TP2 : Tracé de graphes et probabilités

Dans ce TP, nous verrons comment modéliser une expérience aléatoire avec Octave, et nous verront également comment utiliser les fonctionnalités de tracé de graphes sous Octave.

1 Réaliser une expérience aléatoire avec Matlab (30mn)

On veut simuler, à l'aide d'un programme Matlab/Octave, un lancer de dé à 6 faces, pour savoir combien il faut, en moyenne, faire de lancers pour obtenir un score de 20 ou plus. Sur le papier, ce résultat est simple à calculer : le score moyen d'un dé est de 3.5, il faut donc faire $20/3.5 = 5.7$ lancers environ pour obtenir 20. On en conclue qu'avec 6 lancers, on obtiendra 20 ou plus.

On souhaite cependant s'assurer, expérimentalement, de la validité de ce résultat en simulant, plusieurs fois, cette expérience aléatoire. Pour réaliser l'expérience aléatoire avec Octave, on utilisera la fonction **randi(n)**, permettant d'obtenir un nombre entier aléatoire entre 1 et n :

```
result = randi(6,1);
```

Nous additionnerons le résultat ainsi obtenu dans une variable *somme*, et continuerons tant que la valeur de *somme* n'aura pas égalé ou dépassé 20 :

```
somme=0;
while (somme<20)
       somme = somme+randi(6,1);
end
```

Il faut compter le nombre de lancers qu'il aura fallu faire pour que *somme* ne dépasse 20 ; on utilisera ici une troisième variable servant à compter combien de fois nous avons lancé le dé avant de dépasser 20 :

```
somme=0;
lancer = 0;
while (somme<20)
       somme = somme + randi(6,1);
       lancer = lancer + 1;
end
```

Nous avons le résultat du nombre de lancers qu'il aura fallu faire pour que le score dépasse 20. Cependant, il faut maintenant répéter cette expérience plusieurs fois afin d'avoir un échantillon statistique significatif. Ici, répéter l'expérience, à l'aide d'une boucle **for**, dix mille fois sera suffisant (le code sera un peu long à s'exécuter... si cela est trop long, modifier la valeur 10000 à une valeur plus petite) :

```
for i=1:10000
    lancer = 0;
    somme=0;
    while somme<20
        somme = somme+randi(6,1);
        lancer = lancer+1;
    end
end
```

Nous n'avons pas complètement terminé... Il faut, pour chaque expérience, sauvegarder le résultat, c'est à dire la valeur de *lancer* en sortie de la boucle **while**. C'est ce que nous allons mettre en place dans la suite.

Questions

1. On souhaite obtenir un vecteur r tel que $r(i)$ nous indique le nombre de lancer qu'il aura été nécessaire de faire à la i-ème tentative. Modifiez votre code en conséquent.
2. L'histogramme de r nous permettra de visualiser aisément, pour chaque valeur i entre 0 et 20, le nombre de fois où lancer le dé i fois aura permis de dépasser 20. Tracez l'histogramme de r (voir le cours pour la commande permettant de le faire).
3. L'histogramme a peut-être un axe des abscisses un peu étrange, rendant sa lecture difficile (car Matlab a choisi automatiquement l'axe des abscisses, et ses choix ne sont pas toujours très judicieux). Or, on sait que pour gagner le jeu, il faut lancer le dé au moins 0 fois et au plus 20 fois. Trouvez un moyen de préciser à la fonction histogramme que chaque *bin* de l'histogramme (chaque barre) doit correspondre à une valeur entière entre 0 et 20.
4. Pour présenter correctement l'histogramme, trouvez un moyen d'ajuster l'axe des abscisses afin que, sans rappeler la fonction histogramme, on n'affiche que les valeurs de l'histogramme entre 3 et 10 (avec la fonction **xlim**).
5. Enfin, servez-vous de l'histogramme pour analyser les résultats de l'expérience. Combien de fois, le plus souvent, faut-il lancer le dé pour obtenir 20 ou plus ? Est-ce en accord avec ce que nous avions calculé, théoriquement, au début du programme ?

2 Expérience aléatoire des dés sans aucune boucle (10mn)

Il est possible de totalement vectoriser le code de l'expérience aléatoire en ne mettant aucune boucle apparente : c'est ce que l'on appelle "vectoriser le code". Cette approche correspond à la philosophie de Matlab et d'Octave, dans lequel il est conseillé d'éviter les appels de boucle.

