## Examen « Matlab »

## Durée : 3H (sur PC)

## 

## Seuls les documents fournis sont autorisés. La calculatrice collège est autorisée.

**DEPÔT DU LIVRABLE SUR MOODLE**

DEPOSER votre unique livrable au format .zip à votre **Nom\_prenom.zip** ou .rar avec 3 fichiers exo1.m exo2.m exo3.m + fichier data suivant le lien suivant :

<https://moodle-ingenieurs.cesi.fr/mod/assign/view.php?id=94900>

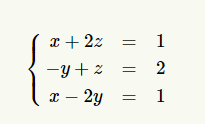
|  |  |
| --- | --- |
| **Compétences à valider** | Pondération |
| C1 - Savoir résoudre un système d’équations sous forme matricielle avec Matlab | x 1 |
| C2 – maitrise de la boucle for | x 1 |
| C3 – Retrouver des paramètres N, Fe | x 1 |
| C4 - Savoir construire une fonction avec et sans paramètres d’entrée et sortie (varargin / varargout) et déployer différents scénarios « démo » et « projet » | x 1 |
| C5 – Maitrise de l’affichage (subplot) | x 1 |
| C6 – Calcul à partir de données préchargées | x 1 |
| C7 - Savoir s’approprier un algorithme et compléter les lignes de codes manquantes | x 2 |

Exercice 1 : Systèmes et matrices

*Les 2 parties de cet exercice sont totalement indépendantes*

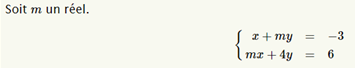
**Partie A** :

Dans cet exercice, on se propose d’utiliser l’écriture matriciel sous Matlab pour résoudre le système d’équations suivant :



**Partie B** :

Dans un second temps, on se propose de déterminer les solutions du système suivant les valeurs d’un paramètre variable m :



Autrement dit, les solutions (x,y) qui vérifient l’équation ci-dessus sont les points situés aux intersections des droites :

x+my = -3 soit :

y1 = (-3-x)/m

et mx+4y = 6 soit :

y2 = (6-mx)/4

On admettra dans le programme Matlab que les droites y1 et y2 se coupe lorsque :

| y1(xo) – y2(xo) | < 0.01

Où Xo est l’abscisse où les droites se croisent.

Exercice 2 :

L’objet de cet exercice est **de compléter la fonction** **filt\_num** (qui réalise le filtrage d’un signal bruité) à partir de 4 paramètres :

1. **SNR** : le rapport signal à bruit
2. **ordre** : ordre du filtre moyenneur
3. **A** : Amplitude du signal utile (signal sans bruit)
4. et **display** :

* S’il est à 0 :
  + cela signifie qu’on ne veut aucun affichage
  + on doit pouvoir récupérer en sortie de la fonction filt\_num les différents signaux
* S’il est à 1 :
  + Affichage des signaux avant et après filtrage et magnitude (= module) du filtre
  + La fonction filt\_num exécutée sans paramètres de sortie

**L’utilisateur doit pouvoir choisir s’il veut un mode démo** :

* Auquel cas, aucun paramètre en entrée (ni en sortie) de la fonction filt\_num (les paramètres sont définis par défaut dans la fonction filt\_num)

**Ou s’il veut un mode projet** :

* Auquel cas, les paramètres SNR, ordre, A et display sont demandés à l’utilisateur et pris en entrée de la fonction filt\_num

Exercice 3 : Codage de Huffmann

Le **codage de Huffman** est un algorithme de **compression de données sans perte**.

Le principe consiste à associer un code binaire à chaque caractère.

Supposons un fichier contenant 1 phrase : l’idée est de coder chaque caractère de la phrase de la manière suivante :

* plus le caractère est présent dans la phrase, plus son code binaire sera court (pour diminuer la taille du fichier)
* plus le caractère sera rare, plus son code binaire sera long

(peu d’impact sur la taille du fichier car caractère peu fréquent)

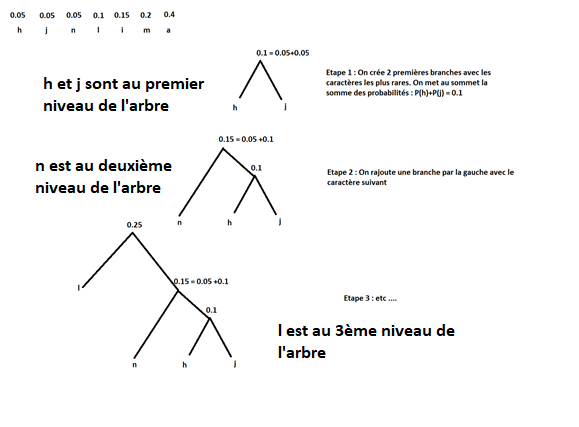
**Exemple que nous allons utiliser dans notre script matlab** :

* Phrase à coder : « ah mamia j’ai mal a la main »
  + Nombre de caractères : 20 (on va omettre les espaces et l’apostrophe)

