

# TP Numérique

V. Laurain

`vincent.laurain@univ-lorraine.fr`


# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Fonctions nécessaires . . . . .	3
1.1.1	Format d’affichage . . . . .	3
1.1.2	Forcer un format . . . . .	3
1.1.3	Afficher une courbe . . . . .	4
1.1.4	Des fonctions utiles pour ce TP . . . . .	4
<b>2</b>	<b>Codage du son</b>	<b>4</b>
2.1	Virgules flottantes . . . . .	4
2.2	Passage en entiers . . . . .	4
<b>3</b>	<b>Codage des images</b>	<b>6</b>
<b>4</b>	<b>Opérations en virgule flottante</b>	<b>7</b>
<b>5</b>	<b>Si vous avez le temps: Un peu de Pop-Art</b>	<b>8</b>

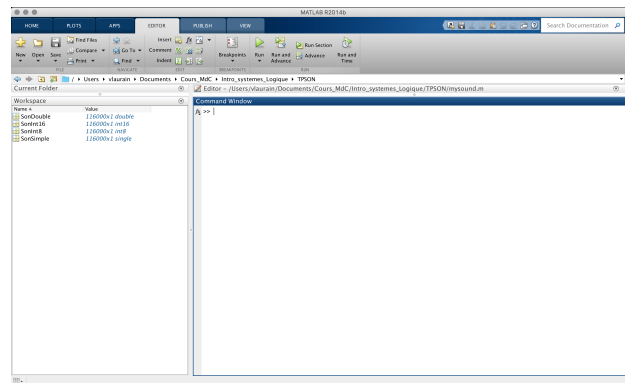
# 1 Introduction

Dans ce TP, nous allons voir comment fonctionne une machine en ce qui concerne le stockage des nombres et nous allons également apprécier les différentes erreurs liées aux approximations induites par les différents codages. Nous allons travailler sous un logiciel qui s'appelle Matlab et qui est très utilisé en recherche et en développement car il permet comme son nom l'indique d'effectuer de manière très simple du calcul Matriciel (même si nous n'en avons pas besoin dans le cadre de ce TP).

Tout d'abord vous allez devoir:

- Télécharger certains fichiers sous ARCHE ( tout le contenu de TP num)
- Placer les fichiers dans un répertoire de votre choix (de préférence sous votre dossier étudiant)
- Ouvrir le logiciel Matlab (Exécuter puis Matlab)
- Cliquer sur l'icône  et indiquez l'endroit où vous avez placé les fichiers

Vous devriez avoir devant vous quelque chose qui ressemble à cela:



Nous voici prêts à commencer

## 1.1 Fonctions nécessaires

Voici toutes les fonctions dont vous aurez besoin pour mener à bien ce TP:

### 1.1.1 Format d'affichage

Sous Matlab il est possible d'afficher un nombre sous différents formats. Nous en utiliserons principalement 2. Si vous tapez:

- **format short**: Matlab affiche les nombres avec leur valeur.
- **format hex**: Matlab affiche le codage des nombres en hexadécimal.

### 1.1.2 Forcer un format

Si vous avez une variable appelée **a**, alors vous pouvez la forcer à différents formats:

- **int8(a)**: Mettra la valeur de **a** au format **int8** (nombre signé entier à 8 bits)
- **uint8(a)**: Mettra la valeur de **a** au format **uint8** (nombre non signé entier à 8 bits)
- **single(a)**: Mettra la valeur de **a** au format **single**: nombre à virgule flottante sur 32 bits

- **double(a)**: Mettra la valeur de **a** au format **double**: nombre à virgule flottante sur 64 bits. C'est la précision maximale proposée par Matlab. Par défaut si vous tapez **a= 45.5** il l'enregistrera en format **double**.

