**NOM Prénom** Groupe

# SIM-SysIn : Test Final, partie machine

Seuls documents autorisés : ceux fournis avec la clef USB du test

**Durée 1h20**

**Lire attentivement le sujet...**

Le travail sur machine est à faire en utilisant la **clef USB** remise par l'enseignant, clef que vous devrez rendre en fin de session. Vous devez réaliser, dans l'ordre, les actions suivantes :

## 1 Renommage

Renommer le répertoire `TestFinalSysIn` de la clef USB en utilisant votre `Nom.prenom` comme nom de répertoire (pas d'accent, pas d'espace !).

Le répertoire renommé `Nom.prenom` contient une application graphique de traitement du signal, constituée de plusieurs fichiers *Python* :

- les fichiers `SPWin.py`, et `FrameImage.py` , basé sur le module PyQt pour la définition de l'interface graphique,
- les fichiers `DiscreteTime.py`, `DigitalSignal.py`, et `SampledSignal.py` pour la définition des classes permettant de représenter et traiter les signaux,
- le fichier et `DSPlotUtils.py` contenant des fonctions utiles au tracé des signaux.

L'outil à utiliser pour éditer et exécuter les fichiers *Python* est IDLE (vu en TD).

## 2 Classe `DiscreteTime`

Ouvrir le fichier `DiscreteTime.py` avec l'éditeur IDLE. Lire attentivement le code et les commentaires.

Ajouter à la fin du fichier des instructions *Python*, exécutables sous IDLE avec la touche F5, conformément aux indications suivantes (*conseil : essayer chaque ligne avec F5 avant de passer à la suivante...*) :

1. Créer un objet `t1` de type `DiscreteTime`, comprenant `5` valeurs du temps commençant à `1.0` seconde, avec un pas de temps de `0.1` seconde.
2. Faire afficher le message "`nbValues:`" suivi du nombre de valeurs du temps de l'objet `t1`.
3. Faire afficher le message "`start:`" suivi de l'instant de début de l'objet `t1`.
4. Faire afficher le message "`step:`" suivi du pas de temps de l'objet `t1`.
5. Faire afficher par `t1` la liste de ses valeurs temporelles.
6. Retarder `t1` de `5.0` sec, et refaire l'affichage du point 3.
7. Modifier la période de `t1` pour la fixer à `1` sec, et refaire l'affichage du point 5.
8. Faire afficher le message "`last:`" suivi de la dernière valeur temporelle l'objet `t1`.
9. **Bonus** : à l'aide d'une boucle `for i in range(...)`, faire afficher tout à tour les valeurs temporelles de l'objet `t1` en utilisant la méthode `t()`.

# 3 Classe `DigitalSignal`

Ouvrir le fichier `DigitalSignal.py` avec l'éditeur IDLE. Lire attentivement le code et les commentaires.

## 3.1 Prise en main

Ajouter à la fin du fichier des instructions *Python* (exécutables sous IDLE avec la touche F5), conformément aux indications ci-après (*conseil : essayer chaque ligne avec F5 avant de passer à la suivante...*) :

1. Créer un objet `ds1` de type `DigitalSignal`, commençant à `2.0` seconde, avec un pas de temps de `0.2` seconde, constitué des 10 valeurs `[0,1,2,3,4,4,3,2,1,0]`.
2. Faire afficher le message "`values:`" suivi de la liste de ses valeurs temporelles de `ds1`.
3. Vérification visuelle : demander au signal `ds1` de se tracer à l'écran, grâce à sa méthode `signal`Plot (*conseil : pour faire simple, utiliser signalPlot() sans argument*).
4. Faire afficher le message "`FTvalues:`" suivi de la liste des valeurs de la transformée de Fourier du signal `ds1` ; vérifier visuellement que les valeurs retournées sont bien complexes (remarquer au passage la notation de la partie imaginaire : $x + iy$ se note `x +yj` sous *Python*).

