Apprendre scilab
à l'aide d'exercices

par Raymond Moché

ancien élève de l'École normale d'instituteurs de Douai et de l'École normale supérieure de Saint Cloud

ancien professeur de l'Université des Sciences et Technologies de Lille
et des universités de Coïmbre et de Galatasaray

http ://gradus-ad-mathematicam.fr

Table des matières

Chapitre 1

Remarques générales

Ceci est un ensemble d'exercices destiné aux professeurs qui pourraient être intéressés. Il est destiné soit à apprendre *scilab*, soit à trouver une solution à un problème particulier. C'est essentiellement élémentaire, mais pas toujours. Le niveau de difficulté est évalué par des *, de [*] à [****]. Bien sûr, cette classification est contestable.

Souvent, on commence à faire du calcul numérique électronique en grognant, parce qu'on en a eu besoin ou qu'on ne pouvait faire autrement. Quand on acquiert un peu d'expérience, on se rend compte que l'on peut, en calcul aussi, faire des astuces, des algorithmes plus ou moins bons ou plus ou moins mauvais, rapides ou non, gourmands en mémoire ou non. Et on y prend goût.

On a souvent un élément de comparaison, à savoir les algorithmes de la bibliothèque de programmes de *scilab*. Pour en prendre connaissance ou se renseigner sur les diverses commandes, il faut avoir un manuel (nous recommandons [11] en plus de [12]) ou consulter l'aide en ligne ([1]). Pour débuter, on peut utiliser [12]. L'aide en ligne est consultable directement depuis la console (commande `help`).

Utilisation habituelle de *scilab* (dans ce travail)

scilab est presque toujours utilisé comme suit :

- un script (ou une fonction *scilab*) est tapé dans l'éditeur de texte *SciNotes*

- il est ensuite chargé dans la console de *scilab* avec l'option « sans écho » du menu déroulant « Exécution » de la console.

- On peut taper les commandes dans la console directement dans les cas simples, mais les corrections sont difficiles. Il vaut mieux utiliser systématiquement l'éditeur de texte.

Que se passe-t-il quand on charge dans la console un script ou une fonction édités dans *SciNotes* ?

On obtient :

```
-->exec('Chemin du fichier', -1)
-->
```

L'invite `-->` indique que le calculateur *scilab* est disponible. Le paramètre -1 indique le mode d'exécution choisi, faire

```
-->help mode
```

dans la console et la touche Entrée, évidemment, pour obtenir la documentation. Il y a d'autres modes d'exécution, dont le pas à pas (mode 7).

Sur la terminologie

- Un script est une suite de commandes. Nous employons assez indifféremment les mots script, algorithme, programme. Désolé.

- *scilab* est très à l'aise avec les matrices. Une matrice est un tableau de nombres (en général) rectangulaire

à n lignes et m colonnes. Si $n = 1$, on dit que c'est une matrice-ligne ou un vecteur ou plus simplement une liste de nombres. Si $m = 1$, on dit que c'est une matrice-colonne. Si $n = m = 1$, c'est un nombre.

- Dans ce travail, dans presque tous les cas, matrice veut dire tableau de nombres. Quand un exercice utilise du calcul matriciel (en gros, la définition du produit de deux matrices ad hoc), on signale « produit matriciel » dans les mots-clefs de cet exercice.

Comment trouver des données à traiter ?

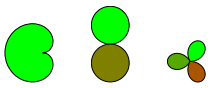
De nombreux exercices consistent à traiter des données : par exemple, ranger une liste de 100 000 nombres dans l'ordre croissant. Pour cela, il faut commencer par produire cette liste, ce qui se fait habituellement en utilisant des générateurs de nombres aléatoires parce que c'est pratique. Il est inutile de connaître un mot de Calcul des Probabilités. Il suffit de connaître la nature des nombres en question, ce qui est précisé dans les exercices concernés.

Nettoyer la console

Après avoir fait des calculs et reçu éventuellement des messages d'erreur, on peut avoir envie d'effacer la console. Cela se fait avec l'option « Effacer la console » du menu déroulant « édition ». On obtient de nouveau une belle page blanche avec seulement l'invite. Mais ce qui a été tapé ou chargé dans la console n'est pas effacé. Pour effacer les variables non-protégées, on utilise la commande `clear` (voir l'aide pour plus de détails).

Du bon usage de *scilab*

scilab est un excellent calculateur, bien supérieur aux calculatrices programmables dont certaines sont déjà remarquables. Faire un algorithme pour ranger 10 nombres dans l'ordre croissant est clairement une mauvaise utilisation de *scilab*. Ce n'est pas intéressant et peut se faire à la main. Il faut voir grand !



Chapitre 2

Liste des énoncés

Chapitre 3 : Calculs numériques non programmés

Énoncé n° 3.1 [*] : *scilab* utilisé comme une calculette.

Quelle est la valeur de π ? de e ? de $\pi \cdot e$? Calculer $\dfrac{\sin 3 \cdot e^2}{1 + \tan \frac{\pi}{8}}$.

Énoncé n° 3.2 [*] : Lignes trigonométriques remarquables.

Saisir le tableau « Arcs » des angles remarquables entre 0 et π ; calculer et afficher les sinus, cosinus et tangentes de ces angles.

Énoncé n° 3.3 [*] : Conversion de la base 2 à la base 10.

Soit n un entier $\geqslant 1$ donné sous sa forme $N = [a_0, a_1, \ldots, a_p]$ dans le système à base 2 : $a_0, \ldots, a_{p-1}$ sont chacun égaux à 0 ou à 1, $a_p = 1$ et p désigne un entier $\geqslant 0$ tels que

$$n = a_0 + a_1 \cdot 2 + \cdots + a_p \cdot 2^p$$

Problème : écrire n dans le système décimal.

Énoncé n° 3.4 [*] : Les erreurs de *scilab*.

1 - Calculer `.6*9+.6`.

2 - Calculer `floor(.6*9+.6)`. Comparer.

Chapitre 4 : Manipulations élémentaires

Énoncé n° 4.1 : [*] Échanger deux nombres.

Soit $X = [a, b]$ une liste (un vecteur) de deux nombres. Donner un script qui retourne $[b, a]$.

Énoncé n° 4.2 : [*] Échanger deux nombres dans une liste.

Écrire un script qui, étant donné une liste de nombres $X = [x_1, \ldots, x_n]$ et deux indices i et j, retourne X après avoir échangé x_i et x_j.

Énoncé n° 4.3 : [**] Échanger deux colonnes (ou deux lignes) d'un tableau de nombres rectangulaire.

Engendrer un tableau de nombres à n lignes et m colonnes à l'aide de la commande `grand(n,m,'uin',-13,14)` puis échanger les colonnes i et j (i et j donnés)

Énoncé n° 4.4 : [**] Ré-arrangement d'une suite de nombres (fantaisie).

Étant donné une liste de nombres $X = [x_1, \ldots, x_n]$, retourner la liste $[x_2, x_4, \ldots, x_1, x_3, \ldots]$ (suite des termes de rang pair de X suivis des termes de rang impair)

Énoncé n° 4.5 : [*] Combien de temps l'exécution d'un script prend-elle ?

Engendrer une liste de 100 000 nombres ; inverser l'ordre de ces nombres et retourner le temps de calcul de cette inversion.

Énoncé n° 4.6 : [***] Produit matriciel.

Soit $A = (a_{i,j})$ et $B = (b_{i,j})$ deux matrices, A ayant n lignes et m colonnes, B ayant m lignes et p colonnes (le nombre de colonnes de A est égal au nombre de lignes de B). Alors $C = A \cdot B$ (commande *scilab* : `C=A*B`) est la matrice à n lignes et p colonnes définies par

$$c_{i,j} = \sum_{k=1,\ldots,m} a_{i,k} \cdot b_{k,j}$$

1 - Soit `A=tirage_entier(0,10,7)` et `B=tirage_entier(-14,27,7)` deux listes de 7 nombres. Peut-on effectuer le produit `A*B` de ces matrices ?

2 - Calculer `B'`. Décrire cette matrice.

3 - Peut-on effectuer le produit `B*A` ? `B*A'` ? Décrire `B*A'`.

4 - Soit `C=grand(7,7,"uin",-2,8)` une matrice à 7 lignes et 7 colonnes (matrice carrée). Décrire les matrices `A*C` et `C*A'`.

Énoncé n° 4.7 : [***] Calcul de nombres de dérangements.

Le nombre de dérangements (ou permutations sans point fixe) de la liste $(1, \ldots, n)$ est donné par la formule

$$N_n = (1 \times \cdots \times n)\left(1 + \sum_{k=1}^{n} \frac{(-1)^k}{1 \times \cdots \times k}\right) \quad \text{ou} \quad N_n = n\,!\left(1 + \sum_{k=1}^{n} \frac{(-1)^k}{k\,!}\right)$$

1 - Écrire un script qui, étant donné l'entier n, retourne N_n.

2 - Quelle est la plus grande valeur de n qui permet à *scilab* d'afficher N_n exactement (c'est à dire sans passer en notations flottantes) ? Combien N_n vaut-il alors ?

Chapitre 5 : Programmer une fonction scilab

Énoncé n° 5.1 [*] : Volume du cône.

1 - Fabriquer une fonction-scilab qui calcule le volume v d'un cône de révolution de hauteur h et dont la base est un disque de rayon r.

2 - Cette fonction accepte-t-elle que r et h soient des listes de nombres (ce qui serait utile si on voulait faire varier le rayon et la hauteur) ?

Énoncé n° 5.2 [*] : Aire d'un trapèze.

Fabriquer une fonction-scilab qui donne l'aire d'un trapèze de bases b et B et de hauteur h.

Énoncé n° 5.3 [*] : Moyenne et variance empiriques.

1 - Calculer la moyenne et la variance du vecteur

$$[\sqrt{3}, -2.7, 4^2, \log(7), \exp(0.8), -4.2, \pi, 17, 13/11, \cos(4), 0]$$

.

2 - Engendrer une liste de `n=50` nombres entiers entre les bornes `a=-7` et `b=39` à l'aide de la commande `tirage_entier(n,a,b)`, puis calculer sa moyenne et sa variance.

Énoncé n° 5.4 [**] : Conversion de temps.

Programmer une fonction-*scilab* qui, étant donné une durée d en secondes, la retourne sous forme d'années, mois, jours, heures, minutes et secondes.

Chapitre 6 : Boucles `pour`, graphes simples

Énoncé n° 6.1 : [*] Tables de multiplication.

Afficher les tables de multiplication par 1, ..., 9.

Énoncé n° 6.2 : [*] Minimum et maximum d'une liste de nombres.

Une liste de nombres étant donnée, calculer son minimum et son maximum.

Énoncé n° 6.3 : [*] Quelques produits de matrices.

1 - Sont donnés 2 vecteurs de n nombres : $X = [x_1, \ldots, x_n]$ et $P = [p_1, \ldots, p_n]$. On demande d'expliciter les résultats des produits matriciels suivants :

1.a - `X*ones(n,1)`

1.b - `X*X'`

1.c - `X*diag(X)*ones(n,1)`,

1.d - `X*diag(P)*X'`,

1.e - `P*diag(X)*X'`.

2 - Calculer la valeur de ces expressions avec *scilab* lorsque $X = [1, \sqrt{3}, \exp(-0.5), 2, \frac{3}{7}]$ et $P = [\frac{1}{2}, 0.23, \log 5, \frac{\pi}{8}, 7]$.

Énoncé n° 6.4 : [*] Construction de matrices.

1 - Écrire la matrice à n lignes et m colonnes dont la première colonne ne contient que des 1, la deuxième colonne ne contient que des 2, *etc.*

2 - Écrire la matrice à m lignes et n colonnes dont la première ligne ne contient que des 1, la deuxième ligne ne contient que des 2, *etc.*

Énoncé n° 6.5 : [*] Sommes de nombres entiers.

1 - Calculer la somme des nombres entiers de 1 à 500.

2 - Calculer la somme des carrés des nombres entiers de 1 à 500.

Énoncé n° 6.6 : [*] Trier un tableau numérique (fantaisie).

Trier un tableau numérique dans l'ordre décroissant (quand on le lit de la gauche vers la droite et de haut en bas, de la première à la dernière ligne).

Énoncé n° 6.7 : [*] Matrice fantaisie (boucles `pour` imbriquées).

1 - Écrire la matrice (n,n) dont les éléments sont 1, 2, ..., n^2 écrits dans l'ordre habituel (sur chaque ligne, de la gauche vers la droite, de la première à la dernière ligne).

2 - Cette matrice étant notée M, expliquer comment se font les calculs suivants : N=M+M, Q=3*M, R=M^2.

Énoncé n° 6.8 : [*] Matrices de Toeplitz.

Écrire la matrice carrée de taille n comprenant des n sur la diagonale principale, des $n-1$ sur les deux lignes qui l'encadrent, etc.

Énoncé n° 6.9 : [*] Graphe de la fonction tangente entre 0 et π.

Tracer le graphe de la fonction tangente entre 0 et π, la variable variant avec un pas de 0.1, puis 0.01, puis 0.001 et enfin 0.0001.

Énoncé n° 6.10 : [*] Sauts de puce.

Une puce fait des bonds successifs indépendants les uns des autres en ligne droite. La longueur de chaque bond est aléatoire. On considère que c'est un nombre choisi au hasard dans l'intervalle $[0, 5[$, l'unité de longueur étant le cm. On note la distance totale parcourue au bout de m bonds. On répète cette expérience n fois. Calculer la distance moyenne parcourue au cours de ces n expériences.

Énoncé n° 6.11 : [**] Résolution graphique d'une équation.

1 - Calculer les valeurs de $f(x) = \dfrac{x^3 \cdot \cos(x) + x^2 - x + 1}{x^4 - \sqrt{3} \cdot x^2 + 127}$ lorsque la variable x prend successivement les valeurs -10, -9.2, -8.4, ..., 8.4, 9.2, 10 (pas 0.8).

2 - Tracer le graphe de la fonction f quand x varie entre -10 et 10 en utilisant seulement les valeurs de $f(x)$ calculées précédemment.

3 - Apparemment, l'équation $f(x) = 0$ a une solution α voisine de 2. En utilisant le zoom, proposer une valeur approchée de α.

4 - Tracer de nouveau le graphe de f entre -10 et 10 en faisant varier x avec un pas de 0.05.

5 - Ce nouveau graphe amène-t-il à corriger la valeur de α proposée ?

Énoncé n° 6.12 : [***] Construction de matrices : matrice-croix (fantaisie).

Écrire la matrice carrée C de taille 2n+1 comportant des 1 sur la $(n+1)^{\text{ième}}$ ligne et la $(n+1)^{\text{ième}}$ colonne et des 0 ailleurs.

Énoncé n° 6.13 : [**] Construction de matrices : matrice Zorro (fantaisie).

Écrire la matrice Zorro, matrice carrée de taille n comprenant des 1 sur la première ligne, sur la deuxième diagonale et sur la dernière ligne et des 0 partout ailleurs.

Énoncé n° 6.14 : [**] Construction de matrices : matrice barrée (fantaisie).

Écrire la matrice carrée de taille n portant des 1 sur les 2 diagonales, des 0 ailleurs.

Énoncé n° 6.15 : [**] Aiguilles de l'horloge.

1 - Combien de fois l'aiguille des minutes tourne-t-elle plus vite que l'aiguille des heures ?

2 - Quand l'aiguille des heures a tourné de α degrés à partir de midi, l'extrémité de l'aiguille des minutes est repérée sur le cadran de l'horloge par l'angle

$$\beta = 12 \cdot \alpha - 360 \cdot \left[\frac{12 \cdot \alpha}{360} \right]$$

Tracer le graphe de β en fonction de α quand celui-ci varie de 0 à 360 degrés.

3 - Combien y a-t-il de superpositions des aiguilles entre midi et minuit ? Calculer les angles correspondants.

4 - Que se passe-t-il à partir de minuit ?

Énoncé n° 6.16 : [**] Calculs d'effectifs et de fréquences.

Une suite S de nombres entiers étant donnée, calculer l'effectif et la fréquence de n dans cette suite, quel que soit n entre $\min(S)$ et $\max(S)$ ($\min(S) \leqslant n \leqslant \max(S)$).

Énoncé n° 6.17 : [**] Écrire à la fin le premier terme d'une liste de nombres.

Écrire un script qui associe à tout vecteur $X = [x_1, \ldots, x_{n-1}, x_n]$ le vecteur $Y = [x_2, \ldots, x_n, x_1]$.

Énoncé n° 6.18 : [*] Écrire une suite de nombres à l'envers.

Écrire un script qui transforme une liste de nombres donnée $X = [x_1, \ldots, x_n]$ en la liste $Y = [x_n, \ldots, x_1]$.

Énoncé n° 6.19 : [***] Permutations (solution matricielle).

Écrire toutes les permutations de 1, 2, ..., n.

Énoncé n° 6.20 : [***] Matrices de permutation.

1 - Une permutation σ de l'ensemble $(1, 2, \ldots, n)$ étant donnée, écrire la matrice de permutation P de σ (voir les explications ci-dessous).

2 - Soit X un vecteur de longueur n. Vérifier sur un exemple que $Y = X \cdot P$ s'obtient en permutant selon σ les composantes de X.

3 - Quelle matrice de permutation M permet d'écrire les composantes de X dans l'ordre inverse ?

Énoncé n° 6.21 : [***] Échanger deux lignes ou deux colonnes d'un tableau (produit d'une matrice par une matrice de permutation).

1 - M étant un tableau de nombres à n lignes et m colonnes,

1.a - soit σ_1 une permutation de $(1, \ldots, m)$ (au choix du lecteur) et P_1 la matrice de permutation qui lui est associée (voir l'exercice 6.20) ; comparer M et $M * P_1$ (effet sur une matrice de la multiplication à droite par une matrice de permutation) ;

1.b - soit σ_2 une permutation de $(1, \ldots, n)$ (au choix du lecteur) et P_2 la matrice de permutation qui lui est associée ; comparer M et $P_2 * M$ (effet sur une matrice de la multiplication à gauche par une matrice de permutation) ;

2 - Engendrer un tableau de nombres entiers M à n lignes et à m colonnes à l'aide de la commande `grand(n,m,"uin",-13,14)`.

2 .a - Échanger les lignes 2 et $n - 1$,

2.b - Échanger les colonnes 2 et $m - 1$.

Énoncé n° 6.22 : [**] Calcul matriciel de l'espérance et de la variance d'une variable aléatoire finie.

1 - Soit X une variable aléatoire ne prenant que les valeurs v_1, ..., v_n avec respectivement les probabilités p_1, ..., p_n (> 0). Exprimer matriciellement son espérance *moy* et sa variance *var* à l'aide des vecteurs $V = [v_1, \ldots, v_n]$ et $prob = [p_1, \ldots, p_n]$.

2 - Application : calculer l'espérance et la variance d'une variable aléatoire suivant la loi binomiale de taille 10 et de paramètres $\frac{1}{3}$.

Énoncé n° 6.23 : [***] Calcul matriciel de la covariance de 2 variables aléatoires finies.

Soit X et Y deux variables aléatoires finies, $V_X = [x_1, \ldots, x_n]$ et $V_Y = [y_1, \ldots, y_m]$ les vecteurs des valeurs qu'elles prennent avec des probabilités > 0. La loi du couple aléatoire (X, Y) est décrite par la matrice $P_{(X,Y)} = (p_{i,j},\ 1 \leqslant i \leqslant n\,;\ 1 \leqslant j \leqslant m)$ où $p_{i,j} = P(X = x_i\,, Y = y_j)$.

1 - Comment calcule-t-on la loi P_X de X à partir de $P_{(X,Y)}$? Même question pour la loi P_Y de Y.

2 - Exprimer matriciellement les espérances m_X et m_Y de X et Y.

3 - Exprimer matriciellement la variance $\mathrm{var}(X)$ de X.

4 - Exprimer matriciellement la covariance $\mathrm{cov}(X, Y)$ de X et Y.

5 - Application : Calculer les quantités ci-dessus lorsque $P_{(X,Y)}$, V_X et V_Y sont respectivement égales à :

$P_{(X,Y)} =$

0.002	0.012	0.004	0.005	0.017	0.012	0.004	0.001	0.003	0.001	0.005	0.	0.009	0.002	0.001
0.011	0.002	0.003	0.01	0.009	0.002	0.009	0.002	0.002	0.006	0.002	0.003	0.011	0.007	0.006
0.003	0.001	0.007	0.001	0.009	0.003	0.009	0.005	0.006	0.004	0.002	0.004	0.003	0.011	0.01
0.001	0.002	0.007	0.012	0.002	0.	0.	0.002	0.003	0.008	0.006	0.009	0.005	0.002	0.008
0.01	0.011	0.001	0.01	0.004	0.004	0.	0.	0.004	0.	0.012	0.	0.005	0.01	0.009
0.001	0.	0.004	0.012	0.001	0.012	0.001	0.004	0.011	0.01	0.005	0.004	0.004	0.012	0.007
0.002	0.012	0.003	0.004	0.006	0.005	0.	0.01	0.004	0.012	0.003	0.004	0.	0.004	0.01
0.009	0.011	0.012	0.009	0.005	0.021	0.011	0.02	0.011	0.003	0.003	0.007	0.008	0.005	0.005
0.006	0.005	0.001	0.004	0.008	0.012	0.007	0.008	0.012	0.	0.	0.006	0.001	0.004	0.001
0.	0.006	0.012	0.01	0.002	0.011	0.006	0.011	0.023	0.023	0.009	0.003	0.011	0.01	0.007
0.003	0.013	0.	0.003	0.002	0.008	0.009	0.006	0.021	0.	0.004	0.003	0.005	0.002	0.007

$$V_X = [1., 1.3, 1.6, 1.9, 2.2, 2.5, 2.8, 3.1, 3.4, 3.7, 4.] \text{ et}$$
$$V_Y = [-4., -3.3, -2.6, -1.9, -1.2, -0.5, 0.2, 0.9, 1.6, 2.3, 3., 3.7, 4.4, 5.1, 5.8]$$

Énoncé n° 6.24 : [***] Ranger n nombres donnés dans l'ordre croissant (tri à bulles).