Les formats existants sont :

- Les nombres non signés: **uint8,uint16,uint32,uint64**
- Les nombres signés: **int8,int16,int32,int64**
- Les virgules flottantes: **single, double**

### 1.1.3 Afficher une courbe

Dans ce TP, nous allons manipuler des signaux. Un signal, au sens informatique, que des valeurs successives stockées. Elles sont placées l'une à côté de l'autre dans la mémoire de l'ordinateur. Si vous avez un signal **S**, alors tapez simplement **S** pour voir défiler toutes ses valeurs. Si vous voulez voir un affichage graphique il faudra taper:

- **figure;plot(S)** : crée une nouvelle figure et affiche **S**
- **hold on;plot(S2)** : Affichera **S2** sur la même figure que **S**.

### 1.1.4 Des fonctions utiles pour ce TP

Dans le cadre de ce TP nous allons utiliser les fonctions :

- **playsound(S)** : joue un son.
- **figure;imshow(I,[])** : Affiche une image de sorte que la valeur la plus basse est un pixel noir et la valeur la plus haute est un pixel blanc.

## 2 Codage du son

**Tapez la commande** **load('Son.mat')**. Vous avez dans votre espace de données deux signaux appelés **SonDouble** et **SonSimple** qui correspondent à un son. Le type de stockage des nombres est indiqué à droite.

### 2.1 Virgules flottantes

- **Quel est le type de chaque signal?**

Puisqu'en cours nous n'avons vu que les virgules flottantes à 32 bits intéressons nous à **SonSimple**:

- tapez **SonSimple**. **Quelle est la dernière valeur de SonSimple?**
- passez en affichage hexadécimal (**format hex**)
- Réaffichez **SonSimple**
- **Vérifiez que le codage est bien une virgule flottante à 32 bits.**
- repassez en **format short**

Ecoutez maintenant les deux sons grâce à la fonction **playsound**.

- **Entendez vous une différence?**

Remarquez que le stockage en **double** nécessite 2 fois plus de places que le stockage en **single**.

## 2.2 Passage en entiers

Puisque il n'y a aucune différence à l'oreille entre les différentes précisions à virgule flottante, pouvons nous dès lors encore réduire la taille d'un son? Pour cela nous allons essayer de passer en entiers. Essayez la commande

- `SonInt32=int32(SonDouble)` et écoutez `SonInt32` **Qu'entendez vous?**

Pour comprendre ce qui se passe, visualisez sur une figure `SonDouble` et `SonInt32`

- **Que se passe t'il ici?**

Nous allons corriger en deux temps:

- **Quelle est la valeur maxi sur un int32?**
- **Quelle est la valeur max de SonDouble?**
- **Comment faire pour coder SonDouble sur un int32 avec un maximum de précision?**
- **Réécoutez SonInt32.** Entendez vous une différence avec le codage à virgule flottante?
- **Passez en format hex et verifiez bien qu'il s'agit d'un codage sur 32 bits.**

Essayons le codage sur 16 bits:

- **Créez SonInt16** de la même manière que précédemment.
- **Passez en format hex et verifiez bien qu'il s'agit d'un codage sur 16 bits. Verifiez avec la dernière valeur**
- **Ecoutez SonInt16. Entendez vous une différence?**

Sachez que le codage sur les CD audio est codé en `int16` qui a été considéré comme provoquant des pertes inaudibles par un humain

Essayons le codage sur 8 bits:

- **Créez SonInt8** de la même manière que précédemment.
- **Passez en format hex et verifiez bien qu'il s'agit d'un codage sur 8 bits.**
- **Ecoutez SonInt8. Entendez vous une différence?**

Pourquoi ceci? Lorsqu'on passe de virgule flottante à un entier, l'erreur maximale qu'on puisse avoir c'est 0.5. Cependant dans le cas de `int16`, il s'agit de 0.5 rapporté à des valeurs entre  $-2^{16}$  et  $2^{16}-1$ . L'erreur **relative** est alors de l'ordre de  $\frac{0.5}{2^{16}} = 1.5 \times 10^{-5}$ . En `int8`, l'erreur relative est de l'ordre de  $\frac{0.5}{2^8} = 0.0039$ , ce qui devient audible pour l'oreille humaine. Très largement même.