## 3.2 Programmation dans la classe

Compléter la méthode `delay()` conformément aux indications suivantes :

1. La méthode `delay()` de la classe `DigitalSignal` doit d'abord exécuter la méthode `delay()` de la classe de base `DiscreteTime` : c'est le but de l'instruction `DiscreteTime.delay(self, d)`, placée dans le fichier, qu'il faut dé-commenter. Vérifier (relire...) ce que fait `delay()` dans la classe de base `DiscreteTime`...
2. Il faut ensuite modifier les valeurs de la transformée de Fourier selon l'algorithme ($d$ représente le retard) :

```
    c  <-  -2*i*π*Δf*d
FTval  <-  transformée de Fourier
pour k <- 0 jusqu'à taille de FTval :
   FTval[k] <- FTval[k]*exp(c*k
```

indications :
- $i$ se note `1j` en *Python*, et $\pi$ peut s'écrire `pi` (*cf* les instructions `import` en début de fichier)
- il faut utiliser la fonction exponentielle complexe, accessible sous le nom `cexp` (*cf* les instructions `import`)
- pour la dernière ligne, on peut utiliser l'opérateur `'*='` : `x *= b` signifie `x = x*b`

3. Écrire le reste du corps de la méthode `delay()` selon les indications ci-dessus. Attention, pour obtenir la transformée de Fourier, il faut utiliser `self.__vZ` et non pas `self.FTvalues()` qui retourne une copie !

**Tests** : ajouter des instructions à la fin du fichier pour vérifier le bon fonctionnement de la méthode `delay()` :

1. Appliquer au signal `ds1` un retard de `5.0` secondes ;
2. Vérification visuelle : demander au signal `ds1` de se tracer à l'écran ;
3. **Bonus** : si vous avez 2 appels à `signalPlot` dans vos lignes de test (un avant l'appel à `delay()` et un après), ajoutez l'argument `False` au premier appel (-> ne pas fermer la figure en cours de tracé), et ajouter les arguments `color='magenta'` et `resetXlim = False` au deuxième appel, puis re-testez...

# 4 Classe SampledSignal

Ouvrir le fichier `SampledSignal.py` avec l'éditeur IDLE. Lire attentivement le code et les commentaires.

Ajouter à la fin du fichier des instructions *Python*, exécutables sous IDLE avec la touche F5, conformément aux indications ci-après (*conseil : essayer chaque ligne avec F5 avant de passer à la suivante...*) :

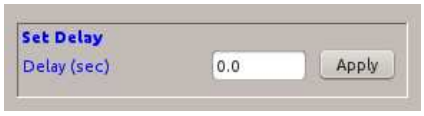
1. Créer un objet `ss1` de type `SampledSignal`, commençant à `0.0` seconde, avec un pas de temps de `0.01` seconde, constitué des 20 valeurs de l'échantillonnage du signal analogique $\sin(2\,\pi\,5\,t)$.
2. Faire afficher le message `"analog signal:"` suivi de l'expression du signal analogique de `ss1`.
3. Vérification visuelle : demander au signal `ss1` de se tracer à l'écran, grâce à sa méthode `signalPlot` (*conseil : utiliser ici signalPlot() sans argument*).
4. Retarder le signal `ss1` de `0.30` seconde, et refaire le tracé temporel.
5. **Bonus** : comme précédemment, si vous avez bien 2 appels à `signalPlot` dans vos lignes de test (un avant l'appel à `delay()` et un après), ajoutez l'argument `False` au premier appel (-> ne pas fermer la figure en cours de tracé), et ajouter les arguments `color='magenta'` et `resetXlim = False` au deuxième appel, puis re-testez...
6. Faire afficher le spectre du signal `ss1`.

# 5 Utilisation de l'interface graphique

## 5.1 Prise en main

L'interface graphique permet d'appliquer des traitements et de visualiser deux types de signaux :
– des signaux numériques, modélisés par la classe `DigitalSignal`,
– des signaux échantillonnés, modélisés par la classe `SampledSignal`.
À ce niveau des développements, vous pouvez tester la méthode `delay()` sur des signaux numériques, ou des signaux échantillonnés.

Charger le programme `SPWin.py` dans l'éditeur IDLE, et l'exécuter : l'interface permet de charger des fichiers par le menu `File->Open Digital Signal`.




Choisir un des signaux disponibles et vérifier l'efficacité du traitement `Set Delay`.

## 5.2 Création d'un fichier représentant "signal numérique"

Les fichiers représentants les signaux numériques sont des fichiers ASCII, contenant les données nécessaires à la création d'un objet de type `DigitalSignal`, utilisant l'extension `'.dsf'`.