Soit une suite de nombres $A = [a_1, \ldots, a_n]$ à ranger dans l'ordre croissant. On appellera *Aneuf* la suite obtenue. Dans le tri à bulles, on fait une suite de passages sur A. Au premier passage, on considère la paire (a_1, a_2) que l'on met dans l'ordre croissant, ce qui modifie A, puis (a_2, a_3), *etc.* À la fin du premier passage, le dernier terme de *Aneuf* est placé. On continue ces passages qui placent définitivement l'avant dernier terme de *Aneuf*, puis le précédent, *etc.* Après le $(n-1)^{\text{ème}}$ passage, *Aneuf* est obtenu.

On demande une fonction *scilab* qui range A dans l'ordre croissant par le tri à bulles.

Énoncé n° 6.25 : [****] Ranger n nombres donnés dans l'ordre croissant.

Soit une suite d'au moins 3 nombres `A=[a_1,…,a_n]` à ranger dans l'ordre croissant. On appellera `Atri` la suite obtenue. Si l'on ne considère que les 2 premiers termes de la suite, `Atri=[min(A([1:2]),max(A([1:2])))]`. Si on tient compte du terme suivant, on va l'introduire dans `Atri` comme suit : on écrit d'abord les termes de `Atri` qui lui sont strictement inférieurs, puis ce nombre, puis les termes de `Atri` qui lui sont supérieurs ou égaux. Ceci nous fournit la nouvelles valeur de `Atri`. Et ainsi de suite. On demande une fonction *scilab* qui range `A` dans l'ordre croissant par ce procédé.

Énoncé n° 6.26 : [**] Trier un tableau suivant l'une de ses lignes.

Voici la liste des fréquences des 26 lettres de l'alphabet dans la langue française :
7.68, 0.80, 3.32, 3.60, 17.76, 1.06, 1.10, 0.64, 7.23, 0.19, 0.00, 5.89, 2.72, 7.61, 5.34, 3.24, 1.34, 6.81, 8.23, 7.30, 6.05, 1.27, 0.00, 0.54, 0.21, 0.07.
Le problème est de ranger ces lettres par ordre des fréquences décroissantes.

Plus généralement, étant donné une matrice numérique M à n lignes et m colonnes et un entier j, $1 \leqslant j \leqslant n$, ranger les colonnes de `M` de manière que la ligne j soit rangée dans l'ordre décroissant.

Chapitre 7 : Instructions conditionnelles

Énoncé n° 7.1 : [*] Fonction valeur absolue.

1 - Programmer une fonction-scilab qui donne la valeur absolue de tout nombre réel
- à l'aide d'une instruction conditionnelle avec alternative,
- à l'aide d'une instruction conditionnelle sans alternative.

2 - En déduire la valeur absolue de -0.87 et de $\sqrt{3}$.

Énoncé n° 7.2 : [*] Calculer le maximum de 2 ou 3 nombres donnés.

Écrire une fonction *scilab* qui retourne le maximum de 2 nombres `x` et `y`, puis une fonction *scilab* qui retourne le maximum de 3 nombres `x`, `y` et `z` donnés.

Énoncé n° 7.3 : [*] Ranger 2 nombres dans l'ordre croissant.

Écrire une fonction *scilab* qui, étant donné deux nombres réels, les retourne rangés dans l'ordre croissant.

Énoncé n° 7.4 : [*] Ranger 3 nombres dans l'ordre croissant.

Ranger trois nombres réels donnés dans l'ordre croissant.

Énoncé n° 7.5 : [**] Sauts de kangourous.

Un kangourou fait habituellement des bonds de longueur aléatoire comprise entre 0 et 9 mètres.

1 - Combien de sauts devra-t-il faire pour parcourir 2000 mètres ?

2 - Fabriquer une fonction-scilab qui à toute distance d exprimée en mètres associe le nombre aléatoire N de sauts nécessaires pour la parcourir.

3 - Calculer le nombre moyen de sauts effectués par le kangourou quand il parcourt T fois la distance d.

Énoncé n° 7.6 : [*] Addition en base 2.

Soit deux entiers n, $m \geqslant 0$ notés en base 2 sous la forme $[a_0, \ldots, a_{k-1}, a_k]$ et $[b_0, \ldots, b_{k-1}, b_l]$.
Cela signifie que ce sont deux suites de 0 et de 1 qui vérifient

$$n = a_0 \cdot 2^0 + a_1 \cdot 2^1 + \cdots + a_k \cdot 2^k \text{ et } m = b_0 \cdot 2^0 + b_1 \cdot 2^1 + \cdots + b_l \cdot 2^l$$

Le problème posé est de calculer leur somme en travaillant dans la base 2.

Énoncé n° 7.7 : [**] L'année est-elle bissextile ?

1 - L'année n est-elle bissextile ?

2 - Soit n et m deux millésimes ($n \leqslant m$). Combien y a-t-il d'années bissextiles de l'année n à l'année m (années n et m comprises) ?

Énoncé nº 7.8 : [**] Calcul et représentation graphique de fréquences (exercice type du Calcul des probabilités)

On tire 10000 nombres au hasard dans l'intervalle $[0, 1]$. À chaque tirage, on se demande si l'événement

$$\text{A : « le nombre tiré au hasard appartient à l'intervalle } [\frac{1}{7}, \frac{5}{6}] \text{ »}$$

est réalisé.

1 - Combien vaut la probabilité p de A ?

2 - Calculer la fréquence de réalisation de A après chaque tirage. Tracer le graphe des fréquences.

3 - Choisir la dernière fréquence calculée comme valeur approchée de p.

Énoncé nº 7.9 : [**] Le point M est-il à l'intérieur du triangle ABC ?

Étant donné un triangle $A_1A_2A_3$ et un point M (par leurs coordonnées), fabriquer une fonction qui indique si M est à l'intérieur du triangle ou non.

Énoncé nº 7.10 : [**] Ce triangle est-il isocèle ?

Un triangle ABC est donné par les coordonnées de ses sommets dans un repère orthonormé. Écrire un algorithme qui permette d'en déduire qu'il est isocèle (sans être équilatéral), équilatéral ou quelconque (c'est à dire, ici, ni isocèle, ni équilatéral).

Chapitre 8 : Boucle tant que

Énoncé nº 8.1 : [*] Transformer une boucle `pour` en une boucle `tant que`.

Calculer le maximum d'une liste de nombres d'après la fonction `Min` de 6.2, puis remplacer la boucle `pour` par une boucle `tant que`.

Énoncé nº 8.2 : [*] Calculer le maximum d'une liste de nombres.

On veut calculer le maximum d'une liste de nombres `L` donnée, de longueur au moins 2, de la manière suivante : si `L(2)` est plus grand que `L(1)`, on échange ces nombres ; ensuite, on efface tous les termes qui sont inférieurs ou égaux à `L(1)` à partir du rang 2. Ce faisant, la longueur de `L` diminue mais reste supérieure ou égale à 1. Si elle est encore supérieure ou égale à 2, on recommence. Quand elle est égale à 1, le terme restant est le maximum recherché.

Énoncé nº 8.3 : [*] Conversion du système décimal vers le système à base b.

Un nombre entier $n \geqslant 0$ est donné sous forme décimale. Le problème posé est de l'écrire dans le système à base $b \geqslant 2$, c'est à dire sous la forme d'une liste $[a_0, a_1, \ldots, a_p]$ de 0, de 1, ..., de $b-1$ de sorte que $n = a_0 \cdot b^0 + a_1 \cdot b^1 + \cdots + a_p \cdot b^p$.

Énoncé nº 8.4 : [*] Sommes de nombres impairs.

Calculer la somme `S(n)` des nombres impairs inférieurs ou égaux à un entier donné n.

Énoncé nº 8.5 : [*] Partie entière d'un nombre réel.

Calculer la partie entière d'un nombre réel donné.

Énoncé nº 8.6 : [*] Formule de la somme des carrés.

La formule $1^2 + \cdots + n^2 = \dfrac{n(n+1)(2n+1)}{6}$ est-elle vraie pour tout entier $n \geqslant 1$?

Énoncé n° 8.7 : [**] Comptage de nombres factoriels.

1 - Combien y a-t-il de nombres factoriels inférieurs ou égaux à 10^6, à 10^9, à 10^{12}, à $(17\,!)$?
2 - 3 629 300 et 39 916 800 sont-ils des nombres factoriels ?

Énoncé n° 8.8 : [***] Ranger 3 nombres donnés dans l'ordre croissant (tri à bulles).

Pour ranger 3 nombres donnés x, y et z dans l'ordre croissant, on considère d'abord x et y que l'on échange si $x > y$; puis on considère y et z et on fait de même ; on compte aussi le nombre de permutations de 2 nombres consécutifs que l'on a réalisées au cours de ce premier passage. Si c'est 0, c'est terminé. Sinon, on fait un deuxième passage et ainsi de suite, éventuellement. On demande d'écrire la suite x, y, z dans l'ordre croissant et d'afficher le nombre de permutations faites.

Énoncé n° 8.9 : [***] Ranger n nombres donnés dans l'ordre croissant.

Programmer une fonction *scilab* dont l'entrée est une liste de nombres `L`, qui retourne cette liste rangée dans l'ordre croissant, notée `R`, fabriquée comme suit : le premier terme de `R` est `min(L)`. Ensuite, on efface de la liste `L` un terme égal à `min(L)` (il peut y en avoir plusieurs). Le second terme de `R` est le minimum des termes de `L` restants, et ainsi de suite.

Énoncé n° 8.10 : [****] Ranger n nombres donnés dans l'ordre croissant.

Programmer une fonction de tri exécutable par *scilab* à partir de la description suivante :
Un vecteur de nombres noté `A` étant donné ; on appellera `Aneuf` ce vecteur rangé dans l'ordre croissant. On détermine d'abord le plus petit nombre `m` de `A`, on compte combien de fois `n` le nombre `m` figure dans `A`, on ajoute à `Aneuf` (vide au début) la suite `m`, ..., `m` (`m` écrit `n` fois), on efface tous les `m` qui figurent dans `A`. Si `A` n'est pas vide, on recommence.

Chapitre 9 : Graphes

Énoncé n° 9.1 : [*] Tracer un triangle.

Tracer le triangle ABC, ces points ayant respectivement pour coordonnées $(1, 3)$, $(-2, 4)$ et $(3, 8)$.

Énoncé n° 9.2 : [***] Tracés de polygones réguliers inscrits dans un cercle.

Tracer dans le même repère orthonormé le cercle de centre O et de rayon 1 ainsi que le triangle équilatéral, le carré et l'heptagone régulier inscrits dans ce cercle et dont un sommet est le point $(1, 0)$.

Énoncé n° 9.3 : [***] Tracer les N premiers flocons de Koch.

Tracer les N premiers flocons de Koch définis comme suit à partir de l'hexagone régulier ABCDEFA (ou K_0) inscrit dans le cercle unité dont le point A de coordonnées $(1, 0)$ est un sommet :
- Construction du premier flocon de Koch K_1 :
• d'abord on remplace le côté AB de l'hexagone par la ligne brisée AMGNB, M étant au tiers de AB à partir de A, N étant au tiers de AB à partir de B, le triangle MNG étant équilatéral, le point G étant à droite quand on se déplace de A à B ;
• puis on fait de même avec les 5 autres côtés de K_0.
Le polygone à 18 côtés ainsi obtenu est appelé premier flocon de Koch, noté K_1.
- Construction des flocons de Koch suivants : on passe de K_n à K_{n+1} comme on est passé de K_0 à K_1.

Énoncé n° 9.4 : [***] Visualisation du tri à bulles.

On applique le tri à bulles à une suite donnée de n nombres $X = (x_1, \ldots, x_n)$ obtenue, pour fixer les idées, à l'aide de la commande `tirage_reel(n,0,1)`, n pouvant éventuellement varier. Représenter l'état de la suite X après chaque passage, y compris le dernier qui produit une illustration de la suite triée, en traçant, dans un repère orthonormé, les $(n-1)$ points (x_1, x_2), (x_2, x_3), ..., ainsi que la droite $y = x$.

Chapitre 10 : scilab vs Xcas

Énoncé n° 10.1 : [*] Calcul numérique avec *scilab* et *Xcas* (test d'égalité de deux nombres).

Vérifier si *scilab* et *Xcas* donnent à $\tan\left(\frac{\pi}{4}\right)$ et à $\left(\sqrt{2}\right)^2$ les valeurs 1 et 2 respectivement.

Énoncé n° 10.2 : [*] Calculs en nombres rationnels avec *scilab* et avec *Xcas*.

Calculer $\dfrac{\frac{1}{2}+\frac{1}{3}+\frac{1}{4}}{\left(\frac{1}{2}\right)^2+\left(\frac{1}{3}\right)^2+\left(\frac{1}{4}\right)^2}$ avec *scilab* puis *Xcas*.

Énoncé n° 10.6 : [*] Les fonctions racine carrée et élévation au carré sont-elles inverses l'une de l'autre ?

Calculer $\sqrt{x}^2$ et $\sqrt{x^2}$.

Énoncé n° 10.4 : [*] Calcul de factorielles.

Calculer $n\,!$ pour $n = 10,\ 20,\ 50,\ 100,\ 500,\ 1000$ à l'aide de *scilab* puis de *Xcas*.

Énoncé n° 10.5 : [*] Longueur d'un nombre entier.

Avec combien de chiffres s'écrit $500\,!$?

Énoncé n° 10.6 : [**] Somme des chiffres d'un nombre entier.

Un entier $n \geqslant 1$ étant donné, calculer son chiffre des unités noté $u\,(n)$ et la somme de ses chiffres notée $s\,(n)$.
Application : démontrer que $u\left(\frac{3^{2014}-1}{2}\right) = s\left(s\left(s\left(\frac{3^{2014}-1}{2}\right)\right)\right)$.

Énoncé n° 10.7 : [*] Sommes de puissances des premiers entiers (démontrer une infinité d'égalités)

Peut-on démontrer que $1 + \cdots + n = \dfrac{n\,(n+1)}{2}$ avec *scilab* ? avec *Xcas* ?
Quelles formules donnent la somme des carrés ? des cubes ? des puissances suivantes ?

Énoncé n° 10.8 : [**] Test d'isocélité et d'équilatéralité d'un triangle.

Écrire un script *scilab* et un script *Xcas* qui indiquent si un triangle donné est équilatéral, isocèle (sans être équilatéral) ou quelconque (c'est à dire non-isocèle).

Chapitre 11 : Plus de scilab

Énoncé n° 11.1 : [****] Crible d'Ératosthène.

On demande une fonction *scilab* qui, à tout nombre entier $n \geqslant 2$ associe la liste des nombres premiers qui lui sont inférieurs ou égaux, rangés dans l'ordre croissant.

Énoncé n° 11.2 : [*] Test de primalité (suite de l'exercice 11.1)

Étant donné un entier $n \geqslant 2$, en utilisant la fonction `Erato.sci` de l'exercice 11.1 ou la commande `liste_premiers`, fabriquer un algorithme qui répond si n est ou non un nombre premier.

Énoncé n° 11.3 : [**] Nombre premier suivant (suite de l'exercice 11.1).

1 - Fabriquer un algorithme qui associe à tout entier $n \geqslant 2$ le plus petit nombre premier P_n qui lui est strictement supérieur.

2 - Quel est le premier nombre premier qui suit 10 000 000 ?

Énoncé n° 11.4 : [**] Factorisation en nombres premiers (suite de l'exercice 11.1).

Fabriquer un algorithme qui à tout entier $n \geqslant 2$ associe la suite de ses facteurs premiers et la suite de leurs puissances respectives.

Énoncé n° 11.5 : [***] Peut-on faire l'appoint ?

Quelqu'un doit payer un achat. Le commerçant n'ayant pas de monnaie lui demande de faire l'appoint, c'est à dire de donner exactement la somme due. Est-ce que c'est possible ? Si oui, quelles pièces doit-il donner ?

Énoncé n° 11.6 : [***] Records.

Soit S une suite de nombres entiers. On appelle record de S son premier élément ainsi que tout terme de cette suite qui est strictement plus grand que tous les termes qui le précèdent.
Le problème est de produire la liste des records de S à l'aide d'un algorithme.

Énoncé n° 11.7 : [***] Un problème d'arithmétique : les citrons.

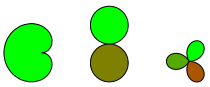
Une paysanne indienne va au marché en bicyclette pour vendre ses citrons. Un passant distrait la bouscule. Le panier de citrons est renversé. Le fautif entreprend de ramasser les citrons.

Il demande à la paysanne : « combien y en avait-il ? »

Elle lui répond : « en les rangeant par deux, il en reste un ; en les rangeant par trois, il en reste un ; en les rangeant par quatre, il en reste un ; en les rangeant par cinq, il en reste un ; en les rangeant par six, il en reste un ; mais en les rangeant par sept, il n'en reste pas. »

« C'est bon » lui dit le passant, « je les ai tous ramassés ».

Sachant qu'il ne pouvait pas y avoir plus de 500 citrons dans le panier (ce qui fait environ 50 kg), combien y en avait-il exactement ?



Chapitre 3

Calculs numériques non programmés

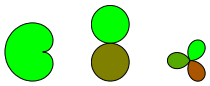
Liste des exercices :

Énoncé n° 3.1 [*] : *scilab* utilisé comme une calculette.

Énoncé n° 3.2 [*] : Lignes trigonométriques remarquables.

Énoncé n° 3.3 [**] : Conversion de la base 2 à la base 10.

Énoncé n° 3.4 [*] : Les erreurs de *scilab*.



3.1 [*] *scilab* utilisé comme une calculette

Quelle est la valeur de π ? de e ? de $\pi \cdot e$? Calculer $\dfrac{\sin 3 \cdot e^2}{1 + \tan \frac{\pi}{8}}$.

Mots-clefs : constantes pré-enregistrées, calcul numérique, `%e`, `%pi`, `sin`, `tan`, `afficher, format`.

Commentaires : On utilise ici *scilab* comme une calculatrice ordinaire. Les règles de priorité des différentes opérations sont les règles habituelles. Quand les commandes à taper sont vraiment simples et peu nombreuses, on peut utiliser directement la feuille de travail *scilab* (appelée *console*). Sinon, il vaut mieux passer par l'intermédiaire de son éditeur de texte *SciNotes*. *SciNotes* sait faire ce que font tous les éditeurs de texte. Les corrections éventuelles sont ainsi simplifiées. De plus, il aide aussi considérablement l'utilisateur en complétant automatiquement certaines instructions (déclaration de fonction, boucles, instructions conditionnelles) et en procédant à des indentations qui augmentent considérablement la lisibilité des scripts.

La procédure recommandée est donc de taper le script dans *SciNotes*, de l'enregistrer puis de le charger dans la console, par exemple en choisissant l'option « sans écho » du menu déroulant « Exécution » de la console. L'invite `-->` de la console indique que *scilab* est prêt pour exécuter de nouvelles commandes.

Pour trouver la valeur approchée que *scilab* donne à π avec les réglages de format d'affichage par défaut, il suffit de taper `%pi` juste après l'invite $(--)$ de la console, puis d'utiliser la touche « Entrée » de votre ordinateur. Nous avons tapé `a=%pi` parce que nous avons désiré appeler `a` la variable qui prendra la valeur recherchée. Idem pour la valeur *scilab* par défaut de e, ici nommée `viventlesmaths`. Nous avons oublié de nommer la variable qui prendra la valeur de $e * \pi$. Ce n'est pas grave. *scilab* lui donne un nom par défaut qui est `ans` (pour answer). Attention : si on fait une deuxième oubli de ce genre, `ans` prendra la valeur nouvellement calculée. L'ancienne valeur sera perdue. Nous avons appelée `x` la variable qui va porter la valeur de `ans`. Le résultat de cette commande n'apparaît pas sur la console parce que la ligne de commande a été terminée par un `;`. Puis, nous avons défini une matrice-ligne `y` de longueur 3 (ou matrice (1,3)), cf. 1. `y` a été affiché par défaut dans le système décimal (16 signes dont + omis et le point). Ensuite, avec la commande `format`, nous avons affiché `y` de diverses façons (affichage décimal à 20 signes puis affichage à virgule flottante à 10 signes).

```
-->a=%pi
 a  =
        3.1415926535898
-->viventlesmaths=%e
 viventlesmaths  =
        2.718281828459
-->%e*%pi
 ans  =
        8.5397342226736
-->x=ans;
-->y=[a,viventlesmaths,x]
 y  =
        3.1415926535898    2.718281828459    8.5397342226736
-->format('v',20);y
 y  =
        3.14159265358979312    2.71828182845904509    8.53973422267356597
-->format('e',10);y
 y  =
        3.142D+00    2.718D+00    8.540D+00
-->
```

Le lecteur comprendra facilement qu'il n'est ni possible ni souhaitable de continuer à donner autant de détails. Se reporter à un manuel, par exemple [12], et essayer d'être aidé par un(e) collègue ou lors de stages.