Pour ce faire, on commence à réfléchir au problème du lancer de dés : afin d'atteindre le score de 20, il faudra lancer le dé au plus 20 fois. Au lieu de générer un seul nombre aléatoire entre 1 et 6, on peut en générer 20 d'un seul coup :

```
alea = randi(6, [20, 1]);
```

Nous obtenons un vecteur de 20 lancers aléatoires de dés. Cependant, il faudra répéter l'expérience dix mille de fois ; autant générer dès le début dix mille fois les 20 lancers de dé :

```
alea = randi(6, [20, 10^4]);
```

La matrice obtenue possède dix mille colonnes. **Chaque colonne représente "une partie" pendant laquelle nous réalisons 20 lancers de dé**, et nous allons nous intéresser à quel moment le score obtenu dépasse 20. Nous allons pour cela faire la somme cumulative de chaque colonne : pour chaque case i, nous allons calculer la somme de toutes les cases situées au-dessus de i dans la matrice.

```
s = cumsum(alea, 1);
```

Nous allons chercher, pour chaque colonne, à quel moment cette somme dépasse 20 :

```
f = (s>=20);
```

Nous obtenons une matrice de booléen : sur chaque colonne, nous avons de 0 au départ, puis de 1 à partir du moment où le nombre de lancers a permis de dépasser le score de 20. Pour récupérer le numéro de ligne où le score dépassé 20 la toute première fois à chaque colonne, nous utiliserons la fonction max qui nous renverra la valeur et la position du premier max de chaque colonne :

```
[valeur_max,position_max] = max(f, [], 1);
```

Dans $valeur\_max$, nous avons la valeur maximale de chaque colonne de f, à savoir 1 (car f est une matrice de booléen). Dans $position\_max$, nous avons, pour chaque colonne, le numéro de ligne où ce 1 apparait la première fois : cela nous donne le nombre de lancers qu'il aura fallu faire pour obtenir 20 ou plus... Le vecteur $position\_max$ est précisément le vecteur r de la partie précédente. Nous pouvons continuer et terminer en faisant le même code pour afficher l'histogramme de r :

```
r = position_max;
hist(r, 3:18)
xlim([3 10])
```

Voici le code résumé, où aucune boucle n'apparait explicitement :

```
alea = randi(6, [20, 10^4]);
s = cumsum(alea, 1);
f = (s>=20);
[valeur_max,position_max] = max(f, [], 1);
r = position_max;
hist(r, 3:18)
xlim([3 10])
```

Évidemment, des boucles sont tout de même présentes dans ce code, dans les appels de **randi**, **cumsum**, **max**, etc... Cependant, elles sont codées dans ces fonctions qui ont été écrites en C, optimisées pour s'exécuter rapidement et, si possible, en parallèle sur plusieurs microprocesseurs de la machine.

> **Questions**
>
> Ce code est-il plus rapide que le code de l'exercice précédent (les deux codes sont sensés donner un résultat similaire) ? Vous est-il possible de tenter, avec ce code, un million de fois le lancer de dé ?

3 Tracé de fonction (25mn)

Dans cet exercice, nous verrons comment tracer les valeurs d'une suite afin de mettre en valeur sa convergence. Le but ici sera de présenter correctement le graphe à l'aide des fonctions vues en cours. **Il est recommandé ici d'écrire et d'exécuter les lignes de code au fur et à mesure, afin de se rendre compte de ce que chacune d'elles permet de faire sur le graphe**.

On pose la suite (u_n) :

$$u_n = \frac{n-3}{n+3}$$

On souhaite illustrer le fait que la suite (u_n) converge vers 1. Pour ce faire, on calculera les valeurs successives de la suite pour n allant de 0 à 100. Tout d'abord, on déclare deux vecteurs : un où l'on rangera les entiers de 0 à 100, et un autre où l'on rangera les valeurs de u_n :

```
n = 0:100;
u_n = (n-3) ./ (n+3);
```

On souhaite ensuite tracer les valeurs de u_n en fonction de n, à l'aide de la fonction **plot**. On va tracer ces valeurs à l'aide d'une ligne rouge :

```
plot( n, u_n, 'Color', 'r' );
```

On voit sur le graphe affiché que la suite semble effectivement converger vers 1. Pour mettre cela en valeur, on va aussi tracer, sur le graphe, la droite $y = 1$. Pour ce faire, on construit un vecteur de la même taille que u_n, rempli de 1 :

```
const = ones( size(u_n) );
```

