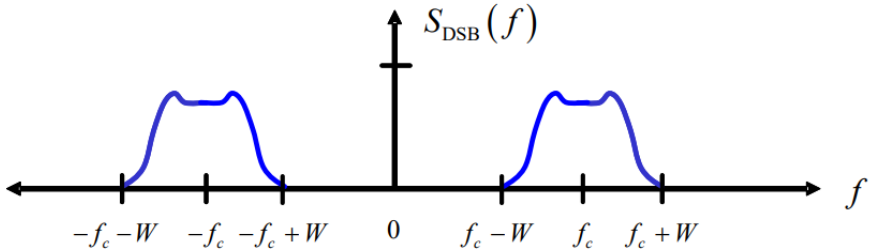
http://lense.institutoptique.fr/ONIP/

Rappels sur la modulation d'amplitude

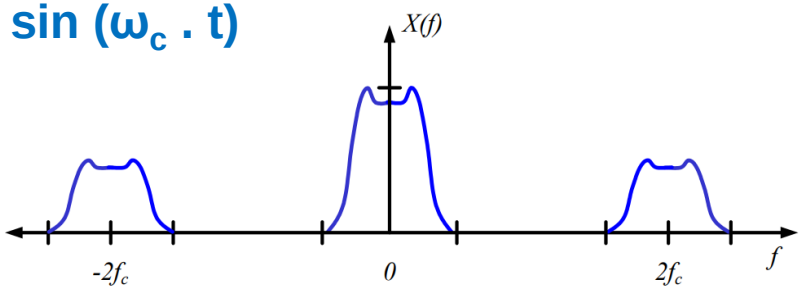
http://wcours.gel.ulaval.ca/2017/a/GEL3006/default/5notes/index.chtml

m(t) × p(t) = A_c . sin (ω_c . t)





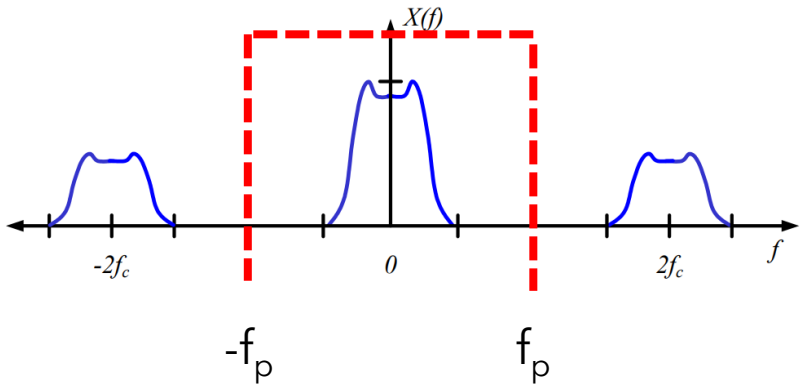
× p(t) = A_c . sin (ω_c . t)