Aide en ligne

L'aide en ligne de *scilab pour les lycées* a beaucoup progressé et est en grande partie rédigée en français. L'aide en ligne de *scilab* est assez pénible, souvent mal rédigée. Il faut dire à la décharge de *scilab* que de nombreuses commandes comprennent beaucoup d'options, ce qui n'apparaîtra pas dans cet ouvrage simple. À titre d'exemple, pour se renseigner sur la commande `format` (qui est un format d'affichage, à ne pas confondre avec la précision des calculs), on tape dans la console la commande

```
--> help format;
-->
```

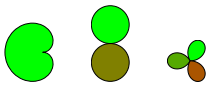
Fin de l'exercice : pour calculer $\dfrac{\sin 3 \cdot e^2}{1+\tan\frac{\pi}{8}}$, on tape le script suivant dans *SciNotes*

```
clear
x=sin(3)*%e^2/(1+tan(%pi/8));
afficher(x)
```

`sin` désigne ici la fonction *sinus*, les angles étant exprimés en radians ; `tan` la fonction *tangente*. Voir [12], qui se termine par un répertoire des « Fonctions scilab utiles au lycée ». On obtient dans la console :

```
-->exec('Chemin du fichier', -1)
    0.7373311103637
-->
```

(avec le format d'affichage par défaut).



3.2 [*] Lignes trigonométriques remarquables

Saisir le tableau « Arcs » des angles remarquables entre 0 et π ; calculer et afficher les sinus, cosinus et tangentes de ces angles.

Mots-clefs : saisie d'un tableau à la main, `sin`, `cos`, `tan`.

Dans *SciNotes*, on tape :

```
// Sinus, cosinus et tangentes remarquables
Arcs=[0,%pi/6,%pi/4,%pi/3,%pi/2,2*%pi/3,3*%pi/4,5*%pi/6,%pi];
Sin=sin(Arcs);
Cos=cos(Arcs);
Tan=tan(Arcs);
afficher('Les valeurs remarquables du sinus entre 0 et pi sont ')
afficher(Sin);
afficher('Les valeurs remarquables du cosinus entre 0 et pi sont ')
afficher(Cos);
afficher('Les valeurs remarquables de la tangente entre 0 et pi sont ')
afficher(Tan)
```

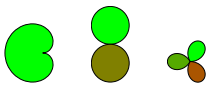
ce qui donne, en chargeant ce script dans la console :

```
-->exec('/Chemin du fichier', -1)
Les valeurs remarquables du sinus entre 0 et pi sont
      column 1 to 5
   0.    0.5    0.7071067811865    0.8660254037844    1.
      column 6 to 9
   0.8660254037844    0.7071067811865    0.5    1.224646799D-16
 Les valeurs remarquables du cosinus entre 0 et pi sont
      column 1 to 4
    1.    0.8660254037844    0.7071067811865    0.5
      column 5 to 8
    6.123233996D-17  - 0.5  - 0.7071067811865  - 0.8660254037844
      column 9
    - 1.
 Les valeurs remarquables de la tangente entre 0 et pi sont
      column 1 to 4
    0.    0.5773502691896    1.    1.7320508075689
      column 5 to 8
    1.633123935D+16  - 1.7320508075689  - 1.  - 0.5773502691896
      column 9
    - 1.224646799D-16
-->
```

Commentaires sur les résultats

- ✓ Le tableau `Arcs` est ici un tableau à une ligne et 9 colonnes.
- ✓ Ce tableau a été saisi à la main. Ses éléments sont séparés par des `,`. Si on avait voulu l'écrire en colonne, on aurait utilisé des `;`. `;` signifie un changement de ligne.
- ✓ On aurait aimé trouver $\cos\left(\frac{\pi}{3}\right) = \frac{\sqrt{3}}{2}$. *scilab* ne sait pas faire cela ; c'est du domaine des logiciels de calcul formel comme *Xcas*, qui contiennent une bibliothèque de formules, notamment les valeurs remarquables des fonctions usuelles.

- ✓ On remarque aussi que *scilab* ne donne pas la valeur 0 à $\cos\left(\frac{\pi}{2}\right)$ et donne une valeur à $\tan\left(\frac{\pi}{2}\right)$, parce qu'en fait, ce n'est pas le cosinus et la tangente de $\frac{\pi}{2}$ que *scilab* calcule, mais celles d'un angle voisin, cf. 3.4.
- ✓ *scilab* ne donne que des résultats approchés. La problème de la précision des calculs est un réel problème pour l'utilisateur ordinaire qui ne sait pas comment *scilab* fait ses calculs, cf. 3.4, et qui considère que c'est une boîte noire : on ne sait pas ce qui se passe dedans. Difficile de faire autrement.
- ✓ Enfin, pour calculer les sinus des éléments d'un tableau `A`, on voit qu'il suffit de la commande `sin(A)`. *scilab* retourne le tableau de mêmes dimensions que `A` dont les éléments sont les sinus des éléments correspondants de `A`. On pourra dire que la commande `sin` est vectorisée.
- ✓ Cette faculté de calculer les valeurs d'une fonction de la bibliothèque *scilab* par tableau, composante par composante, est extrêmement pratique.



3.3 [*] Conversion de la base 2 à la base 10

Soit n un entier $\geqslant 1$ donné sous sa forme $N = [a_0, a_1, \ldots, a_p]$ dans le système à base 2 : $a_0, \ldots, a_{p-1}$ sont chacun égaux à 0 ou à 1, $a_p = 1$ et p désigne un entier $\geqslant 0$ tels que

$$n = a_0 + a_1 \cdot 2 + \cdots + a_p \cdot 2^p$$

Problème : écrire n dans le système décimal.

Mots-clefs : commandes `input`, `size(N,'c')`, `length`, `ones`, `sum`, opérations élément par élément, fonction d'un tableau.

L'algorithme qui suit est un peu violent. Il montre comment *scilab* traite les tableaux de nombres et pourquoi il permet d'écrire des algorithmes très courts.

```
N=input('N=');
n=sum(N.*2.^(0:size(N,'c')-1));
afficher(n);
```

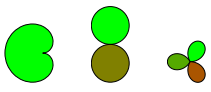
Dans ce script, on obtient `n` en multipliant le vecteur `N` par le vecteur `2.^[0,1,...,p]=[2^0,2^1,...,2^p]` composante par composante (le vecteur `(0:size(N,'c')-1)` est `[0,1,...,p]` ; il faut tenir compte de la priorité de `^` sur `*`). `size(N,'c')` est le nombre de colonnes de `N`.

Voici le même algorithme un peu détaillé et commenté. Nous avons remplacé `size(N,'c')` par `length(N)`.

```
N=input('N=');// Saisir N (liste des coefficients dyadiques).
P=0:(length(N)-1);// P=[0,1,...,p].
Q=2.^P;// Q=[2^0,2^1,...,2^p]
R=N.*Q;//R=[a0*2^0,a1*2^1,...,ap*2^p]
n=sum(R);// n=a0*2^0+a1*2^1+...+ap*2^p
afficher(P);
afficher(Q);
afficher(n);
```

Quand on charge ce script dans la console, *scilab* demande de saisir `N`. Par exemple, si (`N=[1,1,1,1,1,1,1]`), on obtient :

```
-->exec('Chemin du fichier', -1)
N=[1,1,1,1,1,1,1]
    0.    1.    2.    3.    4.    5.    6.
    1.    2.    4.    8.    16.   32.   64.
    127.
-->
```



3.4 [*] Les erreurs de *scilab*

1 - Calculer `.6*9+.6`.
2 - Calculer `floor(.6*9+.6)`. Comparer.

Mots-clefs : erreur, représentation des nombres réels, dépassement de capacité (overflow), erreur d'arrondi, commandes `clear`, `floor`, constantes `%eps`, `%nan` et `%inf`.

Tous les manuels consacrés à *scilab* exhibent des calculs faux ou impossibles. Nous en avons déjà rencontrés dans l'exercice 3.2. On sait bien que diviser 1 par 0 est impossible (par exemple). Ce cas peut se produire si le dénominateur d'un quotient non nul très proche de 0 est transformé en 0 par *scilab*. Ceci est dû à la représentation des nombres réels choisie par *scilab*. Ce problème s'impose de lui-même [1]. Dans une machine, les nombres réels sont représentés comme des nombres flottants [2] :

$$x = \pm 0.b_1 b_2 \dots b_t 10^e$$

La partie $0.b_1 b_2 \dots b_t$ s'appelle la mantisse et e l'exposant. Le nombre t s'appelle le nombre de chiffres significatifs. Scilab utilise l'arithmétique IEEE pour laquelle $-308 \leqslant e \leqslant 308$. Un calcul *bien programmé* ne peut conduire à des nombres en dehors de ces limites. Si cela arrive, il y a dépassement de capacité ou *overflow*. *À l'intérieur de ces limites, la machine arrondit tout résultat vers le plus proche nombre flottant.* La distance entre 1 et nombre flottant supérieur à 1 le plus proche s'appelle la précision de la machine ou ε-machine (`%eps`). Au voisinage de 1, l'erreur d'approximation du résultat d'un calcul par un nombre flottant est donc `%esp/2`.

```
-->clear
-->%eps
 %eps    =
     2.220D-16
-->1+0.5*%eps
 ans    =
     1.
-->1+0.5*%eps==1
 ans    =
  T
-->1+0.6*%eps
 ans    =
     1.
-->1+0.6*%eps==1
 ans    =
  F
-->
```

On voit que `1+0.5*%eps` vaut bien 1. Par contre, l'avant-dernière ligne [3] est troublante. On attendait évidemment `1+%0.6*%eps=1+%eps`.

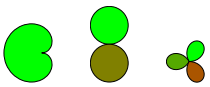
Retenons encore que dans l'arithmétique IEEE, le plus grand nombre > 0 possible est voisin de $1.797 \cdot 10^{308}$. Le plus petit nombre positif possible est de l'ordre de 10^{-324}. Se reporter à [11], *chapitre 15 - L'arithmétique de l'ordinateur* dont ces lignes sont extraites. Le lecteur aura aussi intérêt à prendre connaissance des

1. Il est clair que l'on ne peut pas saisir dans un ordinateur un nombre irrationnel, par exemple parce que celui-ci s'écrit avec une infinité de chiffres dans le système de numération en base 10.
2. nombres à virgule flottante
3. contredite heuresement par la dernière ligne

constantes `%nan` et `%inf`, cf. [11], pp. 67-68. On doit pouvoir les éviter en classe.
L'exercice suivant (cf. [11]) est également troublant :

```
-->.6*9+.6
 ans  =
        6.
-->.6*9+.6-6
 ans  = - 8.881784197D-16
-->floor(.6*9+.6)
 ans  =
        5.
-->
```

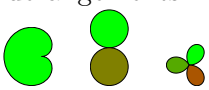
La première réponse est `6`. On s'attend que la partie entière de ce nombre soit 6. Mais *scilab* retourne 5 (dernière ligne). Cela s'explique : pour effectuer ce calcul élémentaire pour nous, *scilab* ré-écrit ces nombres en base 2, ce qui entraîne des arrondis et des erreurs d'arrondis. Dans ce cas, cela se passe assez mal. La deuxième réponse montre que `.6*9+.6` est strictement plus petit que 6 et que l'écart est plus grand que l'ε-machine. Il en résulte que la troisième réponse est fausse et que l'on ne peut pas contrer l'erreur (par exemple en considérant que des nombres dont la distance est inférieure à l'ε-machine sont en fait égaux). Cela n'est pas satisfaisant. *scilab* ne fait que ce qu'on lui à appris à faire lors de sa conception. Il ne fait pas d'erreur. Le problème pour les utilisateurs est qu'ils ne savent pas quels choix ont été faits. C'est très inconfortable.



Chapitre 4

Manipulations élémentaires

Liste des exercices :



4.1 [*] Échanger deux nombres

Soit $X = [a, b]$ une liste (un vecteur) de deux nombres. Donner un script qui retourne $[b, a]$.

Mots-clefs : commandes `X(B)`, `length`, `afficher`.

Solution n° 1 : Pour échanger deux nombres, on peut utiliser une variable auxiliaire, notée `c` ci-dessous :

```
a=input('a=');
b=input('b=');
X=[a,b];
afficher(X);
c=a;
X(1)=b;
X(2)=c;
afficher(X);
```

Ce script ayant été chargé dans la console, appliquons-le :

```
-->exec('Chemin du fichier', -1)
a=-3/2;
b=sqrt(2);
   - 1.5    1.4142135623731
   1.4142135623731   - 1.5
-->
```

La première ligne est la valeur originale de `X` ; la seconde sa valeur retournée.

Solution n° 2 : On peut éviter d'utiliser une variable supplémentaire. C'est l'objet du script suivant :

```
a=input('a=');
b=input('b=');
    a=a+b;
    b=a-b;
    a=a-b;
    X=[a,b];
    afficher(X);
```

Le tableau ci-dessous indique dans la première colonne la commande que l'on vient d'exécuter, dans la deuxième et la troisième colonnes le contenu des variables `a` et `b`.

	a	b
a=a+b	a+b	b
b=a-b	a+b	a
a=a-b	b	a

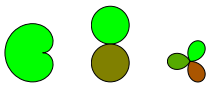
Solution n° 3 : Même si ce n'est pas évident sur un exemple aussi simple, il vaut mieux utiliser les commandes *scilab* parce qu'elles sont plus rapides (ça compte quand elles sont répétées un grand nombre de fois). Le script suivant est bien meilleur. Il utilise la commande `X(B)` que nous décrivons maintenant : `X` étant un vecteur de longueur n, `B` un vecteur d'entiers compris entre 1 et n, la commande `X(B)` retourne le vecteur des éléments de `X` dont les indices appartiennent à `B`. Par exemple (ici on a travaillé directement dans la console) :

```
-->X=[-2,1.5,175,-8,-.4];
-->n=length(X)
```

```
    n  =
        5.
-->B=[3,2,2,5]
     B  =
         3.      2.      2.      5.
-->X(B)
     ans  =
         175.      1.5      1.5   - 0.4
-->
```

Cette commande est très pratique.

```
a=input('a=');
b=input('b=');
X=[a,b];
afficher(X);
X=X([2,1]);
afficher(X);
```



4.2 [*] Échanger deux nombres dans une liste

Écrire un script qui, étant donné une liste de nombres $X = [x_1, \ldots, x_n]$ et deux indices i et j, retourne X après avoir échangé x_i et x_j.

Mots-clefs : commande `X(B)`, `input`, `afficher`.

Cet exercice contient l'exercice 4.1 comme cas particulier. La bonne solution est la seconde.

Rappel : `X` étant un vecteur de longueur n, `B` un vecteur d'entiers compris entre 1 et n, la commande `X(B)` retourne le vecteur dont les composantes sont les éléments de `X` dont les indices appartiennent à `B`.

Solution n° 1 : La solution la plus simple consiste à utiliser une variable auxiliaire `a`.

```
X=input('X=');
i=input('i=');
j=input('j=');
A=X;
a=A(i);
A(i)=A(j);
A(j)=a;
Y=A;
afficher(X);
afficher(Y)
```

On peut éviter d'introduire une variable auxiliaire comme dans la solution n°2 de 4.1.

On charge ce script dans la console de *scilab*. Traitons le cas où `X` est une liste de 10 nombres choisis au hasard entre 0 et 1, `i=3`, `j=7`. On constate en lisant `X` (première liste de nombres) et `Y` (deuxième liste de nombres) que les termes de rang 3 et 7 ont bien été échangés.

```
-->exec('Chemin du fichier', -1)
X=tirage_reel(10,0,1);
i=3;
j=7;
         column 1 to 3
    0.6019819397479    0.4719568016008    0.2629712824710
         column 4 to 6
    0.3402196990792    0.6540790954605    0.5298419883475
         column 7 to 9
    0.6892144982703    0.7161007095128    0.7481515908148
         column 10
    0.9883793974295

          column 1 to 3
    0.6019819397479    0.4719568016008    0.6892144982703
         column 4 to 6
    0.3402196990792    0.6540790954605    0.5298419883475
         column 7 to 9
    0.2629712824710    0.7161007095128    0.7481515908148
         column 10
    0.9883793974295
-->
```

Solution n° 2 : Voici une deuxième solution, plus compacte, plus élégante, plus rapide :

```
X=input('X=');
afficher(X);
i=input('i=');
j=input('j=');
X([i,j])=X([j,i])
afficher(X);
```

Chargeons ce script dans la console de *scilab* et appliquons-le.

```
-->exec('Chemin du fichier', -1)
X=tirage_entier(10,0,100);
          column 1 to 9
    54.    17.    94.    60.    62.    89.    63.    25.    41.
          column 10
    44.
 i=4
 j=9
          column 1 to 9
    54.    17.    94.    41.    62.    89.    63.    25.    60.
          column 10
    44.
-->
```

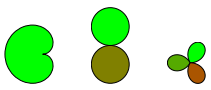
On constate que les termes de rang 4 et 9 ont bien été échangés.

Si on avait voulu supprimer les termes de rangs 3 et 5 de la liste, on aurait tapé, directement dans la console :

```
-->X([3,5])=[];X
```

et obtenir :

```
X  =
    54.    17.    41.    89.    63.    25.    60.    44.
-->
```



4.3 [*] Échanger deux colonnes ou deux lignes d'une matrice

Engendrer un tableau de nombres à n lignes et m colonnes à l'aide de la commande `grand(n,m,'uin',-13,14)` puis échanger les colonnes i et j (i et j donnés)

Mots-clefs : Commandes `input`, `grand(n,m,'uin',p,q)`, `X(B)` [1], `afficher`.

`grand(n,m,'uin',-13,14)`, plus généralement `grand(n,m,'uin',p,q)`, n'est pas une commande de *scilab pour les lycées* mais de *scilab* général. Il suffit de savoir qu'elle retourne ici un tableau de nombres entiers compris entre -13 et 14, bornes comprises, à n lignes et m colonnes (on ne peut pas se contenter des commandes `tirage_reel(n,a,b)` et `tirage_entier(n,a,b)` car ces commandes (de *scilab pour les lycées*) n'engendrent que des listes de nombres).

On peut utiliser le script suivant qui rappelle la solution n°2 de 4.2 :

```
n=input('n=');
m=input('m=');
X=grand(n,m,'uin',-13,14);
afficher(X);
i=input('i=');
j=input('j=');
X(:,[i,j])=X(:,[j,i]);
afficher(X);
```

dont voici une application :

```
-->exec('Chemin du fichier', -1)
n=7
m=9
  - 13.    10.     6.   - 3.     5.   - 11.  - 8.   - 13.  - 8.
    9.   - 9.   - 5.     11.  - 4.     11.  - 10.    11.  - 11.
  - 12.    9.   - 13.  - 6.   - 1.   - 3.     8.   - 11.  - 11.
    1.   - 6.     12.    11.  - 5.     10.  - 11.    2.     13.
  - 10.    2.     4.   - 12.    11.    12.    14.    9.     3.
  - 5.   - 13.  - 11.  - 6.   - 5.   - 2.     0.     1.   - 10.
  - 8.   - 3.   - 10.  - 1.     2.     10.  - 11.  - 5.     9.
i=3
j=4
  - 13.    10.  - 3.     6.     5.   - 11.  - 8.   - 13.  - 8.
    9.   - 9.     11.  - 5.   - 4.     11.  - 10.    11.  - 11.
  - 12.    9.   - 6.   - 13.  - 1.   - 3.     8.   - 11.  - 11.
    1.   - 6.     11.    12.  - 5.     10.  - 11.    2.     13.
  - 10.    2.   - 12.    4.     11.    12.    14.    9.     3.
  - 5.   - 13.  - 6.   - 11.  - 5.   - 2.     0.     1.   - 10.
  - 8.   - 3.   - 1.   - 10.    2.     10.  - 11.  - 5.     9.
-->
```

Les colonnes n°3 et n°4 de la matrice `X` à 7 lignes et 9 colonnes ont été échangées.

Pour échanger deux lignes, problème analogue, on pourra utiliser le script suivant :

1. Voir le rappel au début de l'exercice 4.2.

```
n=input('n=');
m=input('m=');
X=grand(n,m,'uin',-13,14);
afficher(X);
i=input('i=');
j=input('j=');
X([i,j],:)=X([j,i],:);
afficher(X);
```

dont voici une application :

```
-->exec('Chemin du script', -1)
n=7
m=9
  - 2.   - 5.   - 5.   - 12.  - 1.   - 1.    13.    6.     4.
    5.   - 2.     3.     8.   - 5.   - 2.    7.     0.     3.
  - 4.   - 12.  - 11.   14.    5.   - 9.    7.     4.   - 13.
    13.    3.   - 1.     14.    6.     7.    5.     5.   - 6.
    0.   - 13.  - 6.     3.   - 11.  - 13.  - 10.  - 13.  - 1.
  - 5.   - 7.   - 5.     5.   - 5.   - 12.  - 3.   - 6.     4.
    0.   - 4.     14.    1.     4.     11.  - 13.    11.    11.
i=3
j=4
  - 2.   - 5.   - 5.   - 12.  - 1.   - 1.    13.    6.     4.
    5.   - 2.     3.     8.   - 5.   - 2.    7.     0.     3.
    13.    3.   - 1.     14.    6.     7.    5.     5.   - 6.
  - 4.   - 12.  - 11.   14.    5.   - 9.    7.     4.   - 13.
    0.   - 13.  - 6.     3.   - 11.  - 13.  - 10.  - 13.  - 1.
  - 5.   - 7.   - 5.     5.   - 5.   - 12.  - 3.   - 6.     4.
    0.   - 4.     14.    1.     4.     11.  - 13.    11.    11.
-->
```

Les lignes n°3 et n°4 ont été échangées.