Une fois ce vecteur construit, on va l'afficher sous forme d'une ligne pointillée bleue. Tout d'abord, on utilise la commande **hold** afin que le prochain tracé que l'on affichera se superpose à l'ancien tracé :

```
hold on;
```

Puis, on peut tracer le vecteur :

```
plot( n, const, 'Color', 'b', 'LineStyle', '--' );
```

Comme on ne voit pas très bien la ligne bleue qui se retrouve tout en haut du graphe, on modifie l'axe des ordonnées afin de faire "de la place" en haut. Pour ce faire, on utilise la fonction **ylim** qui prend en paramètre un vecteur de deux valeurs : la première est la valeur minimale que l'on souhaite avoir pour l'axe des abscisses, et la seconde est la valeur maximale :

```
ylim([-1.5 1.5]);
```

Pour mieux présenter notre graphe, on peut y ajouter des titres et étiquettes pour les axes :

```
title('La suite u(n) converge vers 1');
xlabel('n');
ylabel('u(n)');
```

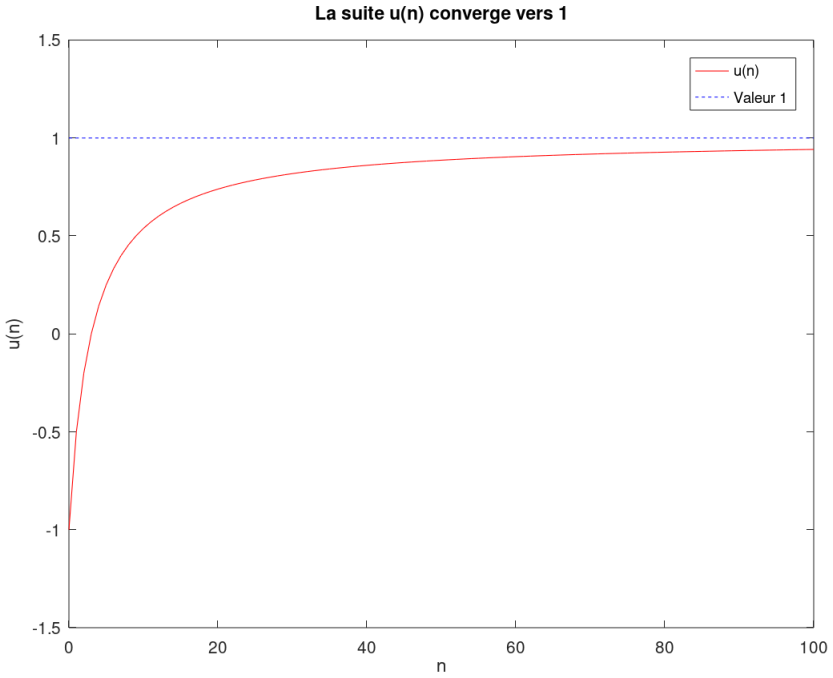
Enfin, une petite légende sera utile afin de préciser à quoi correspondent les deux tracés :

```
legend('u(n)', 'Valeur 1');
```

Puis, pour éviter qu'un prochain tracé ne vienne se superposer à notre graphe par erreur, on désactive le **hold** :

