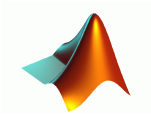

EXAMEN DE GÉNIE INFORMATIQUE

Durée 2h00.
Tout document autorisé.
Communication avec l'extérieur (internet, téléphone ou autre) interdit.
Collaboration avec d'autres étudiants interdite.

1 • INTRODUCTION.

1.1 • But de l'examen.

Le but ultime de cet examen est de réaliser des fonctions permettant de représenter la dépendance de deux vecteurs X et Y ayant la même taille sous plusieurs formes graphique. On vous demande de réaliser plusieurs fonctions permettant de réaliser cette tâche et de les lier entre elles par un programme principal.
Le programme à réaliser est très simple. Si vous êtes astucieux, il ne prendra pas beaucoup de lignes de code. Pour vous noter, nous lancerons simplement votre programme. Si tout fonctionne correctement et que votre programme est bien commenté, vous aurez la note maximale. Dans le cas contraire, les points vous seront attribués en fonction de la qualité de votre travail de programmateur. Pensez à commenter votre programme !!!
Si vous voulez réussir cette épreuve, il vous faut **très bien vous organiser**, ne mettre aucune valeur en dur et écrire tout bien proprement, sans quoi vous aurez toutes les peines du monde à enchaîner les différentes parties de cette épreuve.

1.2 • Restitution de votre travail.

Votre travail doit être restitué en fin d'épreuve sous la forme d'un (ou plusieurs) fichier(s). Vous devez envoyer vos fichiers avec vos nom et prénom à mon adresse (olivier.strauss@lirmm.fr) au maximum 5mn après la fin de l'épreuve! Toute minute de retard est retirée sur la note (1mn = 1/2 pt en moins) - la date d'envoi faisant foi.
Vous devez déclarer clairement vos variables en début de fichier. **Attention** : votre programme doit bien sur fonctionner quelque soit le nombre de valeurs des vecteurs utilisés.

2 • ANALYSE DE LA DÉPENDANCE DE DEUX VECTEURS DE VALEURS.

2.1 • Principe.

L'analyse de la dépendance de deux vecteurs $X = [x_1, \ldots, x_n]$ et $Y = [y_1, \ldots, y_n]$ peut se faire graphiquement par la représentation 1- du nuage de points les représentant 2- de l'ellipsoïde d'incertitude à 99% (ellipsoïde incluant a priori 99% des couples de points) 3- de la droite de régression (la droite passant au plus près des points de coordonnée (x_i, y_i) i allant de 1 à n).
Nous vous proposons de réaliser un programme réalisant ces affichages.

Vous pourrez tester votre programme en utilisant les deux vecteurs X et Y générés par le programme suivant :

```
>> NombreEchantillon = 238 ;
>> T = 1:NombreEchantillon ;
>> X = sinc(5*T/max(T)) * NombreEchantillon ;
>> Y = (T.*T -56*T)/NombreEchantillon ;
>> Y = Y + 0.3*NombreEchantillon*randn(size(T)) ;
>> Y = 0.02*X.*X + Y ;
```

2.2 • Génération des données.

Créez un programme s'appelant **Generation.m** utilisant le code ci-dessus (ou un code que vous imaginerez) pour générer deux vecteurs X et Y qui vous serviront de test.

2.3 • Fonction de calcul des paramètres de la droite de régression.

Vous devez réaliser une fonction que vous appellerez **Regression**.
Cette fonction doit avoir comme prototype :

```
[a, b] = Regression( Vecteur1, Vecteur2 ) ;
```

Elle doit calculer les coefficients a et b de la droite moindres-carrés d'équation $(ax + b = y)$ passant au plus près des couples de points (x_i, y_i). Vous avez déjà réalisé partiellement ce programme en TP. Vous devez le mettre sous forme de fonction.
Il vous est conseillé de bien commenter cette fonction et de la tester.

2.4 • Fonction de calcul de l'ellipsoïde d'incertitude.