Par exemple le contenu du fichier DS1.dsf est illustré ci-dessous :
Les conventions sont usuelles :
– les lignes qui comment par un '#' sont des lignes de commentaires,
– les données sont placées en début de ligne, séparées d'un éventuel commentaire par un ou plusieurs espaces ou tabulations.

```
# data for a digital signal
 15      number of discrete values
 1       time step
-2       start time
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0
```

Travail à faire :

1. En utilisant le menu `'File > New Windows'` de l'éditeur IDLE, créer un fichier `'.dsf'` portant vos initiales, commençant à -8 secondes, avec un pas temporel de 0.1 seconde et comportant les 17 valeurs discrètes : 0 -1 -2 -3 -4 -3 -2 -1 0 1 2 3 4 3 2 1 0.
2. Charger ce fichier dans l'interface graphique,
3. appliquer 8 fois de suite un retard de 1 seconde,
4. enregistrer le tracé temporel (bouton `Save` en bas à droite du tracé) dans un fichier nommé `DSP_vos_initiales.png`

compléter la lecture des fichiers représentants les signaux échantillonnés

essayer retard

# 6 Bonus : codage de la méthode `zeroPadding` de la classe `DigitalSignal`

## 6.1 Programmation dans la classe

Compléter la méthode `zeroPadding()` conformément aux indications suivantes :

1. La méthode `zeroPadding()` doit renvoyer un objet résultat du traitement zéro-padding appliqué à l'objet courant.
2. Si le nombre de zéros à rajouter est nul, on renvoie simplement la copie de l'objet courant grâce à l'instruction `'return copy(self)'` (`copy` est connu grâce aux instructions `import` en début de fichier)
3. Sinon, il faut programmer l'algorithme suivant ( `nbZeros` est le nombre de zéros à ajouter) :

```
00  Début:
01    N    <- nbre de valeurs temporelles discrètes
02    to   <- instant de départ du signal
03    Te   <- le pas temporel du signal * N /(N+nbZeros)
04    Fe   <- 1/Te
05    vY  <- la copie de la transformée de Fourier du signal
06    m   <- le nbre de valeurs de vY
07    c   <- 2*i*π*Δf*to
08    pour k = 0 jusqu'à m-1 :
09          vY[k] <- vY[k]*Fe*exp(c*k)
10    si N égale 2*m-2 :
11          diviser la dernière valeur de vY par 2
12          ajouter 'nbZeros-1' zéros à la fin de vY
13    sinon :
14          ajouter 'nbZeros' zéros à la fin de vY
15    pour k = m-1 jusqu'à 1, par pas de -1 :
16          ajouter à la fin de vY le conjugué de vY[k]
17    valeurs <- partie réelle ( transformée de Fourier inverse (vY))

18  Fin : retourner un objet de type DigitalSignal, construit avec Te,to et valeurs.
```

4. Indications pour écrire le code *Python* :
   - l'utilisation de la méthode `FTvalues()` pour définir `vY` fournit une copie de la transformée de Fourier de type `list`,
   - $i$ se note `1j` en *Python*, et $\pi$ peut s'écrire `pi` (*cf* les instructions `import` en début de fichier)
   - il faut utiliser la fonction exponentielle complexe, accessible sous le nom `cexp` (*cf* les instructions `import`)
   - pour les calculs, on peut utiliser l'opérateur `'*='` : `x *= b` signifie `x = x*b`
   - attention aux bornes de range() pour la ligne 15 : faire des essais dans un shell Pyhon pour bien voir ce que donne `range(start, stop, pas)`
   - pour extraire la liste des parties réelles de la liste vY, on peut :
     - définir une fonction locale `re(z)` (dans le corps de la méthode `zeroPadding()`) qui retourne `z.real`
     - utiliser la fonction `map`() pour appliquer `re`() à `list`(transformée de Fourier inverse(`vY`))
   - la transformée de Fourier inverse est accessible sous le nom `ifft()` (cf les imports en début de fichier)
5. Écrire le corps de la méthode `zeroPadding()` selon les indications ci-dessus.

**Tests** : ajouter des instructions à la fin du fichier pour vérifier le bon fonctionnement de la méthode `zeroPadding()` :

1. Construire un objet `ds2`, résultat de l'appel de `zeroPadding()` sur ds1, avec 100 zéros de padding.
2. Faire tracer le signal `ds2`, en utilisant `signalPlot(False, size=3, color="magenta").`
3. Faire tracer le signal `ds1`, en utilisant `signalPlot(resetXlim=False, resetYlim=False).`

## 6.2 Utilisation de l'interface graphique

Lancer l'interface graphique, et utiliser le traitement zéro-padding sur le signal de votre choix.