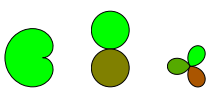
Si on voulait supprimer les lignes de rang impair de `X`, on taperait directement dans la console :

```
-->X(1:2:n,:)=[];X
```

ce qui donnerait :

```
X  =
    5.  - 2.    3.    8.   - 5.  - 2.    7.    0.    3.
  - 4.  - 12.  - 11.  14.    5.  - 9.    7.    4.  - 13.
  - 5.  - 7.   - 5.   5.   - 5.  - 12.  - 3.  - 6.    4.
-->
```

On a effacé les lignes numéros 1, 3, 5 et 7. Il ne reste que les lignes de rangs 2, 4 et 6.



4.4 [*] Ré-arrangement d'une suite de nombres (fantaisie)

Étant donné une liste de nombres $X = [x_1, \ldots, x_n]$, retourner la liste $[x_2, x_4, \ldots, x_1, x_3, \ldots]$ (suite des termes de rang pair de X suivis des termes de rang impair)

Mots-clefs : Commande `X(B)`, `length`, `input`, `x :y` et `x :p :y`.

Rappel : `X` étant un vecteur de longueur n, `B` un vecteur d'entiers compris entre 1 et n, la commande `X(B)` retourne le vecteur dont les composantes sont les éléments de `X` dont les indices appartiennent à `B`.

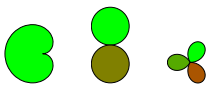
`x :y` retourne la liste des nombres de `x` à `y` par pas de 1 ; `x :p :y` retourne la liste des nombres de `x` à `y` par pas de `p`. On peut utiliser le joli script commenté suivant :

```
X=input('X=');// X est une liste de nombres.
n=length(X);// Longueur de la liste X.
B=2 :2 :n;// Liste des entiers pairs <=n à partir de 2.
C=1 :2 :n;// Liste des entiers pairs <=n à partir de 1.
Y=[X(B),X(C)];
afficher(X);
afficher(Y);
```

Le script ci-dessus, qui n'est pas vraiment élémentaire, ayant été chargé dans la console, en voici une application :

```
-->exec('Chemin du fichier', -1)
X=tirage_entier(7,-3,6)
    2.    5.    0.    2.    2.  - 3.    3.
    5.    2.  - 3.    2.    0.    2.    3.
-->
```

La première ligne est `X`, la seconde est `Y`. La commande `Y=[X(B),X(C)]` produit `Y` comme la liste des termes de rang pair suivie de la liste des termes de rang impair. Il n'y a pas à se soucier de la parité de `n`.



4.5 [*] Combien de temps l'exécution d'un script prend-elle ?

Engendrer une liste de 100 000 nombres ; inverser l'ordre de ces nombres et retourner le temps de calcul de cette inversion.

Mots-clefs : commande « `tic(); opération; toc()` », `tirage_reel(100000,0,1)`, `afficher`.

Dans l'énoncé, inverser une liste de nombres signifie l'écrire en commençant par son dernier terme, puis son avant-dernier, *etc.*

La séquence de commandes `tic(); opération; toc();` affiche le nombre de secondes nécessaires pour `opération` (voir l'aide en ligne). Nous allons engendrer la liste de nombres demandée à l'aide de la commande `tirage_reel(100000,0,1)` qui va fournir 100000 nombres réels entre 0 et 1, puis ranger ces nombres en partant de la fin.

Solution n° 1 : utilisons le joli script :

```
X=tirage_reel(100000,0,1);
tic();
Y=X(100000:-1:1);
temps=toc();
afficher('Les 5 derniers termes de X sont')
afficher(X(99996:100000));
afficher('Les 5 premiers termes de Y sont')
afficher(Y(1:5));
afficher('L"inversion a duré '+string(temps)+' secondes.')
```

Les commandes d'affichage sont ici compliquées et utilisent des manipulations de chaînes de caractères. On voit un peu comment ça marche ; sinon, voir l'aide en ligne. On aurait pu se contenter de :

```
afficher(X(99996:100000));
afficher(Y(1:5));
afficher(temps)
```

ou même se contenter de :

```
afficher(temps,Y(1:5),X(99996:100000));
```

Le temps qui est compté est le temps d'inversion de `X`, sans le temps qu'il faut pour engendrer `X` ni le temps d'affichage. On charge ce script dans la console. Il s'exécute automatiquement et retourne :

```
-->exec('Chemin du fichier', -1)
Les 5 derniers termes de X sont
       column 1 to 3
    0.0006993855350    0.7740111986641    0.5493441692088
       column 4 to 5
    0.3015413670801    0.7123542702757
Les 5 premiers termes de Y sont
       column 1 to 3
    0.7123542702757    0.3015413670801    0.5493441692088
       column 4 to 5
    0.7740111986641    0.0006993855350
L'inversion a duré 0.003 secondes
-->
```

On voit ici que *scilab* est très rapide car on vient de manipuler 100 000 nombres !

Solution n° 2 : utilisons maintenant un script maladroit, voir l'exercice 6.18 :

```
X=tirage_reel(100000,0,1);
tic();
Y=[X(100000)]
for i=2:100000
    Y=[Y,X(100000-i+1)];
end
temps=toc();
afficher('Les 5 derniers termes de X sont')
afficher(X(99996:100000));
afficher('Les 5 premiers termes de Y sont')
afficher(Y(1:5));
afficher('L''inversion a duré '+string(temps)+' secondes')
```

Chargeons ce script dans la console. Il s'exécute automatiquement. On obtient :

```
-->exec('Chemin du fichier', -1)
 Les 5 derniers termes de X sont
       column 1 to 3
    0.1685154619627    0.7311467428226    0.7330885585397
       column 4 to 5
    0.6812806068920    0.4993714094162
       Les 5 premiers termes de Y sont
       column 1 to 3
    0.4993714094162    0.6812806068920    0.7330885585397
       column 4 to 5
    0.7311467428226    0.1685154619627
L'inversion a duré 27.949 secondes
-->
```

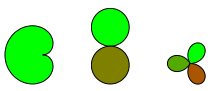
La lenteur du deuxième script est due à la boucle `pour`. Il y a peut-être une boucle `pour` dans le premier script, cachée dans une commande *scilab* de niveau plus élevé. Si c'est le cas (?), elle a été programmée dans un langage de programmation plus efficace. On peut en tirer diverses conclusions :

• il y a souvent plusieurs algorithmes solutions d'un problème donné, on peut mesurer leur vitesse d'exécution à la milliseconde près avec les commandes `tic()` et `toc()` comme on vient de le voir ;

• c'est un jeu de chercher le script le plus rapide et éventuellement de pouvoir le comparer aux algorithmes *scilab*, qui a une bibliothèque d'algorithmes très riche ;

• il est gratifiant d'écrire un algorithme avec des commandes élémentaires uniquement, mais c'est sans espoir concernant la vitesse d'exécution, l'occupation d'espace-mémoire, *etc* ; toutes les fois que c'est possible, il faut remplacer les séquences de commandes qui s'y prêtent par des commandes évoluées ;

• enfin, il est bon d'utiliser au mieux les capacités de calcul de *scilab* directement sur des tableaux.



4.6 [***] Sur la définition du produit matriciel

Soit $A = (a_{i,j})$ et $B = (b_{i,j})$ deux matrices, A ayant n lignes et m colonnes, B ayant m lignes et p colonnes (le nombre de colonnes de A est égal au nombre de lignes de B). Alors $C = A \cdot B$ (commande *scilab* : `C=A*B`) est la matrice à n lignes et p colonnes définies par

$$c_{i,j} = \sum_{k=1,\ldots,m} a_{i,k} \cdot b_{k,j}$$

1 - Soit `A=tirage_entier(0,10,7)` et `B=tirage_entier(-14,27,7)` deux listes de 7 nombres. Peut-on effectuer le produit `A*B` de ces matrices ?

2 - Calculer `B'`. Décrire cette matrice.

3 - Peut-on effectuer le produit `B*A` ? `B*A'` ? Décrire `B*A'`.

4 - Soit `C=grand(7,7,"uin",-2,8)` une matrice à 7 lignes et 7 colonnes (matrice carrée). Décrire les matrices `A*C` et `C*A'`.

Mots-clefs : produit matriciel, transposition de matrices (commande `'`).

Il suffit de savoir que `A=tirage_entier(0,10,7)` et `B=tirage_entier(-14,27,7)` sont deux listes de 7 entiers compris entre 0 et 10 pour la première, entre -14 et 27 pour la seconde.

1 - Non, car le nombre de colonnes de `A` n'est pas égal au nombre de lignes de `B`. Voilà ce que retourne *scilab* (tapé directement dans la console) :

```
-->A=tirage_entier(7,0,10)
 A  =
    6.    3.    10.    9.    4.    7.    0.
-->B=tirage_entier(7,-14,27)
 B  =
   - 9.  - 14.    11.  - 1.    3.    4.  - 14.
-->A*B
    !--error 10
Multiplication incohérente.
-->
```

2 - `B'` s'obtient en écrivant `B` en colonne. `B'` s'appelle la transposée de `B`. En tapant `B'` dans la console puis `Entree`, on obtient en effet :

```
-->B'
 ans  =
  - 9.
  - 14.
    11.
  - 1.
    3.
    4.
  - 14.
-->
```

3 - Le produit matriciel `B*A` est impossible. `B*A'` est la somme des produits des composantes des rangs correspondants de `B` et `A`. Si on avait choisi au départ des listes de 3 nombres (au lieu de 7), on aurait obtenu le produit scalaire de la géométrie euclienne.

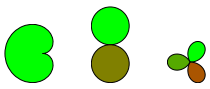
```
-->A*B'
```

```
 ans  =
    45.
-->
```

4 - Pour `A*C`, voir ci-dessous. Décrire les matrices `A*C` et `C*A'` est difficile. En pensant à la question 3, on peut comprendre que `A*C` est la liste des produits scalaires de `A` par les colonnes successives de `C`. De même, `C*A'` est la colonne des produits scalaires des lignes successives de `C` par la colonne A'.

```
-->C=grand(7,7,"uin",-2,8)
 C  =
    - 2.    3.  - 1.    5.    2.  - 2.    0.
    3.    5.    4.    6.    8.    3.    6.
    0.    6.    4.    5.    3.    0.    0.
    1.    4.  - 2.  - 1.    6.    7.  - 1.
    0.    1.    0.    2.    2.    1.    4.
    5.  - 1.  - 1.    7.  - 1.    4.    3.
    4.    5.  - 1.    2.    6.    6.    0.
 -->A*C
 ans  =
       column 1 to 6
    41.    126.    21.    146.    121.    92.
       column 7
    46.
-->C*A'
 ans  =
     26.
    180.
    115.
    62.
    36.
    104.
    113.
-->
```

Bien sûr, toutes les affirmations ci-dessus se démontrent facilement en utilisant la définition du produit matriciel rappelée dans l'énoncé.



4.7 [***] Calcul de nombres de dérangements

Le nombre de dérangements (ou permutations sans point fixe) de la liste $(1, \ldots, n)$ est donné par la formule

$$N_n = (1 \times \cdots \times n)\left(1 + \sum_{k=1}^{n} \frac{(-1)^k}{1 \times \cdots \times k}\right) \quad \text{ou} \quad N_n = n!\left(1 + \sum_{k=1}^{n} \frac{(-1)^k}{k!}\right)$$

1 - Écrire un script qui, étant donné l'entier n, retourne N_n.

2 - Quelle est la plus grande valeur de n qui permet à *scilab* d'afficher N_n exactement (c'est à dire sans passer en notations flottantes) ? Combien N_n vaut-il alors ?

Mots-clefs : commandes `input`, `format`, `cumprod`, `sum`, `$`, puissance composante par composante (`.^`), division composante par composante (`./`).

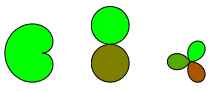
1 - Ceci est un pur exercice de calcul. Il est inutile de savoir ce que « dérangements » veut dire. Dans le script qui suit, `A=1:n` produit la liste ou vecteur $(1, \ldots, n)$, `B=(-1).^A` retourne la liste ou vecteur $\left((-1)^1, \ldots, (-1)^n\right)$ à savoir -1 élevé à la puissance `A` composante par composante, qui a la même longueur que `A`, `C=cumprod(A)` donne le vecteur $(1!, \ldots, n!)$ (premier terme de `A`, produit des 2 premiers termes de `A`, …, produit des n premiers termes de `A`). Cette commande évite d'utiliser des boucles `pour`. `C($)` est le dernier terme de `C`, soit $n!$ et `B./C` est le vecteur des quotients des éléments de `B` par les éléments de `C` de même rang. Cela donne finalement un script compact inutilement compliqué car la suite $(N_n, n \geqslant 1)$ suit une relation de récurrence qui permet de faire ces calculs plus simplement et plus rapidement.

```
n=input('n= ');
format('v',25);// Capacité maximum d'affichage.
A=1:n;
B=(-1).^A;// -1, 1, ... , (-1)^n.
C=cumprod(A);// 1, 1*2, ..., 1* ... * n
N=C($)*(1+sum(B./C));// C($) est le dernier terme de C.
afficher(N);
```

2 - On trouve la plus grande valeur de n qui permet un affichage exact de N_n par tâtonnement. On obtient plus précisément :

```
-->exec('Chemin du script', -1)
n= 23
    9510425471055776710656.
-->exec('Chemin du script', -1)
n= 24
    2.282502113053385907D+23
-->
```

C'était fatal car *scilab* peut afficher exactement $23! = 1 \times \cdots \times 23$ mais passe en notations flottantes pour afficher $24! = 1 \times \cdots \times 24$.



Chapitre 5

Programmer une fonction scilab

La syntaxe de la définiton d'une fonction *scilab* est :

```
function <arguments_sortie>=<nom_de_la_fonction><arguments_entrée>
<instructions>
endfunction
```

(voir l'aide en ligne ou un manuel).

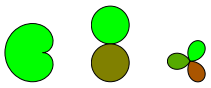
Liste des exercices :

Énoncé n° 5.1 [*] : Volume du cône.

Énoncé n° 5.2 [*] : Aire d'un trapèze.

Énoncé n° 5.3 [**] : Moyenne et variance empiriques.

Énoncé n° 5.4 [**] : Conversion de temps.



5.1 [*] Volume du cône

1 - Fabriquer une fonction-scilab qui calcule le volume v d'un cône de révolution de hauteur h et dont la base est un disque de rayon r.

2 - Cette fonction accepte-t-elle que r et h soient des listes de nombres (ce qui serait utile si on voulait faire varier le rayon et la hauteur) ?

Mots-clefs : fonction *scilab* ; à la fin : calcul matriciel (produit, transposition d'une matrice).

On admet que la formule qui donne ce volume est $v = \dfrac{\pi r^2 h}{3}$.

Solution 1 : Nous pouvons faire de cette formule une fonction-scilab dans *SciNotes*. Ce sera une fonction avec 2 entrées `h` et `r` et une seule sortie `v`, appelée `volcone` ci-dessous. Cette solution est élémentaire.

```
function v=volcone(h,r);// Le nom de cette fonction est volcone.
  v=%pi*r^2*h/3;
endfunction ; //Fin de la definition de volcone
```

Les commentaires (sur une ligne, précédés de `//`) sont souvent utiles mais n'ont aucune action sur l'exécution de la fonction. Aussi, on peut se contenter de :

```
function v=volcone(h,r);
 v=%pi*h*r^2/3;
endfunction
```

Remarques :

• Quand un script est long ou compliqué, les commentaires sont presque toujours indispensables pour le comprendre.

• Enregistrons ce script (fichier de commandes) sous le nom `volcone.sci`. L'extension `sci` rappelle qu'il s'agit d'une fonction *scilab*. L'extension `sce` n'aurait pas posé de problème. C'est un simple procédé mnémotechnique pour distinguer les fonctions *scilab* des scripts ordinaires.

• Pour utiliser cette fonction, il faut la charger dans la console de *scilab*, ce qui se fait avec l'option `Fichier sans écho` du menu `Exécuter` de la console.

• Si on avait choisi `Fichier avec écho`, le script aurait été recopié sur la console de *scilab*, ce qui aurait nui à la clarté de l'écran. Cette option est donc à éviter.

• Les variables `h` et `r` qui figurent dans le script de la fonction `volcone` sont ignorées quand celui-ci est chargé dans la console. Cela veut dire que l'on peut nommer `h` ou `r` d'autres variables, sans inconvénient.

• On ne voit pas bien l'intérêt de définir une fonction comme `volcone` sauf peut-être si ce calcul est répété un grand nombre de fois.

Utiliser `volcone` :

On charge `volcone` dans la console, ce qui donne :

```
-->exec('Chemin du fichier', -1)
-->
```

Ensuite, `volcone` s'utilise comme une fonction *scilab* ordinaire. Comme nous l'avons construite nous-mêmes, nous savons exactement quel est son domaine d'application :

• **Cas 1.** `h` et `r` sont des nombres ($\geqslant 0$). Par exemple, calculons le volume d'un cône de hauteur 3 et de rayon $\sqrt{2}$. Il suffit de taper la commande `vol=volcone(3,sqrt(2))`, si on appelle `vol` le volume à calculer, ce qui retourne, après un retour-chariot :

```
-->exec('Chemin du fichier', -1)
-->vol=volcone(3,sqrt(2))
 vol  =
     6.2831853071796
-->
```

• **Cas 2.** `r` est un nombre et `h` une liste de nombres. Par exemple, si l'on veut calculer les volumes de cônes de rayon $\sqrt{2}$ et de hauteurs variant de 0 à 5 avec un pas de 0.5, on tapera les commandes :

```
-->r=sqrt(2);
-->h=0:0.5:5;
-->volcone(h,r)
```

qui retourneront

```
 ans  =
        column 1 to 3
    0.     1.0471975511966     2.0943951023932
        column 4 to 5
    3.1415926535898     4.1887902047864
        column 6 to 7
    5.235987755983     6.2831853071796
        column 8 to 9
    7.3303828583762     8.3775804095728
        column 10 to 11
    9.4247779607694     10.471975511966
-->
```

(`ans` parce que l'on a omis de donner un nom aux quantités à calculer). Ça marche parce que la commande `v=%pi*h*r^2/3` est exécutable par *scilab* (pour multiplier une matrice, ici `h` par un nombre, ici `r^2`, on multiplie tous les éléments de la matrice par ce nombre ; même chose pour la division de `%pi*h*r^2` par 3).

• **Cas 3.** `h` est un nombre et `r` une liste de nombres : ça ne marche pas parce que la commande `r^2` n'est pas exécutable (produit de matrices impossible, sauf si `r` est un nombre, c'est à dire une matrice à un élément !

• **Cas 4.** A fortiori, `h` et `r` ne peuvent pas être des listes de plus d'un nombre (car apparaît un produit de 2 matrices impossible).

Solution 2 : Pourtant, tous les cas fonctionnent si on modifie légèment la définition de la fonction `volcone`, rebaptisée `volconebis`, comme suit :

```
function v=volconebis(h,r);//r est une matrice-colonne, h une matrice-ligne.
 v=%pi*r.^2*h/3;
endfunction
```

`r.^2` signifie que la matrice `r` est élevée au carré composante par composante. Si on veut faire varier `r` de 0 à 2 avec un pas de 0.1 et `h` comme dans le cas 2, on tapera les commandes :

```
function v=volconebis(h,r);//h et r sont 2 matrices-lignes.
 v=%pi*r'.^2*h/3;
endfunction
format('v',5);
r=0:0.1:1;//matrice-ligne de longueur 11.
h=0:0.5:4;//matrice-ligne de longueur 9.
volumes=volconebis(h,r)
afficher(volumes)
```

Remarques :

• `volconebis` fait tout ce que faisait `volcone` car les nombres sont des matrices à une ligne et une colonne.

• Ce script a été écrit entièrement dans *SciNotes* (c'est une variante) et chargé dans la console. Il s'exécute alors automatiquement.

• Le nombre de décimales a été limité à 2 avec la commande `format` pour raccourcir l'affichage.

• `r'` est matrice transposée de `r` : c'est une matrice-colonne.

• Il en résulte que la matrice `r'.^2` est composée des carrés des éléments de `r` écrits en colonne ; les règles de multiplication des matrices, pour `r'.^2` et `h`, sont bien respectées.

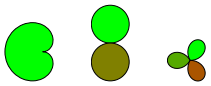
Ce script retourne :

```
-->exec('Chemin du fichier charg\'{e} ',-1)
    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.01  0.01  0.02  0.02  0.03  0.03  0.04  0.04
    0.    0.02  0.04  0.06  0.08  0.10  0.13  0.15  0.17
    0.    0.05  0.09  0.14  0.19  0.24  0.28  0.33  0.38
    0.    0.08  0.17  0.25  0.34  0.42  0.50  0.59  0.67
    0.    0.13  0.26  0.39  0.52  0.65  0.79  0.92  1.05
    0.    0.19  0.38  0.57  0.75  0.94  1.13  1.32  1.51
    0.    0.26  0.51  0.77  1.03  1.28  1.54  1.8   2.05
    0.    0.34  0.67  1.01  1.34  1.68  2.01  2.35  2.68
    0.    0.42  0.85  1.27  1.7   2.12  2.54  2.97  3.39
    0.    0.52  1.05  1.57  2.09  2.62  3.14  3.67  4.19
-->
```

Le script suivant :

```
format('v',5);
r=0:0.1:1;
h=0:0.5:4;
volumes=%pi*r'.^2*h/3;
afficher(volumes)
```

a exactement le même rôle que le précédent. Il se réduit à un simple calcul matriciel.



5.2 [*] Aire d'un trapèze

Fabriquer une fonction-scilab qui donne l'aire d'un trapèze de bases b et B et de hauteur h.