Vous devez réaliser une fonction que vous appellerez **Ellipsoide**.
Cette fonction doit avoir comme prototype :

```
[Rotation, Translation, Rayon] = Ellipsoide( Vecteur1, Vecteur2 ) ;
```

Les variables `Rayon` et `Translation` sont des vecteurs à deux dimensions tandis que la variable `Rotation` est une matrice 2×2.
Pour les calculer, il faut créer la matrice d'inertie Σ des variables. Cette matrice Σ se définit ainsi : $\Sigma = \frac{1}{n} A A^T$ (T veut dire transposée) ou la matrice A est définie par :

$A = \begin{bmatrix} x_1 - \tilde{x}, \ldots, x_n - \tilde{x} \\ y_1 - \tilde{y}, \ldots, y_n - \tilde{y} \end{bmatrix}$, n est le nombre d'éléments de chaque vecteur, $\tilde{x}$ et $\tilde{y}$ étant respectivement la moyenne des valeurs du premier vecteur et la moyenne des valeurs du second vecteur. Il faut ensuite chercher les vecteurs propres et les valeurs propres de Σ. `Rayon` est le vecteur des deux valeurs propres tandis que `Rotation` est la matrice des deux vecteurs propres et `Translation` est le vecteur $\begin{bmatrix} \tilde{x} & \tilde{y} \end{bmatrix}$. Vous devez naturellement chercher sous matlab la fonction qui permet de calculer les valeurs propres et les vecteurs propres d'une matrice (aide : en anglais valeurs propres se dit *eigenvalues*).

2.5 • Fonction calculant des points sur une ellipse.

Vous devez réaliser une fonction que vous appellerez **PointsDuneEllipse** dont le prototype est le suivant :

```
[coo_x, coo_y] = PointsDuneEllipse( Rayon, nombre_de_points ) ;
```

`coo_x` et `coo_y` sont les coordonnées en x et y de points de l'ellipse, `Rayon` est le vecteur des deux rayons de l'ellipse (rx et ry).

Pour générer M points d'une ellipse il faut 1- définir un pas angulaire $\delta\theta = \frac{2\pi}{M}$, 2- calculer la suite des M points par : $\begin{cases} \theta_i = (i-1)\delta\theta \\ coox_i = rx\cos(\theta_i) \\ cooy_i = ry\sin(\theta_i) \end{cases}$ (i allant de 1 à M).

Pensez à tester votre fonction en affichant le résultat (vous devez obtenir une ellipse).

2.6 • Transformation des points.

Vous devez réaliser une fonction que vous appellerez **Transformation**, réalisant une transformation géométrique, dont le prototype est le suivant :

```
[Points_transformes] = Transformation( Points, Rotation, Translation ) ;
```

`Points` est un vecteur de p points 2D (vecteur $2 \times p$ ou $p \times 2$), `Rotation` une matrice 2×2 et `Translation` un vecteur de deux valeurs. `Points_transformes` est un vecteur $2 \times p$ ou $p \times 2$ des points après transformation.

Soit R la matrice de rotation, T le vecteur de translation et $P = \begin{bmatrix} x \\ y \end{bmatrix}$ un point. La transformation de P par R et T s'écrit : $P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = RP + T = R\begin{bmatrix} x \\ y \end{bmatrix} + T$.

Testez cette fonction sur des exemples simples.

2.7 • Programme principal.

Le programme principal doit afficher deux fenêtres. Chaque fenêtre doit avoir comme largeur 80% de la largeur de l'écran et comme hauteur 40% de la hauteur de l'écran. Les deux fenêtres doivent être disposées l'une en dessous de l'autre.

Dans la première fenêtre dont le titre sera "droite de regression" on doit voir le nuage de points originaux (points bleus non connectés) superposé avec la droite de régression (en rouge) dont les points sont calculés grâce à la fonction **Regression**. Inspirez vous des TP pour afficher cette droite.

Dans la seconde fenêtre dont le titre sera "ellipse d'incertitude" on doit voir le nuage de points originaux (points bleus non connectés) superposé avec l'ellipse d'incertitude. Pour calculer cette ellipse, on se servira d'abord de la fonction **Ellipsoide** pour calculer les coefficients de l'ellipse, puis de la fonction **PointsDuneEllipse** pour calculer des points

sur l'ellipse dans son propre repère (elle sera horizontale), puis **Transformation** pour placer ces points dans le repère des points originaux.

La séquence des commandes doit ressembler à :

```
>> Generation ;
>> nombre_de_points = 228 (par exemple) ;
>> [a, b] = Regression( X, Y ) ;
>> (affichage de la droite sur les points originaux)
>> [Rotation, Translation, Rayon] = Ellipsoide( X, Y ) ;
>> [x, y] = PointsDuneEllipse( Rayon, nombre_de_points ) ;
>> (Creation d'un vecteur Point a partir de x et y)
>> [Points_transformes] = Transformation( Points, Rotation, Translation ) ;
>> (affichage de l'ellipse sur les points originaux)
```