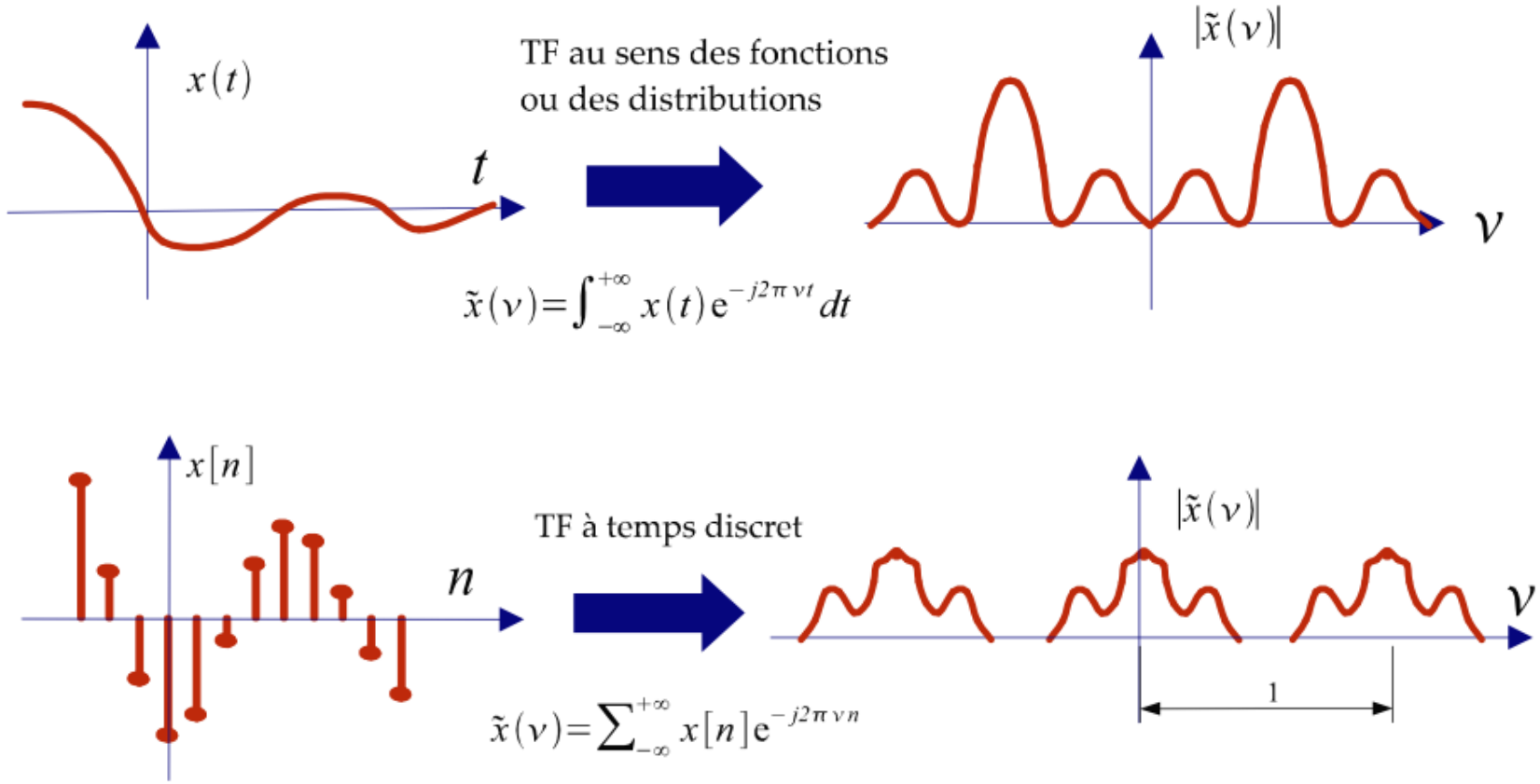
$$s(t) = (K . m(t) + 1) . p(t)$$

Filtrage

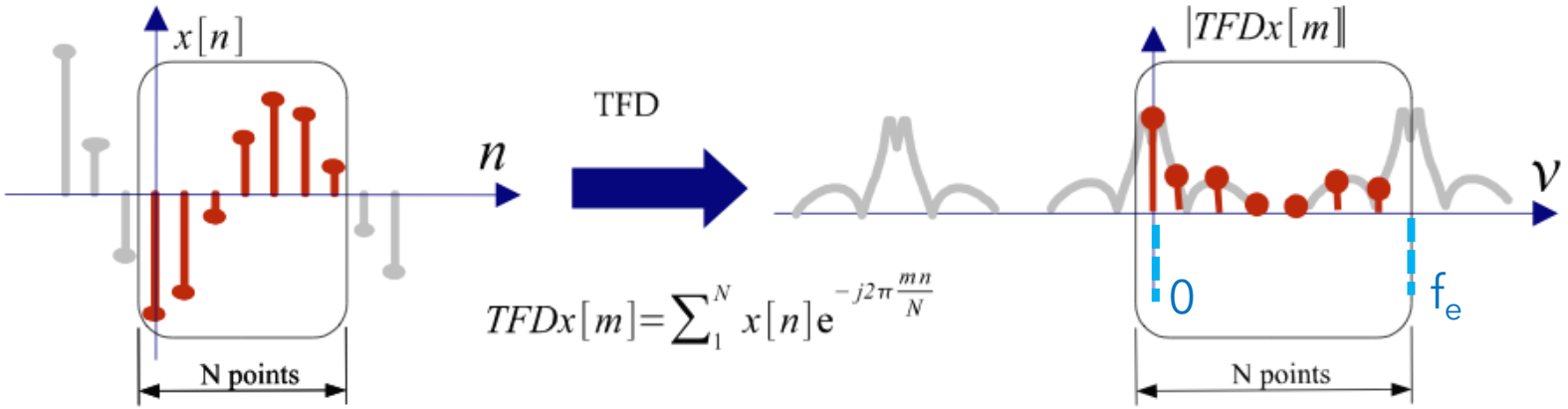
http://wcours.gel.ulaval.ca/2017/a/GEL3006/default/5notes/index.chtml