Mots-clefs : somme de matrices terme à terme, transposition, produit matriciel.

Solution n° 1 : Nous avons appelé cette fonction `airtrap`. Elle donnera l'aire d'un trapèze de bases `b` et `B` et de hauteur `h` (3 variables). Elle retourne une seule variable : l'aire.

```
function olala=airtrap(b,B,h);
olala=(b+B)*h/2;
endfunction
```

Chargeons cette fonction dans la console et calculons, par exemple, l'aire d'un trapèze de bases 4 et 5 et de hauteur 3 :

```
-->airedutrapeze=airtrap(4,5,3)
 airedutrapeze  =
    13.5
-->
```

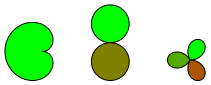
Solution n° 2 : Comme dans l'exercice 5.1, on peut avoir envie de donner à `b` et `B` n valeurs, à `h` p valeurs et de calculer les aires de ces trapèzes. Compte tenu de la définition du produit des matrices, il suffit de modifier comme suit la programmation de la fonction `airtrap` :

```
function olala=airtrapbis(b,B,h);
olala=(b+B)'*h/2;
endfunction
```

(on a simplement transposé la matrice `(b+B)`). Quand on exécute cette fonction, on obtient un tableau à n lignes et à p colonnes qui sont les aires recherchées. Sur les lignes de ce tableau, on lit les aires pour `b` et `B` fixés, `h` variant. Sur les colonnes, c'est `h` qui est fixé . Par exemple :

```
-->h=[1,2.4,7];b=1:0.3:4.1;B=1:11;airesdestrapeze=airtrapbis(b,B,h)
 airesdestrapeze  =
    1.      2.4     7.
    1.65    3.96    11.55
    2.3     5.52    16.1
    2.95    7.08    20.65
    3.6     8.64    25.2
    4.25    10.2    29.75
    4.9     11.76   34.3
    5.55    13.32   38.85
    6.2     14.88   43.4
    6.85    16.44   47.95
    7.5     18.     52.5
-->
```

Remarquer que l'on a saisi plusieurs commandes sur une même ligne, séparées par des `;`. La fonction `airtrap` est devenue inutile car on peut donner à `b`, `B` et `h` une seule valeur et appliquer la fonction `airtrapbis`.



5.3 [*] Moyenne et variance empiriques

1 - Calculer la moyenne et la variance du vecteur
$$[\sqrt{3}, -2.7, 4^2, \log(7), \exp(0.8), -4.2, \pi, 17, 13/11, \cos(4), 0]$$
.
2 - Engendrer une liste de `n=50` nombres entiers entre les bornes `a=-7` et `b=39` à l'aide de la commande `tirage_entier(n,a,b)`, puis calculer sa moyenne et sa variance.

Mots-clefs : `tirage_entier`, `moyenne = mean`, `variance = variance`, `afficher = disp`.

1 - Le script ci-dessous, sans intérêt, montre une fonction *scilab* notée `MV` à une entrée, qui est un vecteur numérique[1] de longueur 11 et deux sorties, qui sont des nombres[2]. Nous avons rédigé la fonction et son application, qui est la solution de la première question, sur un même script dans l'éditeur de texte *Scinote*.

```
function [m,v]=MV(Liste)
    m=mean(Liste);
    v=variance(Liste);
endfunction
L=[sqrt(3),-2.7,4^2,log(7),exp(0.8),-4.2,%pi,17,13/11,cos(4),0];
[mo,va]=MV(L);
disp('La moyenne et la variance de la liste de nombres donnee sont
respectivement egales a');
disp(mo);
disp(va);
```

Voici ce qui apparaît dans la console de *scilab* :

```
--> exec('Chemin du script', -1)
 La moyenne et la variance de la liste de nombres donnee sont
 respectivement egales a
    3.2430245
    47.767386
```

2 - Cette question requiert une fonction notée `mV` à trois entrées : `n`, `a` et `b` et trois sorties : la liste des nombres, sa moyenne et sa variance. Les fonctions utilisées `tirage_entier`, `moyenne`, `variance` sont des fonctions usuelles de *scilab*, voir [12]. Noter que `[x,mo,va]` est un vecteur numérique de longueur `n+2`.

```
function [x,mo,va]=mV(n,a,b)
    x=tirage_entier(n,a,b);
    mo=moyenne(x);
    va=variance(x);
endfunction
//Application
n=50;
a=-7;
b=39;
[x,mo,va]=mV(n,a,b);// Vecteur de longueur n+2, soit 52.
afficher('L''esperance et la variance de ');
afficher(x);
```

1. une liste de nombres
2. les commandes `moyenne`, `variance` et `afficher` ont dû être remplacées par les commandes en anglais correspondantes `mean`, `variance` et `disp` à cause des retards habituels de *scilab* quand l'OS des Mac change.

```
afficher('sont respectivement egales a');
afficher(mo);
afficher(va);
-->
```

On obtient dans la console :

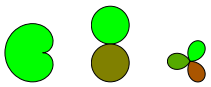
```
-->exec('Chemin du script', -1)
 L'esperance et la variance de
         column  1 to 13
    0.    21.    30.    30.    30.  - 3.    16.    25.  - 2.    39.    32.
33.    35.
         column 14 to 27
    16.    31.    39.    37.    9.    22.  - 2.    37.    1.    1.    1.    23.
- 2.    6.
         column 28 to 40
  - 6.    35.    39.    29.    39.    25.    37.    36.    10.    1.    36.
13.  - 3.
         column 41 to 50
  - 2.    18.    13.    31.    4.    35.    38.    1.    37.    9.
 sont respectivement egales a
    19.6
    242.97959183673
 -->
```

ce qui indique que la moyenne de `x` est approximativement 19.6, sa variance approximativement 242.98.

On aurait pu tout aussi bien choisir $n = 500000$. Alors, *scilab* aurait entrepris d'afficher 500 000 nombres (affichage de `x`), ce qui est sans intérêt, long et parfois impossible pour des questions de taille de mémoire. On peut interrompre le processus (menu déroulant « Contrôle »). Même pour une liste de 500 000 nombres, le calcul de la moyenne et de la variance sont pratiquement instantanés.

Remarques : Interprétation de la deuxième question en terme de Calcul des probabilités

Soit `X` une variable aléatoire prenant les valeurs -7, ..., 39 avec la même probabilité $\frac{1}{47}$ (loi uniforme sur l'intervalle d'entiers $[-7, \ldots, 39]$)[3]. `x` peut être considéré comme un échantillon de taille 500 000 de la loi de `X`[4]. Alors `mo` et `va` sont respectivement la moyenne et la variance empiriques calculées à partir de ces 500 000 tirages. Dans l'exercice 6.22, on refait la deuxième question dans un cadre plus général en n'imposant pas que la loi de `X` soit une loi uniforme, c'est à dire en ne supposant plus que toutes les issues possibles ayant une probabilité > 0 soient équiprobables.



3. On pourrait avec autant de raison supposer que `X` prend les valeurs -10, ..., 50 avec la même probabilité $\frac{1}{61}$!

4. ce que l'on obtient quand on tire au hasard 500 000 fois un nombre entiers compris au sens large entre -7 et 39, les tirages ètant indépendants les uns des autres

5.4 [**] Convertion de temps

Programmer une fonction-*scilab* qui, étant donné une durée d en secondes, la retourne sous forme d'années, mois, jours, heures, minutes et secondes.

Mots-clefs : division euclidienne, commandes `floor`, `pmodulo` (quotient et reste de la division euclidienne), `disp` (`afficher`)[5], facultativement concaténation de chaînes de caractères.

Est donc demandée une fonction à une entrée numérique et 6 sorties (toutes numériques). C'est un bel exercice sur la division euclidienne trouvé dans [13]. En effet, si on divise d par 60, le reste sera le nombre de secondes restantes tandis que le quotient sera le nombre de minutes contenues dans d, et ainsi de suite. Cela donne la fonction-*scilab* suivante :

Listing 5.1 – fonction Durees

```
//La fonction-scilab ci-dessous transforme une duree exprimee en secondes en
//annees, mois, jours, heures, minutes et secondes
//(toutes les annees ont 365 jours, les mois 30 jours).
// n est un  entier >=0.
function [a,m,j,h,mn,s]=Durees(d)
    s=pmodulo(d,60);
    Mn=floor(d/60);
    mn=pmodulo(Mn,60);
    H=floor(Mn/60);
    h=pmodulo(H,24);
    J=floor(H/24);
    j=pmodulo(J,30);
    M=floor(J/30);
    m=pmodulo(M,12);
    a=floor(M/12);
    disp([a,m,j,h,mn,s]);
endfunction
```

Si, par exemple, on veut savoir combien font 1234567890 secondes, on chargera la fonction `Durees` dans la console puis on tapera la commande *ad hac* :

```
-->clear
-->exec('Chemin de la fonction Durees', -1)
-->D=Durees(1234567890);
    39.    8.    8.    23.    31.    30.
-->
```

L'affichage du résultat étant un peu rustique, on pourra préférer le script suivant :

Listing 5.2 – fonction Durees

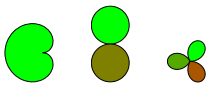
```
function [a,m,j,h,mn,s]=Durees(d)
    s=pmodulo(d,60);
    Mn=floor(d/60);
    mn=pmodulo(Mn,60);
    H=floor(Mn/60);
```

5. Ces commandes en anglais sont utilisées parce que, comme souvent, les commandes en français ne sont pas disponibles, après un changement d'OS. *scilab* prend son temps, beaucoup de temps.

```
    h=pmodulo(H,24);
    J=floor(H/24);
    j=pmodulo(J,30);
    M=floor(J/30);
    m=pmodulo(M,12);
    a=floor(M/12);
    disp([a,m,j,h,mn,s]);
    disp(string(d)+" secondes font "+string(a)+" annees "+string(m)+" mois "
    +string(j)+" jours "+string(h)+" heures "+string(mn)+" minutes "+string(s)
    +" secondes.")
endfunction
```

Les instructions d'affichage sont devenues compliquées. On affiche une chaîne de caractères obtenue en concaténant (opérateur +) 14 chaînes de caractères. Cela donne :

```
-->clear
-->exec('Chemin de la fonction Durees', -1)
-->D=Durees(1234567890);
 1234567890 secondes font 39 annees 8 mois 8 jours 23 heures 31 minutes
 30 secondes.
-->
```



Chapitre 6

Boucles `pour`, graphes simples, matrices

Syntaxe de la boucle « pour » : `for x=vecteur, instruction; end`

L'instruction est exécutée, `x` prenant successivement toutes les valeurs du vecteur.

Tracés élémentaires : commande « plot »

- ✓ Pour tracer le graphe d'une fonction, *scilab* calcule les coordonnées d'un certain nombre de points de ce graphe et relie les points consécutifs par un segment de droite.
- ✓ Les abscisses de ces points sont collectées dans le vecteur-ligne `X`, les ordonnées correspondantes dans le vecteur-ligne `Y`, la commande `plot(X,Y)` retourne alors le graphe.
- ✓ Très souvent, `Y` est calculé à l'aide d'une boucle `pour`, c'est pourquoi il y a des graphes très élémentaires dans ce chapitre.
- ✓ Les matrices apparaissent dans ce chapitre car elles permettent souvent d'économiser des boucles `pour`, de raccourcir les scripts et d'accélérer les calculs. Ce sont parfois de simples tableaux de nombres qui ne demandent aucune connaissance particulière. Cela change dès que l'on multiplie des matrices (en gros). La multiplication des matrices est en relation étroite avec la composition des applications linéaires et les changements de base dans les espaces vectoriels.

Les exercices 6.19, 6.20 et 6.21 dépassent le niveau du bac.

Liste des exercices :

Énoncé n° 6.14 : [**] Construction de matrices : matrice barrée (fantaisie).

Énoncé n° 6.15 : [**] Aiguilles de l'horloge.

Énoncé n° 6.16 : [**] Calculs d'effectifs et de fréquences.

Énoncé n° 6.17 : [**] Écrire à la fin le premier terme d'une liste de nombres.

Énoncé n° 6.18 : [*] Écrire une suite de nombres à l'envers.

Énoncé n° 6.19 : [***] Écrire toutes les permutations de 1 à n (solution matricielle).

Énoncé n° 6.20 : [***] Matrices de permutation.

Énoncé n° 6.21 : [***] Échanger deux lignes ou deux colonnes d'un tableau (produit d'une matrice par une matrice de permutation).

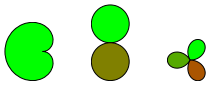
Énoncé n° 6.22 : [**] Calcul matriciel de l'espérance et de la variance d'une variable aléatoire finie.

Énoncé n° 6.23 : [***] Calcul matriciel de la covariance de 2 variables aléatoires finies.

Énoncé n° 6.24 : [***] Ranger n nombres donnés dans l'ordre croissant (tri à bulles).

Énoncé n° 6.25 : [****] Ranger n nombres donnés dans l'ordre croissant.

Énoncé n° 6.26 : [**] Trier un tableau suivant l'une de ses lignes.



6.1 [*] Tables de multiplication

Afficher les tables de multiplication par 1, ..., 9.

Mots-clefs : matrice-colonne (dans la première solution), multiplication composante par composante, commandes `ones`, `disp` (`afficher`), `+` (concaténation de chaînes de caractères), boucle `pour`.

Cet exercice est avant tout un problème d'affichage. Voici un premier script convenable :

```
// Ecrire les tables de multiplication par 1, ..., 9.
A=[0;1;2;3;4;5;6;7;8;9];
for j=1:9
    B=j*ones(10,1);
    disp(string(A)+'x'+string(B)+'='+string(B)+'x'+string(A)+'='+string(A.*B));
    disp('----------')
end
```

Commentaires

- `A` et `B` sont des colonnes à 10 lignes.
- `'x'` est la chaîne réduite à la lettre `x` utilisée comme signe de la multiplication.
- On obtient la table de multiplication par `j` en multipliant la colonne `A` terme à terme par la colonne `B` (commande `A.*B`).

Exécutons ce script. Pour gagner de la place, nous ne faisons apparaître ci-dessous que les tables de multiplication par 1 et par 9.

```
-->exec('Chemin du script', -1)
!0 x 1 = 1 x 0 = 0   !
!                    !
!1 x 1 = 1 x 1 = 1   !
!                    !
!2 x 1 = 1 x 2 = 2   !
!                    !
!3 x 1 = 1 x 3 = 3   !
!                    !
!4 x 1 = 1 x 4 = 4   !
!                    !
!5 x 1 = 1 x 5 = 5   !
!                    !
!6 x 1 = 1 x 6 = 6   !
!                    !
!7 x 1 = 1 x 7 = 7   !
!                    !
!8 x 1 = 1 x 8 = 8   !
!                    !
!9 x 1 = 1 x 9 = 9   !
 ----------
!0 x 9 = 9 x 0 = 0    !
!                     !
!1 x 9 = 9 x 1 = 9    !
!                     !
!2 x 9 = 9 x 2 = 18   !
```

```
!                   !
!3 x9=9x3=27        !
!                   !
!4 x9=9x4=36        !
!                   !
!5 x9=9x5=45        !
!                   !
!6 x9=9x6=54        !
!                   !
!7 x9=9x7=63        !
!                   !
!8 x9=9x8=72        !
!                   !
!9 x9=9x9=81        !
 ---------------
 -->
```

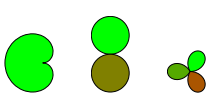
L'affichage obtenu n'est pas agréable. Le script qui suit est meilleur de ce point de vue. Il comprend une boucle `pour` insérée dans un boucle `pour`. Il est très simple.

```
// Ecrire les tables de multiplication par 1, ..., 9.
for i=1:9
    for j=0:9
        disp(string(j)+'x'+string(i)+'='+string(i)+'x'+string(j)
        +'='+string(i*j))
    end
    disp('---------------')
end
```

Ci-dessous, nous ne reproduisons que la table de multiplication par 9 :

```
-->exec('Chemin du script', -1)
---------------
 0x9=9x0=0
 1x9=9x1=9
 2x9=9x2=18
 3x9=9x3=27
 4x9=9x4=36
 5x9=9x5=45
 6x9=9x6=54
 7x9=9x7=63
 8x9=9x8=72
 9x9=9x9=81
 ---------------
 -->
```

Cet affichage est satisfaisant.



6.2 [*] Minimum et maximum d'une liste de nombres

Une liste de nombres étant donnée, calculer son minimum et son maximum.

Mots-clefs : boucle `pour`, `taille`, `min`, `max`.

Calcul du minimum

Le procédé de calcul du minimum utilisé ici est très simple : on examine un par un les termes de la liste (ou vecteur) donnée. `m` désigne son minimum. Après examen du premier terme, on égale `m` à ce terme. Après examen du second terme, on ne change pas la valeur de `m` si ce second terme est supérieur ou égal au premier ; sinon, on la remplace par ce second terme, et ainsi de suite. Après examen du dernier terme de la liste, `m` est bien son minimum. Ce procédé est traduit exactement dans le script suivant qui se présente comme une fonction *scilab* ayant pour entrée une liste de nombres et pour sortie son minimum :

```
function m=Min(A)// A est une liste de nombres ; m sera son minimum.
    m=A(1);
    t=taille(A);
    for i=1:t-1
        if m>A(i+1) then
            m=A(i+1);
        end
    end
endfunction
```

Cette fonction étant chargée dans la console, considérons la liste de nombres définie par la commande `A=tirage_reel(10000,-2,5)` (`A` est une liste de 10000 nombres entre -2 et 5) et calculons son minimum :

```
-->exec('Chemin du fichier charge', -1)
-->A=tirage_reel(10000,-2,5);
-->m=Min(A);
-->m
 m  =
   - 1.9999150051735
-->
```

Calcul du maximum

Il est clair que si `M` désigne le maximum de la liste de nombres donnée `A`, $M = -(Min(-A))$. Cela donne (directement dans la console à la suite du calcul précédent) :

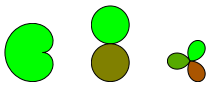
```
-->M=-(Min(-A));
-->M
 M  =
    4.9995910420548
-->
```

Bien sûr, les commandes `min` et `max` de *scilab* sont plus rapides. Il suffit de saisir dans la console `min(A)` et `max(A)` pour obtenir :

```
-->min(A)
 ans  =
  - 1.9999150051735
-->max(A)
 ans  =
```

```
    4.9995910420548
-->
```

Le calcul du maximum d'une liste de nombres est repris dans les exercices 8.1 et 8.2 à l'aide de boucles `tant que`.



6.3 [*] Quelques produits de matrices

1 - Sont donnés 2 vecteurs de n nombres : $X = [x_1, \ldots, x_n]$ et $P = [p_1, \ldots, p_n]$. On demande d'expliciter les résultats des produits matriciels suivants :

1.a - `X*ones(n,1)`

1.b - `X*X'`

1.c - `X*diag(X)*ones(n,1)`,

1.d - `X*diag(P)*X'`,

1.e - `P*diag(X)*X'`.

2 - Calculer la valeur de ces expressions avec *scilab* lorsque $X = [1, \sqrt{3}, \exp(-0.5), 2, \frac{3}{7}]$ et $P = [\frac{1}{2}, 0.23, \log 5, \frac{\pi}{8}, 7]$.

Mots-clefs : produit de matrices, commandes `ones`, `diag`, `'` (transposition), `sum`, `disp`.

`ones(n,m)` est la matrice à `n` lignes et à `m` colonnes qui ne comprend que des 1 ; `diag(x1, …, xn)` est la matrice (carrée) à `n` lignes et à `n` colonnes qui porte `x1, …, xn` sur la diagonale principale, des 0 ailleurs.

Ceci est censé être un exercice plus ou moins amusant sur le produit des matrices. Il suffit d'appliquer la définition de ce produit dans la première question. Nous n'avons pas mis de parenthèses quand on multiplie 3 matrices car c'est inutile puisque le produit des matrices est associatif, ce qu'il faut savoir aussi.

1.a - $X * \text{ones}(n,1) = x_1 + \cdots + x_n$. Du point de vue des dimensions, on pourrait écrire : $(1,n) * (n,1) \longrightarrow (1,1)$. Le résultat est une matrice à une ligne et une colonne, c'est à dire un nombre.

1.b - $X * X' = x_1^2 + \cdots + x_n^2$.

1.c - $X * \text{diag}(X) * \text{ones}(n,1) = x_1^2 + \cdots + x_n^2$.

1.d - $X * \text{diag}(P) * X' = p_1 x_1^2 + \cdots + p_n x_n^2$.

1.e - $P * \text{diag}(X) * X' = p_1 x_1^2 + \cdots + p_n x_n^2$.

Le lecteur débutant aura intérêt à calculer à la main $X * \text{diag}(P)$, $\text{diag}(P) * X'$, $(X * \text{diag}(P)) * X'$ et $X * (\text{diag}(P) * X')$, ce qui illustrera dans un cas particulier l'associativité du produit matriciel.

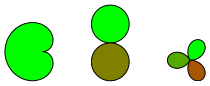
2 - On peut utiliser le script suivant :

```
X=[1,sqrt(3),exp(-0.5),2,3/7];
P=[1/2,0.23,log(5),%pi/8,7];
disp(X*ones(5,1));
disp(sum(X));
disp(X*X');
disp(sum(X.^2));
disp(X*diag(X)*ones(5,1));
disp(X*diag(P)*X');
disp(sum(P.*(X.^2)));
disp(P*diag(X)*X');
```

qui, exécuté dans la console, donne :

```
-->exec('Chemin du script', -1)
    5.7671529
    5.7671529
    8.5515529
    8.5515529
    8.5515529
```

```
    4.6385897
    4.6385897
    4.6385897
-->
```



6.4 [*] Construction de matrices

Énoncé 6.4.1 .