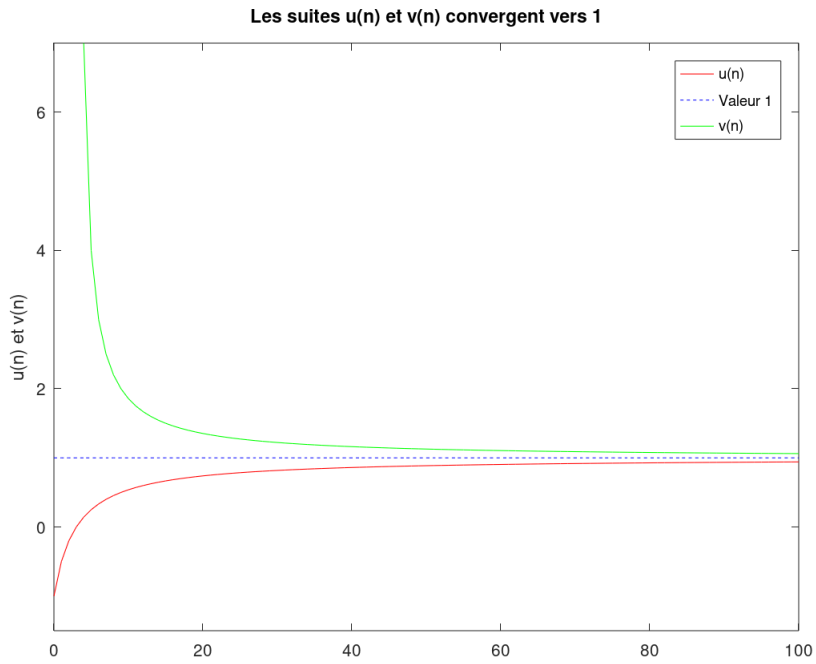
```
hold off;
```

Voici à quoi devrait ressembler votre figure une fois toutes les instructions ci-dessus effectuées :



Il est possible de sauvegarder le graphe obtenu en cliquant sur *File* en haut de la fenêtre contenant le graphe, puis *Save as*. Il est conseillé d'utiliser le format d'image PNG pour sauvegarder votre graphe en image et l'inclure ensuite dans un rapport (il faut simplement nommer votre fichier avec l'extension .png pourque Octave comprenne que vous souhaitez utiliser le format PNG).

> **Questions**
>
> Complétez votre graphe afin de rajouter la suite $v_n = \frac{n+3}{n-3}$, et illustrez qu'elle converge aussi vers 1. Attention, cette suite n'est pas définie pour $n = 3$, on commencera donc le tracé pour $n = 4$. Votre graphe devra ressembler à ceci :



4 Facultatif - Trajectoire d'un robot

Les positions (abscisse et ordonnée) d'un robot se déplaçant sur une table sont relevées à chaque seconde :

t (s)	0	1	2	3	4	5	6	7	8	9
x (cm)	7.5	3.8	6.5	9.5	5.7	8.4	2.7	6.2	5.8	9.6
y (cm)	0.8	5.0	5.2	0.9	9.0	8.8	4.3	7.8	1.4	6.1

Autrement dit, vous avez deux vecteurs :

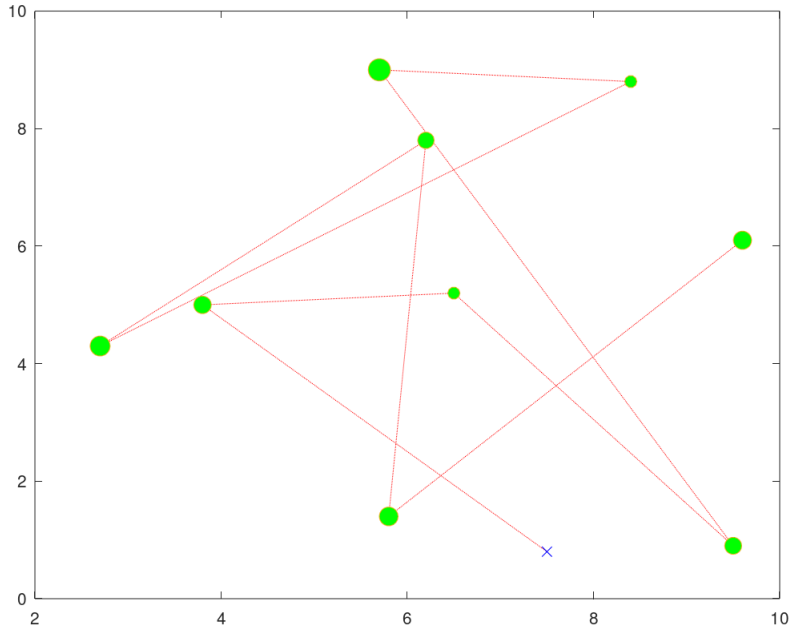
```
x = [7.5 3.8 6.5 9.5 5.7 8.4 2.7 6.2 5.8 9.6];
y = [0.8 5.0 5.2 0.9 9.0 8.8 4.3 7.8 1.4 6.1];
```

Questions

1. Représentez, sur un graphe, la trajectoire du robot par une ligne pointillé rouge. Entre deux relevés, on imaginera que le robot a suivi une ligne droite.
2. Représentez en plus le point de départ du robot par une croix bleue à l'aide de la fonction **scatter**.
3. A une position (x_i, y_i), l'incertitude sur la position du robot est proportionnelle à la vitesse moyenne du robot entre les points (x_{i-1}, y_{i-1}) et (x_i, y_i). Comme les positions du robot son relevées à chaque seconde, la vitesse du robot est donc directement proportionnelle à la distance entre les points (x_{i-1}, y_{i-1}) et (x_i, y_i).