Rappel sur la Transformée de Fourier



Rappel sur la FFT

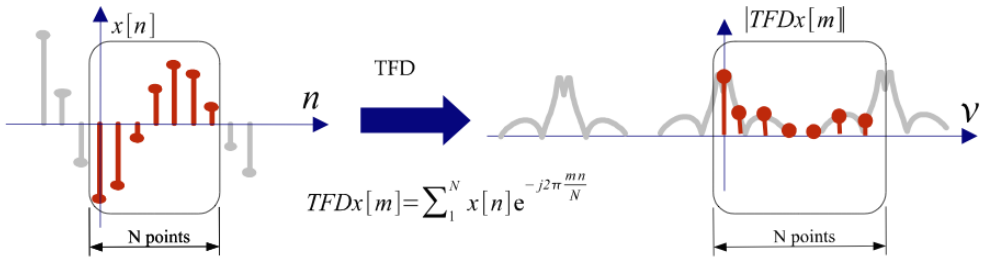


$$TFDx[m] = \sum_{1}^{N} x[n] \mathrm{e}^{-j2\pi \frac{mn}{N}}$$

numpy .fft.fft .fft.fftshift

$$A_k = \sum_{m=0}^{n-1} a_m \exp\left\{-2\pi i \frac{mk}{n}\right\} \qquad k = 0, \ldots, n-1.$$

Rappel sur la FFT



numpy .fft.fft .fft.fftshift

$$A_k = \sum_{m=0}^{n-1} a_m \exp\left\{-2\pi i \frac{mk}{n}\right\} \qquad k = 0, \ldots, n-1.$$

If **A = fft(a, n)**, then **A[0]** contains the zero-frequency term
Then **A[1:n/2]** contains the positive-frequency terms,
and **A[n/2+1:]** contains the negative-frequency terms
For an **even number** of input points, **A[n/2]** represents
both positive and negative Nyquist frequency,
For an **odd number** of input points, **A[(n-1)/2]** contains the largest positive frequency,
while **A[(n+1)/2]** contains the largest negative frequency.

Format Binaire 64

- Codage ASCII
  - 1 caractère codé sur 8 bits / 1 octet
- Codage entier
  - 1 entier sur 4 octets
- Codage Base 64
  - 1 donnée sur 6 bits : 4 données sur 3 octets

base64 .b64encode .b64decode

```
import base64

encoded = base64.b64encode(b'data to be encoded')
b'ZGF0YSB0byBiZSBlbmNvZGVk'

data = base64.b64decode(encoded)
b'data to be encoded'
```

```
with open('B3_data_03.txt', 'rb') as file_to_decode:
    binary_file_data = file_to_decode.read()
    data = np.frombuffer(base64.b64decode(binary_file_data), np.int16)
```

Conversion en signaux sonores

The Canonical WAVE file format

endian	File offset (bytes)	field name	Field Size (bytes)	
big	0	ChunkID	4	The "RIFF" chunk descriptor
little	4	ChunkSize	4	The Format of concern here is "WAVE", which requires two sub-chunks: "fmt " and "data"
big	8	Format	4	
big	12	Subchunk1 ID	4	The "fmt " sub-chunk
little	16	Subchunk1 Size	4	describes the format of the sound information in the data sub-chunk
little	20	AudioFormat	2	
little	22	NumChannels	2	
little	24	SampleRate	4	
little	28	ByteRate	4	
little	32	BlockAlign	2	
little	34	BitsPerSample	2	
big	36	Subchunk2ID	4	The "data" sub-chunk
little	40	Subchunk2Size	4	Indicates the size of the sound information and contains the raw sound data
little	44	data	Subchunk2Size	

scipy.io .wavfile.read .write