1 - Écrire la matrice à n lignes et m colonnes dont la première colonne ne contient que des 1, la deuxième colonne ne contient que des 2, *etc.*
2 - Écrire la matrice à m lignes et n colonnes dont la première ligne ne contient que des 1, la deuxième ligne ne contient que des 2, *etc.*

Mots-clefs : `input`, `ones(p,q)`, boucle `pour`, transposition de matrices (deuxième question).

---

1 - On peut utiliser le script suivant qui adjoint à la colonne de 1 une colonne de 2 et ainsi de suite :

```
n=input('n=');// On choisira n a l'execution du script.
m=input('m=');// On choisira m a l'execution du script.
M=ones(n,1);
    for j=2:m
      M=[M,j*ones(n,1)];
    end
afficher('M est la matrice');
afficher(M);
```

Exécutons ce script pour `n=5` et `m=7` :

```
-->exec('Chemin du fichier charge', -1)
n=5
m=7
 M est la matrice
    1.    2.    3.    4.    5.    6.    7.
    1.    2.    3.    4.    5.    6.    7.
    1.    2.    3.    4.    5.    6.    7.
    1.    2.    3.    4.    5.    6.    7.
    1.    2.    3.    4.    5.    6.    7.
-->
```

On aurait aussi pu écrire la ligne `1 ... m` puis la reproduire `n` fois, comme ceci :

```
n=input('n=');// On choisira n a l'execution du script.
m=input('m=');
M=1:m
    for i=2:n
      M=[M;1:m];
    end
afficher('M est la matrice');
afficher(M);
```

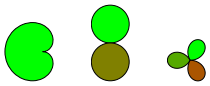
Remarquer le `;` dans la commande `[M;1:m]` qui provoque un passage à la ligne dans la construction de la matrice.

2 - On peut procéder comme précédemment, mais le plus simple est de transposer la matrice `M` que nous venons d'obtenir. Dans la console, saisissons les commandes

```
-->T=M'
```

(`T` est le nom choisi pour la matrice transposée de `M`). On obtient :

```
T  =
    1.    1.    1.    1.    1.
    2.    2.    2.    2.    2.
    3.    3.    3.    3.    3.
    4.    4.    4.    4.    4.
    5.    5.    5.    5.    5.
    6.    6.    6.    6.    6.
    7.    7.    7.    7.    7.
-->
```



6.5 [*] Sommes de nombres entiers

1 - Calculer la somme des nombres entiers de 1 à 500.
2 - Calculer la somme des carrés des nombres entiers de 1 à 500.

Mots-clefs : Boucle `pour`, `sum`, `.^2`, `afficher(A,B)`, `afficher('A ='+string(A))`.

La première solution est très élémentaire et maladroite :

```
// Liste X des 500 premiers entiers, leur somme S.
X=[];// La liste est vide au debut. On va la remplir.
S=0;
for i=1:500
    X=[X,i];// On fabrique la liste des 500 premiers entiers.
    S=S+i;// On calcule leur somme au fur et a mesure.
end;
// Liste Y des 500 premiers carres, leur somme T.
Y=[];
T=0;
for i=1:500
    Y=[Y,i^2];
    T=T+i^2;
end;
// Affichage
afficher(X,S,Y,T);
```

On rappelle que la commande `afficher(X,S,Y,T)` affiche d'abord `T`, puis `Y`, `S` et enfin `X`, à l'envers. Comme il est inutile d'afficher la liste des entiers de 1 à 500 et la liste de leurs carrés, on aurait dû se contenter de :

```
// Affichage
afficher(T,S);
```

(pour obtenir `S` d'abord) auquel cas, on obtient :

```
    125250.
    41791750.
```

Les commandes d'affichage suivantes :

```
// Affichage
afficher('La somme des 500 premiers entiers est '+string(S));
afficher('La somme des 500 premiers carres est '+string(T));
```

donnent un affichage satisfaisant :

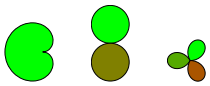
```
La somme des 500 premiers entiers est 125250
La somme des 500 premiers carres est 41791750
```

On a déjà dit que ceci est maladroit. En effet, il est tout à fait inutile de construire les suites `X` et `Y`. Il suffit d'utiliser les commandes `X=1:n` et `Y=X.^2` (La commande `.^2` est l'élévation au carré composante par composante). De plus « Pour un tableau `x`, `y=sum(x)` renvoie dans `y` la somme de tous les éléments de `x` » (d'après l'aide en ligne de *scilab*). On peut donc utiliser le script

```
X=1:500;
```

```
Y=X.^2;
S=sum(X);
T=sum(Y);
afficher('La somme des 500 premiers entiers est '+string(S));
afficher('La somme des 500 premiers carres est '+string(T));
```

qui retourne les résultats avec un affichage explicite.



6.6 [*] Trier un tableau numérique (fantaisie)

Trier un tableau numérique dans l'ordre décroissant (quand on le lit de la gauche vers la droite et de haut en bas, de la première à la dernière ligne).

Mots-clefs : commandes `trier`, `size(A,'r')`, `size(A,'c')`, `A(j,a:b)`, `A(j,:)`, `size(A)`, `A(:)` [1], `matrix(A,n,m)` [2], transposition de matrices.

• La commande `trier` trie les vecteurs, autrement dit des tableaux à une seule ligne, dans l'ordre croissant ou décroissant suivant l'option choisie. Comment l'utiliser pour trier un tableau à n lignes et m colonnes ?

• Dans les scripts ci-dessous, on écrit en ligne ou en colonne le tableau donné, on le trie et on le reconstitue.

Script n°1

• `size(T,'r')` est le nombre de lignes de la matrice `T`.

• `T(j,:)` et `T(j,a:b)` sont respectivement la `j` ème ligne de `T` et la partie de cette ligne située entre les colonnes `a` et `b` (comprises).

```
// Ranger un tableau numerique dans l'ordre decroissant.
T=input('T=');
n=size(T,'r');// Nombre de lignes de T.
m=size(T,'c');// Nombre de colonnes de T.
Ligne=T(1,:);// On se prepare a ecrire T en ligne.
for j=2:n
    Ligne=[Ligne,T(j,:)];//On ajoute la ligne de T numero j; noter la virgule.
end
Ligne=trier(Ligne,'<');// Ligne est maintenant dans l'ordre decroissant.
T=Ligne(1,1:m);
for j=2:n
    T=[T;Ligne(1,(j-1)*m+1:j*m)];
end
afficher(T);
```

En exécutant ce script, on a saisi une matrice `T` à la main. C'est long (habituellement, on utilise des matrices de nombres aléatoires) :

```
-->exec('Chemin du script', -1)
T=[-3,sqrt(2),1;0,1.17,-63;15,%pi,4;cos(3),sin(-1),14;tan(-2),0,4]
    15.                14.                4.
    4.                 3.1415926535898    2.1850398632615
    1.4142135623731    1.17               1.
    0.                 0.                 - 0.8414709848079
  - 0.9899924966004    - 3.               - 63.
-->
```

Script n°2

On peut réaliser le même programme beaucoup plus vite en utilisant les instructions *scilab* ad hoc. C'est l'objet du script suivant :

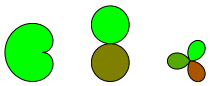
```
T=input('T=');
```

1. [11], **6.3**
2. [11], Syntaxe **6.3.3**

```
afficher(T);
[n,m]=size(T);// Nombre de lignes de T.
C=T(:);//Les colonnes de T sont empilees en une seule colonne.
D=trier(C);//Tri de la colonne C dans l'ordre croissant.
U=matrix(D,m,n)';// On reconstitue la matrice.
afficher(U);
```

dont voici une application :

```
-->exec('Chemin du script', -1)
T=[1,2,3,4,5;6,7,8,9,0;9,8,7,6,5;4,3,2,1,2]
    1.    2.    3.    4.    5.
    6.    7.    8.    9.    0.
    9.    8.    7.    6.    5.
    4.    3.    2.    1.    2.

    0.    1.    1.    2.    2.
    2.    3.    3.    4.    4.
    5.    5.    6.    6.    7.
    7.    8.    8.    9.    9.
 -->
```



6.7 [*] Matrice fantaisie (boucles `pour` imbriquées)

1 - Écrire la matrice (n,n) dont les éléments sont 1, 2, ..., n^2 écrits dans l'ordre habituel (sur chaque ligne, de la gauche vers la droite, de la première à la dernière ligne).

2 - Cette matrice étant notée M, expliquer comment se font les calculs suivants : N=M+M, Q=3*M, R=M^2.

Mots-clefs : commandes `input`, `a:b`, `matrix(A,n,m)` [3], transposition des matrices, boucles `pour` imbriquées.

1 - Solution n°1 : on définit la matrice `M` en définissant chacun de ses éléments :

```
n=input('n=');// On choisira n a l'execution du script.
for i=1:n
    for j=1:n
        M(i,j)=(i-1)*n+j;// On definit M(i,j).
    end
end
afficher(M);
```

Dans le script ci-dessus, `Mbis=[Mbis,(i-1)*n+j]` signifie que l'on adjoint à la matrice-ligne `Mbis` une composante supplémentaire (`(i-1)*n+j`), ce qui est indiqué par « , » et que la nouvelle valeur de `Mbis` est cette matrice-ligne allongée. Au contraire, dans `M=[M;Mbis]`, le « ; » indique que l'on ajoute une nouvelle ligne à `M`.

Pour $n = 9$ (on choisit `n` petit en pensant à l'affichage), on obtient :

```
--> n=9

    1.    2.    3.    4.    5.    6.    7.    8.    9.
    10.   11.   12.   13.   14.   15.   16.   17.   18.
    19.   20.   21.   22.   23.   24.   25.   26.   27.
    28.   29.   30.   31.   32.   33.   34.   35.   36.
    37.   38.   39.   40.   41.   42.   43.   44.   45.
    46.   47.   48.   49.   50.   51.   52.   53.   54.
    55.   56.   57.   58.   59.   60.   61.   62.   63.
    64.   65.   66.   67.   68.   69.   70.   71.   72.
    73.   74.   75.   76.   77.   78.   79.   80.   81.
-->
```

Solution n°2 : on fabrique `M` ligne par ligne. Il n'y a qu'une seule boucle `pour` apparente, l'autre étant prise en charge par *scilab*. Ce sera plus rapide.

```
n=input('n=');// On choisira n a l'execution du script.
L=1:n^2;
M=[];
for i=1:n
    P=L((i-1)*n+1:i*n);
    M=[M;P]
end
afficher(M);
```

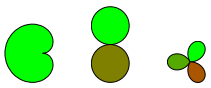
3. [11], Syntaxe **6.3.3**

Pour se rendre compte de la puissance et de la rapidité de *scilab*, le lecteur pourra tester de plus grandes valeurs, par exemple $n = 1000$. Dans ce cas, l'affichage est long et peu parlant. On l'interrompra.

Solution n°3 : on construit ci-dessous une matrice à `n` lignes et à `m` colonnes dont la première ligne est 1, ..., `m`, la suivante `n+1`, ..., `2m`, *etc.* Le script ci-dessous est rédigé de manière compacte :

```
n=input('n=');
m=input('m=');
X=matrix((1:n*m)',m,n)';
afficher(X);
```

2 - Les calculs de la deuxième question se font composante par composante.



6.8 [*] Matrices de Toeplitz

Écrire la matrice carrée de taille n comprenant des n sur la diagonale principale, des $n-1$ sur les deux lignes qui l'encadrent, etc.

Mots-clefs : `abs` (fonction valeur absolue), boucles `pour` imbriquées.

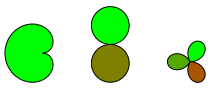
Ceci est un cas particulier de matrice de Toeplitz, voir [17].

```
n=input('n=');
for i=1:n
    for j=1:n
        A(i,j)=n-abs(i-j);
    end
end
afficher(A);
```

Cette solution ne présente aucune difficulté. C'est un script banal avec une boucle `for` insérée dans une boucle `for`.

Voici ce que l'on obtient pour $n = 6$:

```
-->clear
-->exec('Chemin du fichier charge', -1)
n=6

    6.    5.    4.    3.    2.    1.
    5.    6.    5.    4.    3.    2.
    4.    5.    6.    5.    4.    3.
    3.    4.    5.    6.    5.    4.
    2.    3.    4.    5.    6.    5.
    1.    2.    3.    4.    5.    6.
 -->
```



6.9 [*] Graphe de la fonction tangente entre 0 et π

Tracer le graphe de la fonction tangente entre 0 et π, la variable variant avec un pas de 0.1, puis 0.01, puis 0.001 et enfin 0.0001.

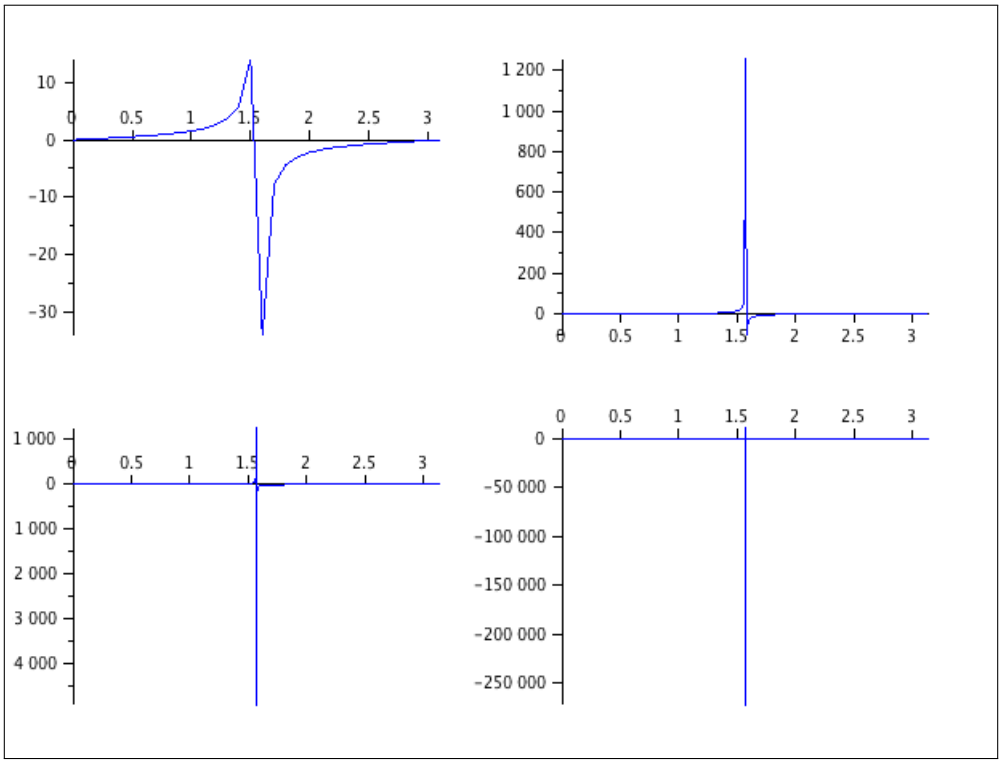
Mots-clefs : `plot`, `subplot`.

Cet exercice peut servir d'introduction à la commande `plot` et à ses dangers. Il est très instructif.

Dans le script ci-dessous, `subplot(2,2,i)` signifie que le tracé que l'on va faire est le i ème d'un tableau de 4 graphes répartis en 2 lignes et 2 colonnes, la numérotation se faisant de gauche à droite et de haut en bas.

```
clf
for i=1:4
    X=0:10^(-i):%pi;
    Y=tan(X);
    subplot(2,2,i);
    plot(X,Y);
end
```

Voici ce que l'on obtient sur la fenêtre graphique de *scilab* :



(on a simplement ajouté un cadre à la fenêtre graphique) ce qui amène les remarques suivantes :

• On rappelle que `plot` trace les points dont les coordonnées se trouvent dans `X` et `Y` et relie deux points consécutifs par un segment de droite.

• Les abscisses évitant la valeur $\dfrac{\pi}{2}$, on obtient donc des lignes polygonales. On rate donc complètement le fait que

$$\operatorname{tg} x \xrightarrow[x\to\frac{\pi}{2}-]{} +\infty \quad \text{et} \quad \operatorname{tg} x \xrightarrow[x\to\frac{\pi}{2}+]{} -\infty$$

Entre les deux points du graphe dont les abscisses encadrent $\frac{\pi}{2}$, *scilab* ignore ce qui se passe. Plus généralement, si une fonction quelconque fait des extravagances entre deux points calculés de son graphe, on ne le verra pas.

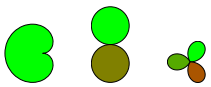
• S'il y a beaucoup de points calculés, on ne verra pas les segments qui les joignent et on aura l'impression de voir le vrai graphe de la fonction étudiée. Cette impression est trompeuse. Un graphe *scilab* ne peut remplacer une étude mathématique. Ouf.

• Lorsque le pas vaut 0.001 ou 0.0001, on s'approche trop de $\frac{\pi}{2}\cdot$ La fonction *tangente* prend des valeurs très grandes en valeur absolue. *scilab* adapte automatiquement l'échelle sur l'axe des ordonnées. Il en résulte que le graphe est écrasé sur l'axe des abscisses, sauf au voisinage de $\frac{\pi}{2}\cdot$ C'est très net sur le dernier graphe.

• Avec *scilab*, on peut tracer de nombreuses courbes (cartésiennes, paramétriques, notamment polaires) et de nombreuses surfaces et acquérir ainsi *une bonne connaissance pratique* des fonctions.

• Avec d'autres réglages de `plot` et de `plot2d`, *scilab* peut produire des nuages de points(ensembles de points).

Évidemment, il n'est pas question de démontrer avec *scilab* qu'une fonction est continue, *a fortiori* qu'elle est dérivable.



6.10 [*] Sauts de puce

Une puce fait des bonds successifs indépendants les uns des autres en ligne droite. La longueur de chaque bond est aléatoire. On considère que c'est un nombre choisi au hasard dans l'intervalle $[0, 5[$, l'unité de longueur étant le cm. On note la distance totale parcourue au bout de m bonds. On répète cette expérience n fois. Calculer la distance moyenne parcourue au cours de ces n expériences.

Mots-clefs : Commandes `rand`, `moyenne`, `stacksize('max')`, Calcul des Probabilités.

Cet exercice ne pose pas de problème d'algorithmique mais demande des connaissances en Calcul des Probabilités. La commande `rand(1,N)`, où `N` désigne un entier naturel quelconque, produit `N` nombres qui peuvent être considérés comme les valeurs prises par `N` variables aléatoires indépendantes prenant leurs valeurs dans l'intervalle $[0, 1[$ en suivant la loi uniforme. Si $N = m$ et si on multiplie ces nombres par 5, leur somme représente la distance parcourue par la puce au cours de m bonds. On répète n fois cette expérience aléatoire. Dans les deux scripts qui suivent, de rapidités très différentes, *scilab*, à l'exécution, demandera de saisir la valeur de n, puis de m.

Dans le premier script, on stocke les distances parcourues au cours des n expériences dans S à l'aide d'une boucle `pour`.

```
n=input("valeur de n :")
m=input("valeur de m :")
tic();
S=[];
for j=1:n
   sauts=5*rand(1,m);
   S=[S,sum(sauts)];
end
Moyenne=moyenne(S);
temps=toc();
afficher('La moyenne des distances parcourues est '+string(Moyenne));
afficher('Le temps de calcul est egal a '+string(temps));
```

Exécutons ce script. On obtient :

```
-->exec('Chemin du fichier charge', -1)
valeur de n : 100000
valeur de m : 50
 La moyenne des distances parcourues est 124.99021173086
 Le temps de calcul est egal a 28.256
 -->
```

Ce gros calcul (séries de 50 sauts répétées 100000 fois) a duré presque 30 secondes, ce qui est long et dû à la boucle `pour`.

On peut compacter ce script en additionnant les longueurs des $n \times m$ sauts et en divisant par n. On remarquera que l'on a dû augmenter la place allouée à la mémoire (on travaille sur une liste de 5 000 000 nombres) à l'aide de la commande `stacksize('max')`, voir l'aide.

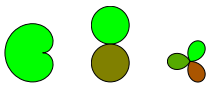
```
n=input("valeur de n :")
m=input("valeur de m :")
stacksize('max');
tic();
```

```
Moyenne=sum(5*rand(1,n*m))/n;
temps=toc();
afficher('La moyenne des distances parcourues est '+string(Moyenne));
afficher('Le temps de calcul est egal a '+string(temps));
```

Exécutons ce script. On obtient :

```
-->clear
-->exec('Chemin du fichier charge', -1)
valeur de n  : 100000
valeur de m  : 50
 La moyenne des distances parcourues est 124.97435382959
 Le temps de calcul est egal a 0.123
 -->
```

Ce calcul a donc duré un gros dizième de seconde. On remarquera que les valeurs de la distance moyenne parcourue par la puce sont différentes. C'est normal car les calculs sont basés sur des tirages aléatoires différents.



6.11 [**] Résolution graphique d'une équation

1 - Calculer les valeurs de $f(x) = \dfrac{x^3 \cdot \cos(x) + x^2 - x + 1}{x^4 - \sqrt{3} \cdot x^2 + 127}$ lorsque la variable x prend successivement les valeurs -10, -9.2, -8.4, ..., 8.4, 9.2, 10 (pas 0.8).

2 - Tracer le graphe de la fonction f quand x varie entre -10 et 10 en utilisant seulement les valeurs de $f(x)$ calculées précédemment.

