TP d'initiation à MATLAB n^0 4 : Fonctions et nombres pseudo-aléatoires

On commence d'abord par créer des fonctions. On s'intéresse ensuite à la génération de nombres pseudo-aléatoires, qui seront utilisés dans le TP suivant pour simuler des réalisations de variables aléatoires suivant les principales lois de probabilités.

Fonctions

On va maintenant créer des fonctions en MATLAB; cela demande de créer des sous-programmes (ou encore routines), qui sont appelés dans une commande (directe ou dans un programme). Lancer Matlab. On commence par ouvrir un nouveau M-fichier que vous pouvez appeler *logistique.m* (donc, tapez !*xemacs logistique.m*). Ce qui suit doit être écrit dans ce fichier.

	Première fonction: fonction logistique
$\%\ Fonction\ logistique$	Commentaire descriptif de la fonction.
$function\ y = logistique(x)$	On décrit la fonction: elle s'appelle *logistique*, elle a pour entrée x
$y = 4 * x * (1 - x);$	et pour sortie y. On définit la fonction (pas d'affichage).
	Sauvegarder alors le fichier (sous le nom *logistique.m*).

> On dénommera toujours de la même manière la fonction et le M-fichier contenant la fonction.

	Revenir maintenant dans la fenêtre de commandes de MATLAB.
$logistique(0.6)$	On calcule l'image de 0.6 par la fonction logistiques. Modifier la fonction *logistique* de manière à calculer un vecteur image Y d'un vecteur X.
	Créer un nouveau programme, appelé *tracelogis.m*.
$\%\ Trace\ de\ la$	Pas d'accents dans les commentaires !!!
$\%\ fonction\ logistique$	Commentaire descriptif du programme.
$fi = input('Finesse?')$	On choisit la qualité du tracé.
$X = -3 : fi : 3;$	On trace pour x dans $[-3, 3]$.
$Y = logis(X);$	Calcul du vecteur image Y de X.
$plot(X, Y)$	Tracé. Revenir dans la fenêtre de commandes.
	Lancer plusieurs fois le programme. Imprimer et sauvegarder un tracé.
$fplot('logistique', [-3, 3])$	Tracé de la fonction.
$quad('logistique', -3, 3)$	Calculer l'intégrale de la fonction entre -3 et 3.
	On s'intéressera un peu plus tard aux méthodes numériques utilisées pour calculer une valeur approchée (d'une intégrale, d'un zéro,...).
$fzero('logistique', 2)$	Chercher un zéro de la fonction proche de 2 (on cherche une valeur de x proche de 2 telle que $logistique(x) \simeq 0$.
$x1 = fminbnd('logistique', -3, 3)$	Chercher une valeur approchée de l'argument minimum de la fonction sur $[-3, 3]$ (donc $logistique(x1) \simeq \min_{x \in [-3,3]} logistique(x)$).
$y1 = logistique(x1)$	Valeur du minimum.
$ligos = inline('3 - logistique(x)')$	On définit une nouvelle fonction dans la session (ne sera pas mémorisée).
$x2 = fminbnd(ligos, -1, 2)$	Argument minimum de la fonction *ligos*.
$y2 = logistique(x2)$	Valeur de ce minimum.
	Ce procédé permet de trouver le maximum d'une fonction. Pourquoi ? Trouver effectivement le maximum de la fonction logistique sur $[-3, 3]$.

	Ouvrir un nouveau M-fichier.
	Deuxième fonction: fonction de Hénon
$\%\ Fonction\ de\ Henon$	Commentaire descriptif de la fonction.
$function\ [x2, y2] = henon(x1, y1, a, b)$	On décrit la fonction: elle s'appelle *henon*, elle a pour entrée $x1$, $y1$, a et b et pour sortie $x2$ et $y2$.
$x2 = 1 - a * x1\hat{}2 + y1;$	On définit les deux variables de sortie: $x2$
$y2 = b * x1;$	et $y2$. Sauvegarder alors le fichier sous le nom *henon.m*.
$[p, t] = henon(1, 1, 1, 1)$	Un calcul particulier.
$l = henon(0, 0, 1, 0)$	Ne donne que la première variable de sortie de la fonction.

Exercice

Faire un programme appelé *attracteur.m* permettant en ayant fixé a et b de transformer par itération de la fonction *henon* un point initial (x_0, y_0) (ainsi (x_0, y_0) est transformé en $(x1, y1)$, puis $(x1, y1)$ est transformé en $(x2, y2)$, $\cdots$). On choisira pour commencer $a = 1.4$, $b = 0.3$, $x_0 = 0$ et $y_0 = 0$, et on transformera 10000 fois le point (x_0, y_0) (donc on calcule (x_n, y_n) pour $n = 0, 1, \cdots, 10000$). Tracer alors l'ensemble des points de coordonnées (x_n, y_n) ainsi obtenus (ensemble appelé orbite). Faire un zoom sur ce tracé pour voir apparaître un phénomène de reproduction d'un certain motif à différentes échelles : ceci caractérise les fractales (quoique ce ne soit pas encore théoriquement démontré pour cette fonction de Hénon). Sauvegarder et imprimer cette orbite. Recommencer en modifiant légèrement a, puis (x, y). Que remarque-t-on? (utiliser par exemple $a = 1.24$ et $b = 0.3$ ou $a = 1.3$ et $b = 0.3$)