On peut se poser la question de savoir dès lors, à partir de quelle approximation votre oreille est sensible. Malheureusement, Matlab ne dispose pas de types d'entiers intermédiaires tels que `int15`. Cependant, il est possible de simuler une précision.

- **Si on voulait créer un SonInt15 quelle serait le multiplicateur C de SonDouble?**
- **Puisqu'il est toujours possible de coder un int15 sur un int16**  
`tapez(SonInt15Sim=int16(SonDouble*C);)`
- Simuler "un codage sur 8 bits" en utilisant 16 bits en créant un signal `SonInt8Sim` de la même manière que `SonInt15Sim`.
- Pour vérifier que notre théorie est bonne, **Visualisez la différence entre SonInt8Sim et SonInt8.** Attention si vous faites `figure; plot(SonInt8Sim-SonInt8);` Matlab indiquera un message d'erreur indiquant que vous ne pouvez pas soustraire des `int8` à des `int16`. C'est vrai à peu près pour tous les langages de programmation. Tapez plutôt `figure; plot(double(SonInt8Sim)-double(SonInt8));`

- A partir de maintenant, simulez graduellement le codage de `SonDouble` sur des `int15`, `int14`, ..., `int2`, `Bool` et écoutez à chaque fois la dégradation du son. A partir de quelle approximation entendez vous une différence avec le son d'origine?

Le son que vous entendez dure 2,6 secondes.

- Quelle est la place nécessitée pour stocker cette information?
- De quelle place a t'on besoin pour un morceau de 4 minutes?
- Comment est ce possible qu'un mp3 de 4 minutes pèse seulement 3 ou 4 Mo en bonne qualité?

Ceci est dû à la **compression** qui est du ressort de la **théorie de l'information**. La théorie de l'information permet d'analyser la structure d'une information pour changer le codage de façon à le rendre moins volumineux. On distingue

- la compression sans perte : toute l'information est contenue mais de manière moins volumineuse (zip rar)
- la compression avec perte: pour gagner encore de la place, on décide de perdre de la qualité d'information mais à des endroits qui ne dérangent pas trop (mp3, mpeg, mp4, h264...). C'est souvent lié au traitement de signal.

### 3 Codage des images

Pour effacer toutes vos variables créées, tapez `clear all;`. Tapez ensuite `load 'Images';` Vous voyez alors apparaître deux images qui sont en fait des tableaux d'entiers.

- Quel est leur type?
- Quelles sont leurs valeurs min et max?
- Visualisez les en utilisant `imshow(M, [])` et `imshow(A, [])`

`imshow` est faite pour toujours afficher la valeur mini comme un pixel noir et la valeur maxi comme un pixel blanc. Ici le 0 est noir et 255 est blanc.

- Affichez `-M`, l'image négative de `M`. Que voyez vous? Pourquoi?
- Comment faire un "négatif" de l'image `M`?
- Créez une image `AM = A+M`
- Visualisez-la. Est-ce ce que vous vous attendiez à voir?
- Pourquoi est elle saturée?
- Comment pourriez vous faire pour empêcher cette saturation? Que voyez vous?
- Comment pourriez vous maintenant coder cette somme d'images sur 8 bits?

Si vous avez fait la somme correctement et que vous l'avez bien affichée, vous devriez pouvoir constater un phénomène intéressant. De près, vous voyez certainement principalement le visage d'Albert. Cependant, lorsque vous vous éloignez de l'écran vous ne verrez certainement plus que l'image de Marylin. Pourquoi?

Vous avez pu constater que ces images ne sont pas des photos originales, elles ont été légèrement modifiées:

- Marylin est floue.

- Albert est en contour.

Ces transformations sont interprétables par la discipline du **traitement de signal**. En terme de traitement de signal, on parle de fréquences. Sur une image, une basse fréquence correspond à quelque chose qui ne varie pas beaucoup ( grand surface de même intensité par exemple ) et une haute fréquence correspond surtout en image à des contours. Une image est toujours une combinaison de hautes fréquences ET de basses fréquences.