3 - Apparemment, l'équation $f(x) = 0$ a une solution α voisine de 2. En utilisant le zoom, proposer une valeur approchée de α.

4 - Tracer de nouveau le graphe de f entre -10 et 10 en faisant varier x avec un pas de 0.05.

5 - Ce nouveau graphe amène-t-il à corriger la valeur de α proposée ?

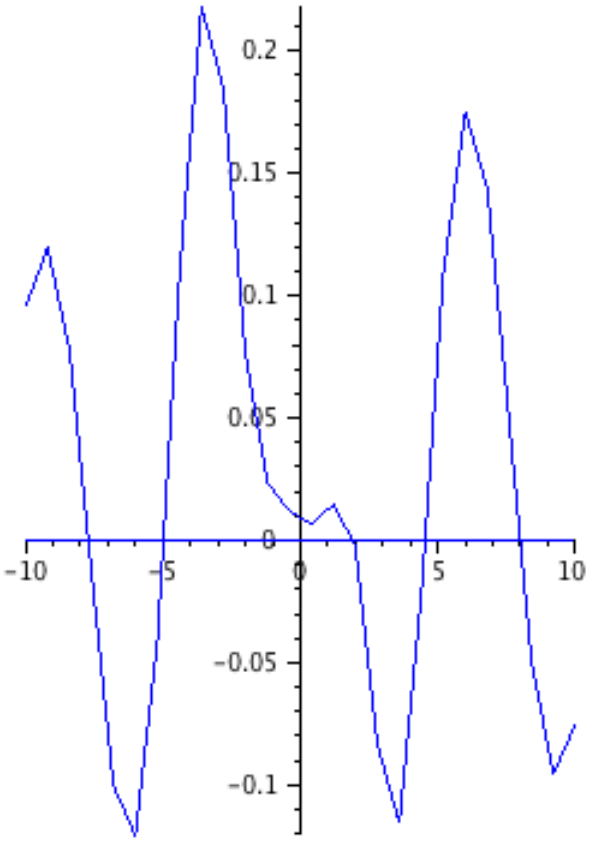
Mots-clefs : commandes `clf`, `clear`, `plot`, `subplot`, résolution graphique d'une équation, zoom, annulation du zoom.

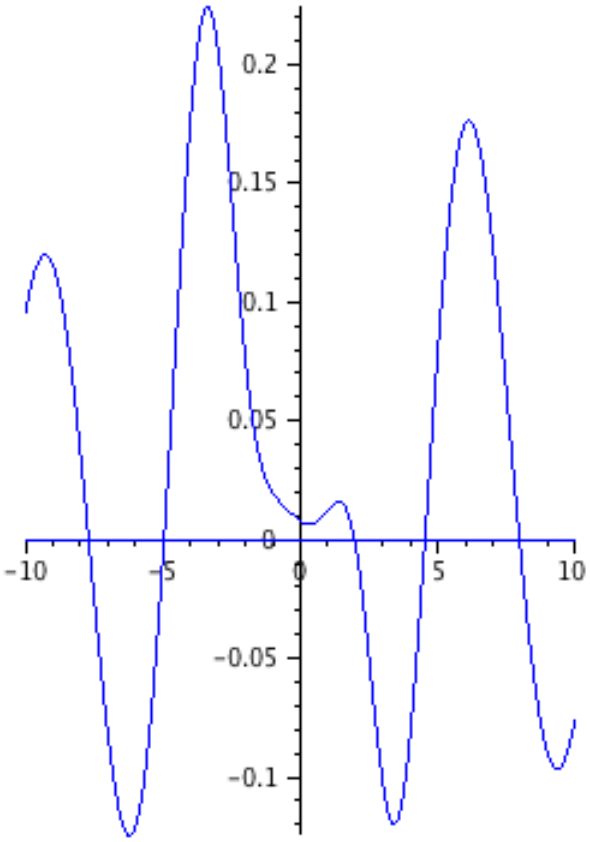
`clf` nettoie la fenêtre graphique, `clear` supprime toutes les variables non protégées, `clear a` supprime la variable `a` si elle n'est pas protégée. La commande `subplot(121)` indique que la fenêtre graphique a été divisée en deux sous-fenêtres placées côte à côte (en ligne) et que la sous-fenêtre sélectionnée (pour le graphe à venir) est la première, voir l'aide en ligne.

On rappelle que `plot(X,Y)` trace les points dont les abscisses sont dans `X`, les ordonnées dans `Y` et relie deux points consécutifs par un segment de droite. La figure ci-dessous contient deux graphes de la même fonction. Le graphe de gauche est visiblement une ligne polygonale (le nombre de points calculés est 26) ; le second en est une aussi, mais cela n'apparaît qu'en zoomant (401 points calculés). Pour les fonctions usuelles, on peut dire que le second graphe est plus précis que le premier. On pourra utiliser le script suivant qui contient une fonction *scilab* :

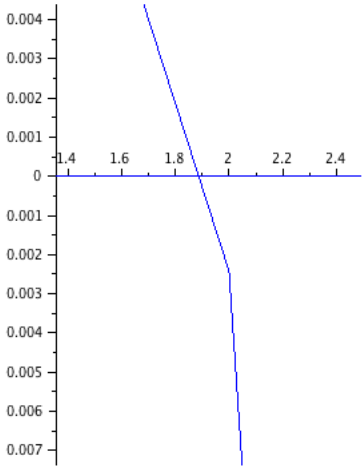
```
clear;
function y=effede(x);
    y=(x^3*cos(x)+x^2-x+1)/(x^4-sqrt(3)*x^2+127);
endfunction
X=-10:0.8:10;
Y=[];
for i=1:length(X)
    Y=[Y,effede(X(i))];
end;
clf;
subplot(121)
plot(X,Y);
X=-10:0.05:10;
Y=[];
for i=1:length(X)
    Y=[Y,effede(X(i))];
end;
subplot(122)
plot(X,Y);
```

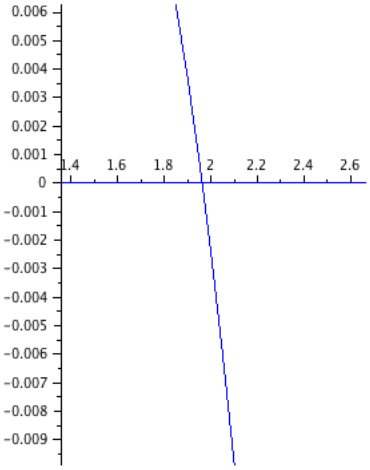
Voici la fenêtre graphique que l'on obtient :



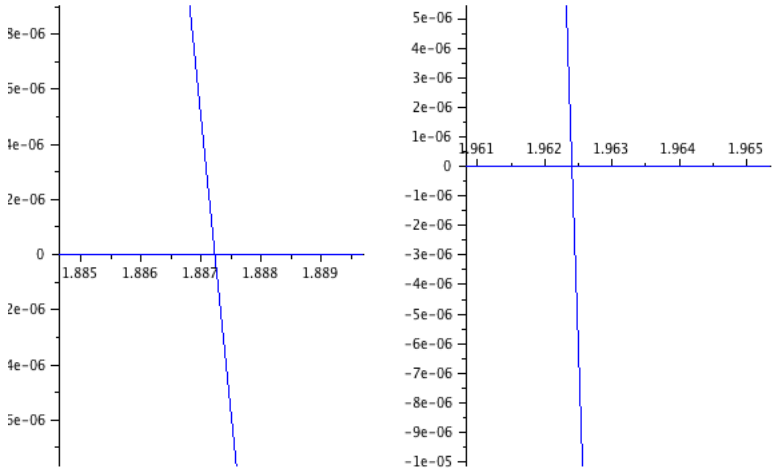


On en déduit que l'on peut proposer $\alpha \approx 2$ à partir de chacun des deux graphes.
De plus, en zoomant une première fois, on obtient :





ce qui amènera à proposer des valeurs différentes pour α suivant que l'on se sert du premier ou du second graphe. Enfin, en zoomant deux fois de plus, on arrive à :



On proposera $\alpha \approx 1.88725$ avec le premier graphe, $\alpha \approx 1.9624$ avec le second. On constate une fois de plus que la méthode graphique de résolution d'une équation dépend bien évidemment de la qualité du graphe.

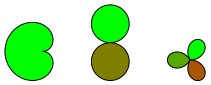
Remarques

Le script utilisé est un peu lourd. On peut transformer la fonction `effede` de manière à ce qu'elle accepte comme argument un vecteur. Cela donne (en utilisant des opérations composante par composante) :

```
clear;
function Y=effede(X);
    Y=((X.^3).*cos(X)+X.^2-X+1)./(X.^4-sqrt(3).*(X.^2)+127);
endfunction
X=-10:0.8:10;
Y=effede(X);
clf;
subplot(121)
plot(X,Y);
X=-10:0.05:10;
Y=effede(X);
subplot(122)
plot(X,Y);
```

De plus, on n'a pas vraiment besoin de `effede`. On peut finalement se contenter du script :

```
clear;
X=-10:0.8:10;
Y=((X.^3).*cos(X)+X.^2-X+1)./(X.^4-sqrt(3).*(X.^2)+127);
clf;
subplot(121)
plot(X,Y);
X=-10:0.05:10;
Y=((X.^3).*cos(X)+X.^2-X+1)./(X.^4-sqrt(3).*(X.^2)+127);
subplot(122)
plot(X,Y);
```



6.12 [**] Matrice-croix (fantaisie)

Écrire la matrice carrée C de taille 2n+1 comportant des 1 sur la $(n+1)^{\text{ième}}$ ligne et la $(n+1)^{\text{ième}}$ colonne et des 0 ailleurs.

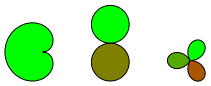
Mots-clefs : Commandes `zeros(p,q)`, `ones(p,q)`.

Le script qui suit, typique, est particulièrement compact (pour le plaisir). On peut faire moins bien ! Remarquer l'usage des `,` et des `;` dans la description de la matrice C.

```
n=input('n=');
C=[zeros(n,n),ones(n,1),zeros(n,n);ones(1,2*n+1);
zeros(n,n),ones(n,1),zeros(n,n)];
afficher(C);
```

Voici ce que l'on obtient pour $n = 4$:

```
-->exec('Chemin du script', -1)
n=4
    0.    0.    0.    0.    1.    0.    0.    0.    0.
    0.    0.    0.    0.    1.    0.    0.    0.    0.
    0.    0.    0.    0.    1.    0.    0.    0.    0.
    0.    0.    0.    0.    1.    0.    0.    0.    0.
    1.    1.    1.    1.    1.    1.    1.    1.    1.
    0.    0.    0.    0.    1.    0.    0.    0.    0.
    0.    0.    0.    0.    1.    0.    0.    0.    0.
    0.    0.    0.    0.    1.    0.    0.    0.    0.
    0.    0.    0.    0.    1.    0.    0.    0.    0.
-->
```



6.13 [**] Construction de matrices : matrice Zorro (fantaisie)

Écrire la matrice Zorro Z de taille n, c'est à dire la matrice carrée de taille n comprenant des 1 sur la première ligne, sur la deuxième diagonale et sur la dernière ligne et des 0 partout ailleurs.

Mots-clefs : commandes `ones`, `zeros`, `eye`.

La solution ci-dessous est un maniement de matrices typique de *scilab*. Z est abordée comme un assemblage de 3 matrices : la matrice `ones(1,n)`, matrice à 1 ligne et n colonnes ne comportant que des 1, puis la matrice `zeros(n-2,n)`, matrice à $n-2$ lignes et n colonnes ne comportant que des zéros et de nouveau la matrice `ones(1,n)`. Les `;` qui séparent ces matrices indiquent qu'elles sont disposées les unes en dessous des autres (en colonne). La matrice obtenue, notée Z, n'est pas la matrice demandée mais presque. Elle a n lignes et n colonnes.

Remarque : Si on avait remplaçé les `;` par des`,`, les 3 matrices précédentes auraient dû être disposées en ligne mais cela aurait provoqué un message d'erreur du script car on ne peut aligner que des matrices ayant le même nombre de lignes (de même qu'on ne peut empiler que des matrices ayant le même nombre de colonnes).

Ensuite, il reste à remplacer des zéros par des uns, en deuxième diagonale, à l'aide d'une boucle `pour`, ce que fait le script suivant :

```
n=input('n=');
Z=[ones(1,n);zeros(n-2,n);ones(1,n)];
for i=2:n-1
    Z(i,n-i+1)=1;
end
afficher(Z);
```

Voici ce que l'on obtient pour n=7 :

```
-->exec('Chemin du script', -1)
n=7
    1.    1.    1.    1.    1.    1.    1.
    0.    0.    0.    0.    0.    1.    0.
    0.    0.    0.    0.    1.    0.    0.
    0.    0.    0.    1.    0.    0.    0.
    0.    0.    1.    0.    0.    0.    0.
    0.    1.    0.    0.    0.    0.    0.
    1.    1.    1.    1.    1.    1.    1.
-->
```

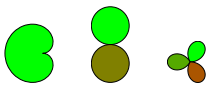
On peut préférer le script suivant dans lequel la matrice A est obtenue à partir de la matrice unité de taille n (commande `eye(n,n)`) en remplaçant les zéros de la première et de la dernière lignes par des uns. Ensuite, la matrice Z est obtenue en recopiant les colonnes de A de la dernière à la première.

```
clear
n=input('n=');
A=eye(n,n);
A(1,2:n)=ones(1,n-1);
A(n,1:n-1)=ones(1,n-1);
afficher(A);
Z=[];
```

```
for  i=1 :n
     Z=[Z,A( : ,n-i +1)];
end
afficher(Z);
```

Voici une exécution de ce script pour $n = 6$:

```
-->exec('Chemin du script', -1)
n=6

    1.    1.    1.    1.    1.    1.
    0.    1.    0.    0.    0.    0.
    0.    0.    1.    0.    0.    0.
    0.    0.    0.    1.    0.    0.
    0.    0.    0.    0.    1.    0.
    1.    1.    1.    1.    1.    1.

    1.    1.    1.    1.    1.    1.
    0.    0.    0.    0.    1.    0.
    0.    0.    0.    1.    0.    0.
    0.    0.    1.    0.    0.    0.
    0.    1.    0.    0.    0.    0.
    1.    1.    1.    1.    1.    1.
-->
```



6.14 [*] Construction de matrices : matrice barrée (fantaisie)

Écrire la matrice carrée de taille n portant des 1 sur les 2 diagonales, des 0 ailleurs.

Mots-clefs : commandes `eye`, `min`.

La commande `eye(n,n)` donne la matrice-unité de taille n ; la commande `min(A,B)` prend le minimum terme à terme des matrices de même taille `A` et `B`. On peut utiliser le script suivant :

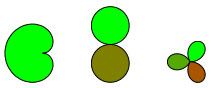
```
clear
n=input('n=');
A=eye(n,n);
B=[];
for i=1:n
    B=[B,A(:,n+1-i)];
end//La premiere colonne de B est la derniere de A, etc.
//B a ses 1 sur la deuxieme diagonale.
B=min(A+B,ones(A));// Le minimum est pris composante par composante.
afficher(B);
```

La boucle `pour` produit la matrice B qui porte des 1 sur la deuxième diagonale, des 0 ailleurs. En additionnant A et B, on obtient la matrice demandée sauf si n est impair, auquel cas le terme central de la matrice $A+B$ est 2 (l'addition des matrices se fait composante par composante). On corrige ce défaut à l'aide de la commande `min` (qui agit composante par composante).

Voici ce que l'on obtient pour $n = 9$:

```
-->exec('Chemin du script', -1)
n=9

    1.    0.    0.    0.    0.    0.    0.    0.    1.
    0.    1.    0.    0.    0.    0.    0.    1.    0.
    0.    0.    1.    0.    0.    0.    1.    0.    0.
    0.    0.    0.    1.    0.    1.    0.    0.    0.
    0.    0.    0.    0.    1.    0.    0.    0.    0.
    0.    0.    0.    1.    0.    1.    0.    0.    0.
    0.    0.    1.    0.    0.    0.    1.    0.    0.
    0.    1.    0.    0.    0.    0.    0.    1.    0.
    1.    0.    0.    0.    0.    0.    0.    0.    1.
-->
```

On peut obtenir B en multipliant A par une matrice de permutation convenable, voir l'exercice 6.21.



6.15 [**] Aiguilles de l'horloge

1 - Combien de fois l'aiguille des minutes tourne-t-elle plus vite que l'aiguille des heures ?

2 - Quand l'aiguille des heures a tourné de α degrés à partir de midi, l'extrémité de l'aiguille des minutes est repérée sur le cadran de l'horloge par l'angle

$$\beta = 12\cdot\alpha - 360\cdot k$$

si elle a déjà fait k tours complets (les angles étant comptés en degrés.). Tracer le graphe de β en fonction de α quand celui-ci varie de 0 à 360 degrés.

3 - Combien y a-t-il de superpositions des aiguilles entre midi et minuit ? Calculer les angles correspondants.

4 - Que se passe-t-il à partir de minuit ?

Dans cet énoncé, [ ] désigne la fonction partie entière.

Mots-clefs : commandes `plot(X,Y,'b')`, `plot(X,Y,'k--')`, fonction définie par morceaux.

L'option `'b'` ajoutée à la commande `plot(X,Y)` provoque le tracé du graphe en bleu ; le tracé se fera en pointillés noirs si on ajoute l'option `'k--'` à `plot(X,Y)`.

1 - Pour parcourir le cercle de l'horloge, l'aiguille des minutes met une heure, l'aiguille des heures prend 12 heures. L'aiguille des minutes tourne donc 12 fois plus vite que l'aiguille des heures. Si l'aiguille des heures tourne de α degrés, l'aiguille des minutes tourne de $12\cdot\alpha$ degrés. L'aiguille des minutes fait un tour complet quand α augmente de 30 degrés.

2 - Il y a évidemment superposition des aiguilles à midi. Nous comptons α à partir de ce moment. Pendant la deuxième heure ($30 \leqslant \alpha < 60$), quand l'aiguille des heures a tourné de α degrés (à partir de midi), l'aiguille des minutes est repérée par l'angle β, $0 \leqslant \beta < 360$, défini par $\beta = 12\cdot\alpha - 360$. De même, plus généralement, pendant la $(k+1)^{\text{ème}}$ heure, $0 \leqslant k \leqslant 11$, l'aiguille des minutes est repérée par $\beta = 12\cdot\alpha - 360\cdot k$ (β est le reste de la division euclidienne de $12\cdot\alpha$ par 360).

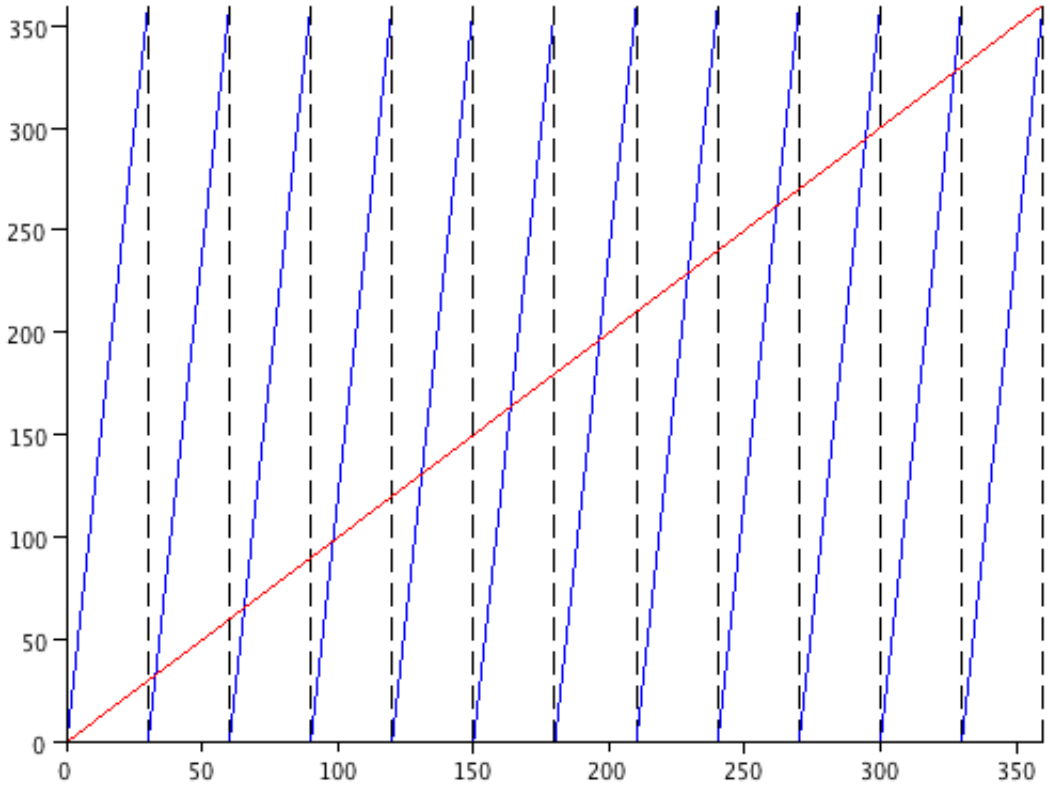
Graphe de β :

La partie du graphe de β correspondant à la $(k+1)^{\text{ème}}$ heure (quand α varie de $30\cdot k$ à $30(k+1)$) est le segment qui joint les points $(30\cdot k, 0)$ à $(30\cdot(k+1), 360)$. Le graphe de β est un ensemble de 12 segments ouverts à droite (α variant précisément sur les intervalles $[0, 30[$, ..., $[330, 360[$). Nous avons ajouté des segments verticaux en pointillés noirs pour améliorer la lisibilité du graphe. Deux courbes sont tracées : β en fonction de α et α en fonction de α (en rouge).

```
clf;
A=[];
format('v',7);
for k=0:11
    X=[k*30,(k+1)*30];
    Y=[0,360];
    plot(X,Y,'b');
    plot([(k+1)*30,(k+1)*30],[0,360],'k--');
    A=[A,360*k/11];
end;
plot([0,360],[0,360],'r');
afficher('Les angles en degres ou les aiguilles se superposent sont');
afficher(A);
```

Les lignes pointillées ne font pas partie du graphe. Elles ont été tracées pour le rendre plus lisible. La commande de tracé utilise l'option `'k--'` (k avec 2 tirets), qui signifie que la ligne sera pointillée en noir (k).