Note "historico-culturelle" : Les notions de "fractales" (le flocon de neige de Van Koch par exemple) et de "chaos" (le fameux et symbolique effet "aile de papillon" : les battements d'ailes d'un papillon aujourd'hui pourraient contribuer à provoquer un ouragan en Floride dans 3 mois...) ont actuellement le vent en poupe scientifiquement et médiatiquement. Même si des mathématiciens Français avaient développé ces notions il y a environ un siecle (Julia, Fatou et Hadamard pour les fractales, Poincaré pour le chaos), c'est un météorologue, Lorenz, en 1963, qui a essentiellement contribué au développement actuel de ces concepts. Surtout, ce sont des **simulations numériques avec un ordinateur**, à partir d'équations de convection utilisées pourtant depuis longtemps en mécanique des fluides, qui ont mis en évidence (un peu par hasard) ces phénomènes. Le système dynamique de Hénon date de 1976 et est issu d'étude sur des oscillations en régime forcé.

Nombres pseudo-aléatoires

Le problème que nous nous posons maintenant est : comment générer des nombres pris "au hasard" entre 0 et 1 ? En termes plus mathématiques, la question se pose sous la forme suivante : si $X : \Omega \to [0, 1]$ est une variable aléatoire de distribution uniforme sur $[0, 1]$, comment produire une réalisation de X, c'est-à-dire $X(\omega)$ avec $\omega \in \Omega$? (ce qu'on appellera nombre pseudo-aléatoire par la suite) Et si $(X_1, \cdots, X_n)$ sont n variables indépendantes de même loi que X, comment générer une réalisation de $(X_1, \cdots, X_n)$, soit $(X_1(\omega), \cdots, X_n(\omega))$? Ces questions se posent car un processeur fonctionne de façon intégralement déterministe, donc a priori sans possibilité de produire du "hasard". Pourtant, et c'est aussi ce que permettent les fonctions logistiques et de Hénon vues plus haut, certaines fonctions déterministes peuvent approcher le "hasard" :

	Première génération de nombres pseudo-aléatoires
$mod(27, 10)$	Fonction modulo : on calcule 27 modulo 10, soit le reste de la division euclidienne de 27 par 10 : on obtient 7 car $27 = 2 * 10 + 7$.
$mod(19, 7)$	Autre exemple. Faire suffisamment d'exemples jusqu'à avoir compris *mod*. Ouvrir un fichier *pseudo1.m* et y créer une fonction *pseudo1* telle que $y = pseudo1(x) = mod(3x + 1, 10)$. Retourner alors sur la fenêtre de commandes Matlab. Générer la suite (x_n) avec $n = 0, 1, \cdots, 20$ telle que $x_{n+1} = pseudo1(x_n)$ et $x_0 = 0$. Par ce biais, on a générer des nombres qui semblent se comporter comme des chiffres entre 0 et 9 pris au "hasard" de façon équiprobable et indépendante.

Cependant, on s'aperçoit si on génère (x_n) avec $n = 0, 1, \cdots, 100$ qu'un motif se répète de façon périodique. Lequel ? Quelle période ? Pourquoi ?
Ceci met en défaut la prétention à générer ainsi des nombres "aléatoires".
Représenter maintenant les points de coordonnées (x_n, x_{n+1}) pour $n = 0, 1, \cdots, 99$.
Quelles sont donc les couples possibles pour (x_n, x_{n+1}) ?
Faire la même chose avec les couples (x_n, x_{n+2}) et (x_n, x_{n+7}). Conclusions ?
Ceci met en défaut la prétention à générer ainsi des variables indépendantes.
Noter qu'en divisant les x_n par 10, on génèrerait ainsi des nombres de $[0, 1]$.

Seconde génération de nombres pseudo-aléatoires

Même s'il n'est pas parfait, le procédé précédent donne une idée d'un générateur de nombres pseudo-aléatoires plus performant. Ouvrir un fichier *pseudo2.m*.

$\%\ Generation\ de\ nombres$	Commentaire de la fonction.
$\%\ pseudo - aleatoires$	
$function\ y = pseudo2(x)$	La fonction est donc *pseudo2*.
$a = 7^7;\ b = 2 \wedge 31 - 1;$	On utilise deux entiers suffisamment grands.
$y = mod(a * x + 1, b)$	Comme précédemment, on utilise la fonction *mod*. On aura donc pour tout $x \in \mathbb{N}$, un entier y dans $\{0, 1, \cdots, 2^{31} - 2\}$. Après avoir sauvegarder, revenir sur la fenêtre de commandes Matlab et faire des essais. Simuler par ce procédé (x_n) pour $n = 1, \cdots, 1000$ avec $x_0 = 0$. Pour obtenir des nombres dans $[0, 1]$, il suffit de diviser les x_n par $2^{31} - 2$. Faire d'autres essais en changeant x_0. Conclusion ?
$rand(1, 1000)$	Génération de 1000 nombres pseudo-aléatoires. La fonction *rand* utilisée par Matlab utilise la fonction *pseudo2*. Déconnectez vous alors de Matlab. Pour vérifier que les nombres générés ne sont pas vraiment aléatoires, taper
$rand(1, 5)$	et noter les nombres. Ensuite, fermer Matlab puis reconnectez-vous et tapez
$rand(1, 5)$	Conclusions ? Un moyen pour éviter ce déterminisme serait d'affecter pour valeur à x_0 la date donnée par l'horloge interne de l'ordinateur. Pourquoi ? Ce n'est pas encore fait par Matlab ...