Ici Marylin n'est qu'en basse fréquence, et Albert qu'en haute fréquence qu'on obtient grâce à des filtres. Lorsque les images se superposent alors on a bien un visage avec des hautes et des basses fréquences. L'illusion d'optique vient elle du fait que de près, vos yeux vont se focaliser sur les hautes fréquences de l'images pour analyser les détails et donc, vous verrez le visage d'Albert principalement (puisque'il n'est que haute fréquence). En revanche quand vous vous éloignez, le cerveau choisi d'ignorer les basses fréquences (si vous vous concentrez, vous les voyez, mais le cerveau ne s'en sert pas pour interpréter l'information).

- **Effectuez la ligne `figure;imshow((M/2*2),[])`; Voyez vous une différence?**
- On s'attend à ce qu'il n'y en ai pas car diviser par deux puis multiplier par deux ne change rien. **Essayez la même chose avec 8 à la place de 2. Que se passe t'il?** Si vous avez du mal à comprnedre ce qui se passe essayez avec 256
- **Effectuez la ligne `figure;imshow((M*2/2),[])`; Que se passe t'il?**

Il faut toujours faire attention. Nous ne manipulons pas des nombres mais des nombres codés, et pour un uint8, les opérations  $\times 1$ ,  $/2 \times 2$  et  $\times 2/2$  ne sont absolument pas équivalentes. Si on veut faire ces opérations avec le minimum de pertes, on fera alors une conversion vers les **double** `figure;imshow(uint8((double(M)*2/2)),[])`;

Cependant cette conversion n'est pas toujours possible et dans tous les cas même les virgules flottantes souffrent d'approximation. Ce que nous allons voir dans la partie suivante.

## 4 Opérations en virgule flottante

Nous allons montrer ici pourquoi il faut faire attention à toujours garder en tête qu'un nombre en informatique est un codage de nombre.

- **Créez une variable a qui vaut 0.1 en virgule flottante à 32 bits:**
- **Verifiez par le calcul que c'est le bon codage en IEE 754**
- **Multipliez a par 2, que se passe t'il**
- **passiez en format short**
- **Faites l'opération `a*7`. Quelle est la valeur?**
- **Faites l'opération `a+a+a+a+a+a+a`. Quelle est la valeur?**
- **Faites l'opération `a*10`. Quelle est la valeur?**
- **Faites l'opération `a+a+a+a+a+a+a+a+a+a`. Remarquez vous une différence d'affichage?**
- **Comment pourrions nous faire pour voir la différence entre les deux nombres?**
- **Quelle est l'exacte différence entre les deux?**
- Ca peut paraître minime, mais que se passe t'il si l'on refait cette opération, disons, 60000 fois? Pour que vous ne tapiez pas 60000 fois `a+a+a+a..+a` (certains ont essayé, ça a duré assez longtemps) voici quelques lignes de programmation qui feront la même chose:  
`b=single(0);`

```

for i=1:60000
b=b+a;
end

```

- Maintenant affichez b. **Combien vaut-il?**
- Maintenant affichez  $a \times 60000$ . **Combien vaut-il?**
- Refaites la même chose que ci-dessus mais 360 000. Comparez b et  $a \times 360\,000$ . **Que constatez vous? Pourquoi selon vous?**
- Pour mieux comprendre ce qui se passe effectuez l'opération  $10\,000\,000 + 3600$  mais de la manière suivante: `b=single(10 000 000);`  

```

for i=1:360 000
b=b+a;
end

```
- **Affichez b, que vaut il? Pourquoi?**

Afin de comprendre les conséquences de ceci, lisez le texte suivant:

*"En février 1991, pendant la Guerre du Golfe, une batterie américaine de missiles Patriot, à Dhahan (Arabie Saoudite), a échoué dans l'interception d'un missile Scud irakien. Le Scud a frappé un baraquement de l'armée américaine et a tué 28 soldats. La commission d'enquête a conclu à un calcul incorrect du temps de parcours, dû à un problème d'arrondi. Le temps était compté par l'horloge interne du système en 1/10 de seconde. Malheureusement, 1/10 n'a pas d'écriture finie dans le système binaire. L'ordinateur de bord arrondissait 1/10 à 24 chiffres, d'où une petite erreur dans le décompte du temps pour chaque 1/10 de seconde. Au moment de l'attaque, la batterie de missile Patriot était allumée depuis environ 100 heures, ce qui avait entraîné une accumulation des erreurs d'arrondi de 0,34 s. Pendant ce temps, un missile Scud parcourt environ 500 m, ce qui explique que le Patriot soit passé à côté de sa cible.*

Je n'aime pas cet exemple car cela traite d'un sujet relativement critique. Cependant, il montre le genre de conséquences que peut avoir l'oubli de ces petites considérations. Le même genre de choses arrive régulièrement dans l'aérospatiale. Ici on voit par exemple que la multiplication des single n'est pas une simple suite d'addition. **L'analyse numérique** est la discipline qui traite de ces sujets.

## 5 Si vous avez le temps: Un peu de Pop-Art

On peut faire des images en couleur. Une image en couleur est simplement composée de 3 images : 1 rouge, 1 verte et une bleue (RGB). Pour faire ceci, il suffit par exemple de faire :

```

C(:,:,1)=A;
C(:,:,2)=0;
C(:,:,3)=0;

```

et tapez `imshow(C,[])`.

C sera une image couleur avec en couche 1 (rouge) Albert, en bleu, rien et en vert, rien. C'est donc un Albert rouge.



$C(:, :, 1)=0;$	$C(:, :, 1)=0;$	$C(:, :, 1)=A;$
$C(:, :, 2)=A;$	$C(:, :, 2)=0;$	$C(:, :, 2)=A;$
$C(:, :, 3)=0;$	$C(:, :, 3)=A;$	$C(:, :, 3)=A;$
est un Albert vert et	est un Albert bleu.	est un Albert blanc.

Remarquez d'ailleurs que l'albert blanc n'est pas "blanc" il est rouge +vert+bleu. Si vous aviez un oeil détecteur de magenta, vous verriez qu'il n'est pas blanc.

Voici quelque chose d'intéressant dans la perception des couleurs:

$C(:, :, 1)=M;$	$C(:, :, 1)=A;$
$C(:, :, 2)=A;$	$C(:, :, 2)=M;$
$C(:, :, 3)=0;$	$C(:, :, 3)=0;$
Où Marylin est en rouge et Albert en vert	Où Albert est en rouge et Marylin en vert

Voyez vous comme vous êtes plus sensible au vert? C'est normal ça vient de votre oeil. D'ailleurs cette propriété est souvent utilisée pour la compression d'image. Vous savez que  $M/8*8$  dégrade la qualité d'une image (imitation d'un codage 5 bits si M est un uint8).

Dégradons le Rouge seulement, puis le bleu seulement, puis le rouge ET le bleu, puis enfin le vert seulement.

$C(:, :, 1)=M/8*8;$	$C(:, :, 1)=M;$	$C(:, :, 1)=M/8*8;$	$C(:, :, 1)=M;$
$C(:, :, 2)=M;$	$C(:, :, 2)=M;$	$C(:, :, 2)=M;$	$C(:, :, 2)=M/8*8;$
$C(:, :, 3)=M;$	$C(:, :, 3)=M/8*8;$	$C(:, :, 3)=M/8*8;$	$C(:, :, 3)=M;$

**Affichez à chaque fois C. Au vu de ces résultats, proposez une compression avec perte de l'image.**

**si vous en êtes la vous avez tout ce qu'il faut pour faire du pop-art. Essayez.** par exemple:  $C(:, :, 1)=M/32*32;$

$C(:, :, 2)=255-M/32*32;$

$C(:, :, 3)=M/128*128;$