   Représentez, pour chaque relevé de position (x_i, y_i) (autre que le point de départ), l'incertitude sur la position du robot à l'aide d'un disque vert dont le rayon variera en fonction de l'incertitude. On regardera, pour cela, la documentation de la fonction **scatter**, et spécifiquement le rôle du troisième paramètre que peut prendre la fonction.

Vous devriez avoir obtenu, à ce stade, quelque chose ressemblant à ceci :



On souhaiterait maintenant avoir une spline cubique interpolant notre ensemble de points, comme ce que l'on peut voir sur la seconde figure de la page https://fr.mathworks.com/help/curvefit/examples/constructing-spline-curves-in-2d-and-3d.html.

Dans un premier temps, sur un nouveau graphe, nous allons tracer la position x du robot en fonction du temps t :

```
figure;
t = 0:9;
plot(t,x, 'r');
```

Maintenant, nous allons calculer la spline cubique permettant d'interpoler la courbe que l'on vient de tracer :

```
ppx = spline(t,x);
```

La variable *ppx* ainsi obtenue est un objet de type spline cubique, c'est à dire une fonction polynomiale (de degré 3) par morceaux. Si nous souhaitions connaître sa valeur au point d'instant $t = 1.5$, nous ferions

```
v = ppval(ppx, 1.5);
```

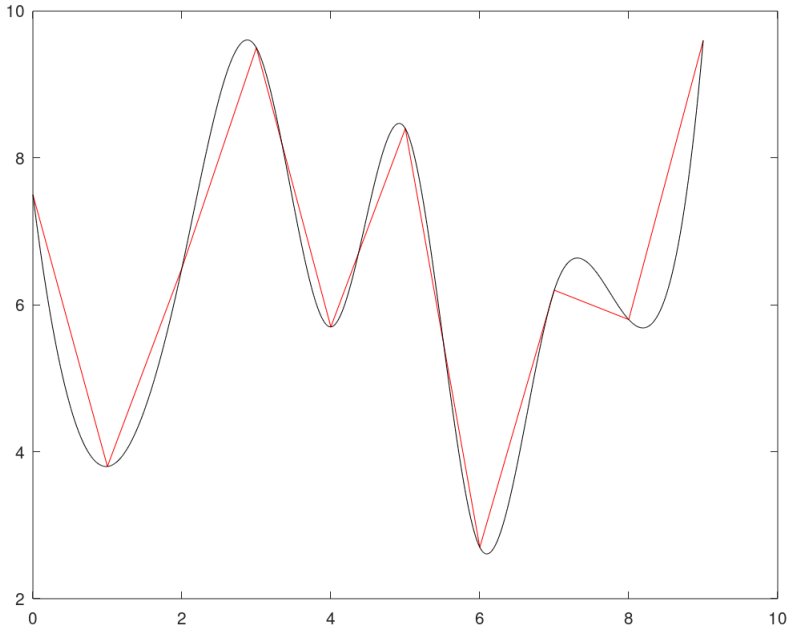
Or, ici, nous souhaitons connaître la valeur de la spline pour de nombreux instants entre 0 et 9 secondes. Ces valeurs seront les positions "estimées" du robot pour de nombreux instants entre 0 et 9 secondes. Nous allons demander (par exemple) toutes les valeurs intermédiaires pour des instants, au centième de seconde près, entre 0 et 9 secondes :

```
t_prime = 0:0.01:9
vx = ppval(ppx, tprime);
```

On peut tracer ces valeurs intermédiaires sur le graphe précédent (pensez à le retracer si vous l'avez fermé) en faisant

```
hold on;
plot(tprime, vx, 'k');
```

On obtient alors ce graphe :



On voit que la courbe noire interpole correctement les positions x du robot en fonction du temps.

Questions

1. Réalisez les mêmes étapes pour obtenir le vecteur vy qui sera l'interpolation de la position y du robot en fonction du temps.
2. Vous avez maintenant deux vecteurs :
   - le vecteur vx représente les abscisses intermédiaires du robot pour de nombreux instants entre 0 et 9 secondes.
   - le vecteur vy représente les ordonnées intermédiaires du robot pour de nombreux instants entre 0 et 9 secondes.

   Comment tracer, sur le graphe d'origine, les positions intermédiaires du robot et obtenir ainsi une trajectoire interpolée comme montrée ci-dessous ?