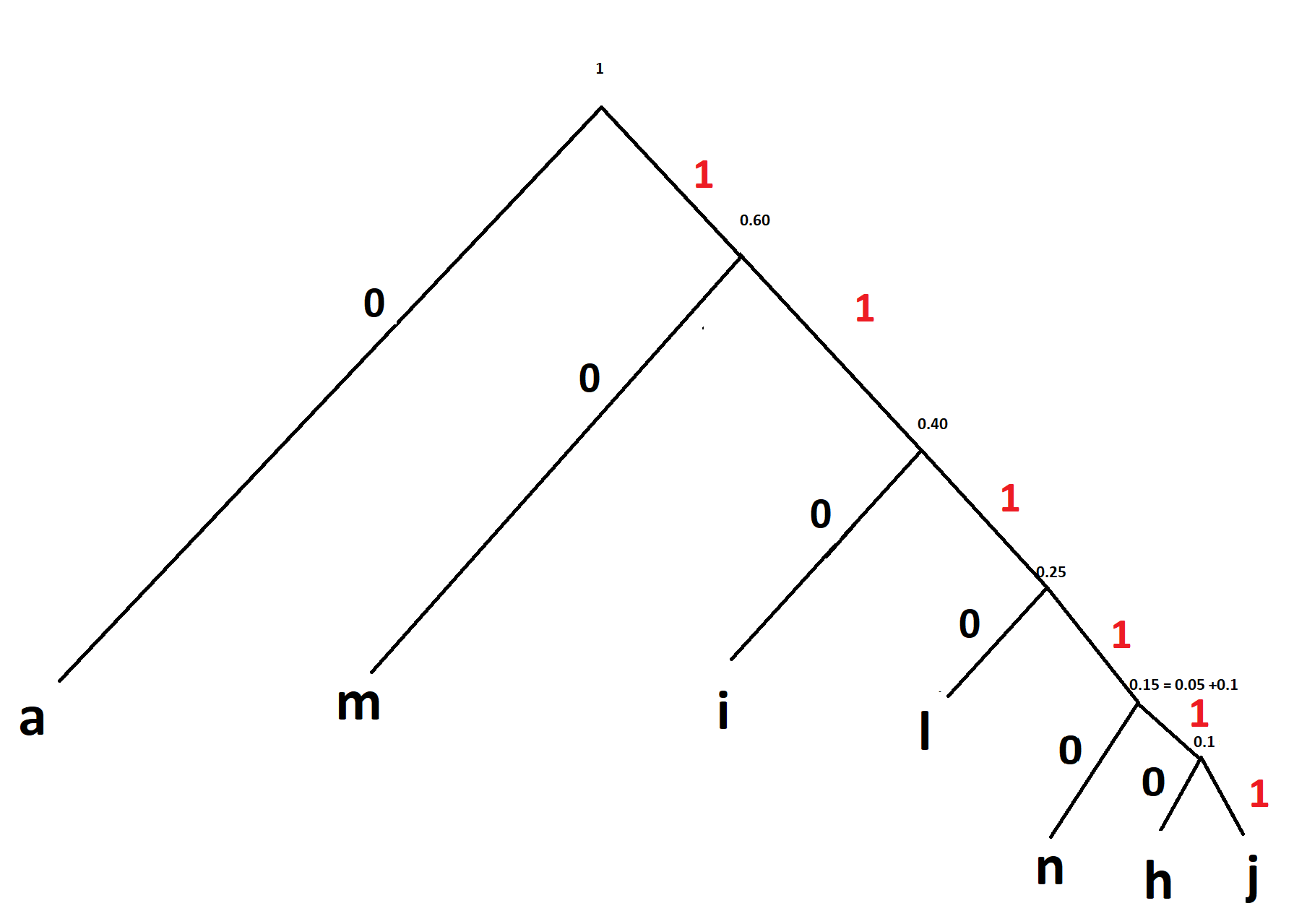
On compte le nombre d’occurrences de chaque caractère :

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Caractère | **a** | **h** | **i** | | **j** | **l** | **m** | **n** | |
| Nombre d’apparitions | **8** | **1** | **3** | | **1** | **2** | **4** | **1** | |
| fréquences | **0,4 = 8/20** | **0.05**  **=**  **1/20** | **0.15 =**  **3/20** | **0.05 =**  **1/20** | | **0.1 =**  **2/20** | **0.2**  **=**  **4/20** | | **0.05**  **=**  **1/20** |
| Total de caractères | **20** | | | | | | | | |

On réalise ensuite un arbre



**Au final on arrive à l’arbre suivant par construction en suivant les étapes précédentes.**

* Chaque branche de gauche sera codée avec un ‘0’
* Chaque branche de droite sera codée avec un ‘1’
* En parcourant le chemin depuis le sommet jusqu’au caractère, on obtient son code binaire :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Caractère | **a** | **m** | **i** | **l** | **n** | **j** | **h** |
| Code | 0 | 10 | 110 | 1110 | 11110 | 111110 | 111111 |
| Nombre de bits | 1 | 2 | 3 | 4 | 5 | 6 | 6 |
| Niveau dans l’arbre | 6 | 5 | 4 | 3 | 2 | 1 | 1 |
| Indice caractère | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

**Bilan de compression des données**

* Sans Compression (SC) et avec un codage sur 3 bits
  + car 3 bits suffisent pour coder 7 caractères
  + chaque caractère est codé sur 3 bits, la phrase a une taille de :

**SC** = 3 bits x 20 caractères = **60 bits**

* Avec Compression (AC) la phrase a une taille de :

AC = nombre de ‘a’ x taille du code de ‘a’ + nombre de ‘m’ x taille du code de ‘m’+ + nombre de ‘i’ x taille du code de ‘i’ + ….

**AC** = 8 x 1 + 4 x2 + 3 x 3 + 2x4+ 1x5+ 1x6 + 1x6 = **50 bits**

* Par conséquent le taux de compression est de :

Taux de compression = [1 - (50/60)]\*100 = **17%**

**Remarque :**

* + Ce taux serait beaucoup important avec tous les caractères de l’alphabet.