Finalement, on obtient le graphe suivant :



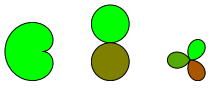
3 - La superposition des aiguilles est donnée par la solution des équations

$$\alpha = 12\cdot\alpha - 360\cdot k \quad \text{ou} \quad \alpha = \frac{360\cdot k}{11}, \quad 0 \leqslant k \leqslant 11.$$

ce qui fait 12 superpositions ($0 \leqslant \alpha < 360$).

```
-->exec('Chemin du script', -1)
 Les angles en degres ou les aiguilles se superposent sont
         column 1 to 9
    0.   32.727   65.455   98.182   130.91   163.64   196.36   229.09   261.82
         column 10 to 12
    294.55   327.27   360.
 -->
```

4 - À minuit, nous sommes revenus dans la configuration de midi et tout se répète.



6.16 [**] Calculs d'effectifs et de fréquences

Une suite S de nombres entiers étant donnée, calculer l'effectif et la fréquence de n dans cette suite, quel que soit n entre $\min(S)$ et $\max(S)$ ($\min(S) \leqslant n \leqslant \max(S)$).

Mots-clefs : Commandes `input`, `min` et `max`, `taille`, `tirage_entier`, `frequence`, matrice `zeros`, `find`.

Dans l'algorithme ci-dessous, on calcule les minimum et maximum de S notés a et b. Il y a au plus $b-a+1$ entiers différents dans la suite S.

Solution n°1 : Dans le script ci-dessous, on va stocker les effectifs des entiers de a à b dans un vecteur C de longueur $b-a+1$ initialisé par la commande `C=zeros(1,b-a+1);`. Ensuite, on passe en revue les termes successifs de S. Quand on s'occupe du $i^{\text{ème}}$ terme de S, qui est $S(i)$, on ajoute 1 à la composante de C qui compte les occurrences de la valeur de $S(i)$, qui est la composante de C de rang $S(i)-a+1$, soit $C(S(i)-a+1)$. À la fin de la boucle `pour`, C est le vecteur des effectifs.

```
S=input('S=');// S est une suite de nombres entiers.
a=min(S);
b=max(S);
C=zeros(1,b-a+1);
N=taille(S);
for i=1:N;
    C(S(i)-a+1)=C(S(i)-a+1)+1;
end;
afficher(C);
```

On exécute cet algorithme en le chargeant dans la console de *scilab*. Pour l'entrée S, nous avons choisi dans l'application ci-dessous une suite de 2011 nombres entiers tirés au hasard entre 1 et 6 (ce qui simule le lancer d'un dé 2011 fois de suite, ce qu'il est inutile de savoir). On obtient :

```
-->exec('Chemin du script', -1)
S=tirage_entier(2011,1,6)
    356.    325.    315.    334.    334.    347.
-->
```

On peut ajouter directement dans la console les commandes qui permettent de calculer les fréquences de $a, a+1, ..., b$ dans S, (ici, $a=1$ et $b=6$ puisqu'il y a 6 valeurs affichées), ce qui donne finalement, en limitant à 5 le nombre de décimales à l'affichage :

```
-->F=C/N;
-->format('v',8);
-->afficher(F);
    0.17703    0.16161    0.15664    0.16609    0.16609    0.17255
-->
```

Bien sûr, si on ré-exécute le script précédent, on obtiendra, avec une probabilité très forte, un résultat différent puisque S provient d'un tirage aléatoire.

Solution n°2 : la solution n°1 est très bonne. La seconde utilise la commande `find`, voir l'aide en ligne. Cette solution est beaucoup plus directe car on se contente de compter le nombre d'occurrences de i dans S à l'aide de la commande `taille(find(S==i))`, i variant de a à b.

```
S=input('S=');// S est une suite de nombres entiers.
a=min(S);
b=max(S);
```

```
E=zeros(1,b-a+1);
for i=a:b
    E(i-a+1)=taille(find(S==i));
end;
afficher(E);
```

Cela donne :

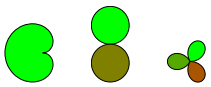
```
-->exec('Chemin du script', -1)
S=tirage_entier(2011,1,6);
    331.    362.    333.    326.    346.    313.
 -->
```

On ne retrouve pas les résultats précédents car on a fait un nouveau tirage de S. À noter l'intérêt qu'il y aurait à faire en classe les deux premières solutions. Malheureusement, la première demande beaucoup de soin. La commande `find` est à connaître par les élèves. Elle figure dans la liste *Utilitaires* du chapitre *Fonctions scilab utiles au lycée* de [12].

Solution n°3 (celle que l'on préfère quand on n'a pas de temps à perdre) : on peut retrouver immédiatement les fréquences de $1, ..., 6$ à l'aide de la commande `frequence` de *scilab* (consulter l'aide en ligne). Par exemple, la fréquence de 1 est donnée par la commande `frequence(1,S);` :

```
-->f1=frequence(1,S);
-->afficher(f1);
    0.17703
-->
```

Si on avait demandé la fréquence de 0, on aurait obtenu 0.



6.17 [**] Écrire à la fin le premier terme d'une liste de nombres

Écrire un script qui associe à tout vecteur $X = [x_1, \ldots, x_{n-1}, x_n]$ le vecteur $Y = [x_2, \ldots, x_n, x_1]$.

Mots-clefs : Commandes `taille`, `ones(p,q)`, `zeros(p,q)`, `format`, `afficher`, `X(B)` [4], `$`.

Solution n°1 : avec une boucle `pour`.

```
X=input("X=");// X est une liste de nombres.
afficher(X);
n=taille(X);
for i=1:n-1
    Y(i)=X(i+1);
end
Y(n)=X(1);
afficher(Y');// Y' est la liste demandee.
```

Nous avons appliqué ce script à `X=[-1,1/4,%e,%pi,5,log(2),tan(2*%pi/3)]`. On a choisi ci-dessous de limiter l'affichage des réels à deux décimales à l'aide de la commande `format('v',5)`. On obtient :

```
-->format('v',5);
-->exec('Chemin du script', -1)
X=[-1,1/4,%e,%pi,5,log(2),tan(2*%pi/3)]
 X =
    - 1.    0.25    2.72    3.14    5.    0.69  - 1.73
 Y =
    0.25    2.72    3.14    5.    0.69  - 1.73  - 1.
 -->
```

Remarques

• Un logiciel de calcul formel comme *Xcas* n'aurait pas écrit ces nombres en notation scientifique standard.

• Dans le script, on a demandé l'affichage de `Y'` au lieu de `Y` car par défaut, la boucle `pour` a écrit `Y` en colonne. `Y'` est le transposé de `Y`, c'est à dire `Y` écrit en ligne, ce qui fait gagner de la place.

Solution n°2 La solution n°1 est là car ce chapitre est consacré aux boucles `pour`, mais elle n'est pas bonne. Il vaut mieux manipuler des vecteurs, par exemple ajouter le premier élément de la liste X à la fin puis ne retenir que les éléments de la nouvelle liste allant du rang 2 au rang $n+1$, c'est à dire au dernier rang qui s'écrit `$`, ce qui permet d'oublier la commande `n=taille(X);` et par conséquent permet de raccourcir le script.

```
X=input("X=");
format('v',5);
afficher("X = ");afficher(X);
X=[X,X(1)];
Y=X(2:$);
afficher("Y = ");afficher(Y);
```

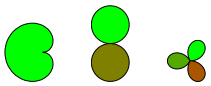
On peut aller un peu plus vite :

```
X=input("X=");
```

4. Voir l'exercice 4.2.

```
format('v',5);
afficher("X = "); afficher(X);
Y=[X(2:$),X(1)];
afficher("Y = "); afficher(Y);
```

On peut aussi utiliser un produit par la matrice de permutation *ad hoc*, voir l'exercice 6.20, mais c'est inutilement compliqué.



6.18 [*] Écrire une suite de nombres à l'envers

Écrire un script qui transforme une liste de nombres donnée $X = [x_1, \ldots, x_n]$ en la liste $Y = [x_n, \ldots, x_1]$.

Mots-clefs : commande `X(B)` [5], vecteur des éléments du vecteur `X` dont les rangs sont dans `B`.

Solution n°1 : Voici une solution laborieuse (et lente) utilisant une boucle `pour` : on écrit $X(n)$, puis $X(n-1)$, ..., $X(1)$.

```
X=input('X=');
n=length(X);
Y=[X(n)]
for i=2:n
    Y=[Y,X(n-i+1)];
end
afficher(X);
afficher(Y);
```

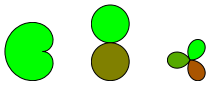
Exemple d'application :

```
-->exec('Chemin du script', -1)
X=[3,1,-2,7,4,-9,6];
    3.    1.   - 2.    7.    4.   - 9.    6.
    6.   - 9.    4.    7.   - 2.    1.    3.
-->
```

Solution n°2 : Cette solution est plus rapide car elle utilise des commandes *scilab* plus évoluées, voir la signification de la commande `X(B)` ci-dessus. Ici, `B = n:-1:1`, suite des entiers de `n` à 1 de pas -1.

```
X=input('X=');
n=length(X);
Y=X(n:-1:1);
afficher(X);
afficher(Y);
```

On peut aussi utiliser un produit par la matrice de permutation *ad hoc*, voir l'exercice 6.20, mais c'est inutilement compliqué.



5. Voir l'exercice 4.2.

6.19 [***] Permutations (solution matricielle).

Écrire toutes les permutations de 1, 2, ..., n.
(solution matricielle)

Mots-clefs : Commandes `stacksize`, `size`, `ones(p,q)`, `perms`.

Dans le script ci-dessous, pour calculer toutes les permutations des entiers de 1 à n (qui seront écrites en colonne, ce qui donnera une matrice à $n!$ lignes et à n colonnes), on écrit toutes les permutations de 1, puis de (1, 2), puis de (1, 2, 3), *etc*,

$$1 \quad \longrightarrow \quad \begin{matrix} 2 & 1 \\ 1 & 2 \end{matrix} \quad \longrightarrow \quad \begin{matrix} 3 & 2 & 1 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \\ 1 & 3 & 2 \\ 2 & 1 & 3 \\ 1 & 2 & 3 \end{matrix} \quad \longrightarrow$$

et on s'arrête quand on a obtenu les permutations de $(1, 2, \ldots, n)$. Pour passer d'une permutation quelconque de $(1, 2, \ldots, k)$ à une permutation de $(1, 2, \ldots, (k+1))$, on peut ajouter $k+1$ devant cette permutation, ou bien après le premier terme, ou le second, *etc*, ou finalement après le dernier terme. Cela promet d'être long.

En fait, nous allons travailler matriciellement : nous partons de la matrice 1, puis nous ajoutons 2 devant 1 puis derrière 1 pour obtenir les 2 permutations de (1, 2) écrites l'une au-dessous de l'autre, puis nous construisons les permutations de (1, 2, 3) en partant de la matrice (2,2) que nous venons d'obtenir et en ajoutant une colonne de deux 3 devant la première colonne, puis derrière la première colonne, puis derrière la seconde. On empile les 3 matrices à 2 lignes et 3 colonnes ainsi obtenue. Et ainsi de suite. Cela conduit à la fonction *scilab* suivante :

```
function y=f_permut(M);// M est une matrice.
    [r,c]=size(M);//r : nombre de lignes, c, de colonnes.
    A=(c+1)*ones(r,1);// Colonne de (c+1) a r lignes.
    y=[];
    for j=0:c
        B=[M(:,1:j),A,M(:,j+1:c)];//Matrice a r lignes, c+1 colonnes :
        //on a intercale A au debut, entre les colonnes, a la fin.
        y=[y;B];
    end
endfunction
```

On charge cette fonction dans la console de *scilab*. Dans le script qui suit, elle est chargée de calculer le passage de la matrice des permutations de $(1, 2, \ldots, k)$ à la matrice des permutations de $(1, 2, \ldots, (k+1))$. Le script lui-même définit la fonction *scilab* qui à `n` associe la liste des permutations de 1, ..., `n` et le temps du calcul.

La commande `stacksize('max');` alloue le maximum de mémoire à *scilab*.

```
function liste_permut(n);// n est un entier >0.
    tic();
    M=[1];
    for j=1:n-1
        M=f_permut(M);
    end;
    temps=toc();
    afficher(M);
    afficher('La duree du calcul, en secondes, est '+string(temps))
endfunction
```

On charge la deuxième fonction dans la console. Voici ce que l'on obtient pour $n = 4$:

```
-->liste_permut(4)
    4.    3.    2.    1.
    4.    3.    1.    2.
    4.    2.    3.    1.
    4.    1.    3.    2.
    4.    2.    1.    3.
    4.    1.    2.    3.
    3.    4.    2.    1.
    3.    4.    1.    2.
    2.    4.    3.    1.
    1.    4.    3.    2.
    2.    4.    1.    3.
    1.    4.    2.    3.
    3.    2.    4.    1.
    3.    1.    4.    2.
    2.    3.    4.    1.
    1.    3.    4.    2.
    2.    1.    4.    3.
    1.    2.    4.    3.
    3.    2.    1.    4.
    3.    1.    2.    4.
    2.    3.    1.    4.
    1.    3.    2.    4.
    2.    1.    3.    4.
    1.    2.    3.    4.
 La duree du calcul, en secondes, est 0.001
-->
```

Quand n grandit, le temps d'affichage s'allonge très vite, le temps de calcul aussi. À partir de $n = 9$, l'affichage est vraiment trop long. Heureusement, il suffit souvent de calculer les permutations sans les afficher. Supprimons donc l'affichage de M en remplaçant `afficher(M);` par `//afficher(M);` (la commande d'affichage est devenue un commentaire). Pour $n = 9$, puis 10, puis 11 (rappel : $9\,! = 362880$, $10\,! = 3628800$, $11\,! = 39916800$), on obtient :

```
-->liste_permut(9)
 La duree du calcul, en secondes, est 0.478
-->liste_permut(10)
  La duree du calcul, en secondes, est 5.777
 -->liste_permut(11)
 !--error 17
Taille de la pile depassee
Utilisez la fonction stacksize pour l'augmenter.
Memoire utilisee pour les variables : 119759967
Memoire intermediaire requise : 159667215
Memoire totale disponible : 268435455
at line       8 of function f_permut called by :
at line       5 of function liste_permut called by :
liste_permut(11)
-->
```

Cet algorithme n'écrit donc les permutations de $(1, 2, \ldots, n)$ que lorsque $n \leqslant 10$. Les problèmes de taille

de mémoire sont fréquents dans les dénombrements.

Remarque : La commande `perms(x)` retourne toutes les permutations des composantes d'un vecteur `x`. Par exemple

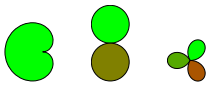
```
-->perms(1:4)
 ans  =
    4.    3.    2.    1.
    4.    3.    1.    2.
    4.    2.    3.    1.
    4.    2.    1.    3.
    4.    1.    3.    2.
    4.    1.    2.    3.
    3.    4.    2.    1.
    3.    4.    1.    2.
    3.    2.    4.    1.
    3.    2.    1.    4.
    3.    1.    4.    2.
    3.    1.    2.    4.
    2.    4.    3.    1.
    2.    4.    1.    3.
    2.    3.    4.    1.
    2.    3.    1.    4.
    2.    1.    4.    3.
    2.    1.    3.    4.
    1.    4.    3.    2.
    1.    4.    2.    3.
    1.    3.    4.    2.
    1.    3.    2.    4.
    1.    2.    4.    3.
    1.    2.    3.    4.
-->
```

ou

```
-->perms(['Anatole','Anthelme','Athanase','Antoine'])
 ans  =

!Antoine    Athanase   Anthelme   Anatole    !
!                                             !
!Antoine    Athanase   Anatole    Anthelme   !
!                                             !
!Antoine    Anthelme   Athanase   Anatole    !
!                                             !
!Antoine    Anthelme   Anatole    Athanase   !
!                                             !
!Antoine    Anatole    Athanase   Anthelme   !
!                                             !
!Antoine    Anatole    Anthelme   Athanase   !
!                                             !
!Athanase   Antoine    Anthelme   Anatole    !
!                                             !
!Athanase   Antoine    Anatole    Anthelme   !
!                                             !
```

```
!Athanase   Anthelme   Antoine    Anatole    !
!                                             !
!Athanase   Anthelme   Anatole    Antoine    !
!                                             !
!Athanase   Anatole    Antoine    Anthelme   !
!                                             !
!Athanase   Anatole    Anthelme   Antoine    !
!                                             !
!Anthelme   Antoine    Athanase   Anatole    !
!                                             !
!Anthelme   Antoine    Anatole    Athanase   !
!                                             !
!Anthelme   Athanase   Antoine    Anatole    !
!                                             !
!Anthelme   Athanase   Anatole    Antoine    !
!                                             !
!Anthelme   Anatole    Antoine    Athanase   !
!                                             !
!Anthelme   Anatole    Athanase   Antoine    !
!                                             !
!Anatole    Antoine    Athanase   Anthelme   !
!                                             !
!Anatole    Antoine    Anthelme   Athanase   !
!                                             !
!Anatole    Athanase   Antoine    Anthelme   !
!                                             !
!Anatole    Athanase   Anthelme   Antoine    !
!                                             !
!Anatole    Anthelme   Antoine    Athanase   !
!                                             !
!Anatole    Anthelme   Athanase   Antoine    !
-->
```

La commande `perms` pose les mêmes problèmes de dépassement de capacité que précédemment.



6.20 [***] Matrices de permutation

1 - Une permutation σ de l'ensemble $(1, 2, \ldots, n)$ étant donnée, écrire la matrice de permutation P de σ (voir les explications ci-dessous).

2 - Soit X un vecteur de longueur n. Vérifier sur un exemple que $Y = X \cdot P$ s'obtient en permutant selon σ les composantes de X.

3 - Quelle matrice de permutation M permet d'écrire les composantes de X dans l'ordre inverse ?

Mots-clefs : `eye(n,n)`, produit matriciel (dont la définition est rappelée à l'exercice 4.6).

Explications : Une permutation σ de $(1, \ldots, n)$ peut être décrite par le vecteur `sigma` des images successives des nombres $1, \ldots, n$ par σ. Une matrice de permutation de taille n est une matrice carrée à n lignes et colonnes, chacune d'elles comprenant un 1 et $n-1$ zéros. Nous disposons d'une excellente matrice de permutation de taille n, à savoir la matrice-unité I donnée par la commande `eye(n,n)`. Nous allons voir que la matrice de permutation P de σ s'obtient à partir de I en remplaçant sa première colonne par la colonne de rang $\sigma(1)$ de I, sa deuxième colonne par la colonne $\sigma(2)$ de I, *etc.* Cela veut dire que si X est un vecteur de longueur n, $Y = X \cdot P$ est le vecteur obtenu en effectuant la permutation σ sur les composantes de X. *scilab* nous permet de constater cela sur des exemples. La démonstration, qui s'appuie sur la définition du produit des matrices, est évidente.

La fonction `matpermut` ci-dessous a pour entrée `n` et `sigma` et retourne la matrice `P` que nous venons de décrire :

```
function P=matpermut(n,sigma)
    I=eye(n,n);
    P=[];
    for i=1:n
        P=[P,I(:,sigma(i))];
    end
endfunction
```

Nous répondons maintenant à la question 3, donc aussi à la question 2. La permutation à utiliser est $\sigma = n, n-1, \ldots, 1$. Pour $n = 15$, X étant une liste de 15 nombres entiers choisis au hasard entre -3 et 7, cela donne, la fonction `matpermut` - téléchargeable - ayant été chargée dans la console :

```
-->n=15;
-->sigma=n:-1:1;
-->P=matpermut(n,sigma)
 P  =
        column 1 to 10
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.    0.    0.    0.    0.    1.
    0.    0.    0.    0.    0.    0.    0.    0.    1.    0.
    0.    0.    0.    0.    0.    0.    0.    1.    0.    0.
    0.    0.    0.    0.    0.    0.    1.    0.    0.    0.
    0.    0.    0.    0.    0.    1.    0.    0.    0.    0.
    0.    0.    0.    0.    1.    0.    0.    0.    0.    0.
    0.    0.    0.    1.    0.    0.    0.    0.    0.    0.
```

```
    0.    0.    1.    0.    0.    0.    0.    0.    0.    0.
    0.    1.    0.    0.    0.    0.    0.    0.    0.    0.
    1.    0.    0.    0.    0.    0.    0.    0.    0.    0.
       column 11 to 15
    0.    0.    0.    0.    1.
    0.    0.    0.    1.    0.
    0.    0.    1.    0.    0.
    0.    1.    0.    0.    0.
    1.    0.    0.    0.    0.
    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.
    0.    0.    0.    0.    0.
-->X=tirage_entier(15,-3,7)
 X  =
       column  1 to 10
    3.    0.    7.    6.    1.    4. - 3.    3.    2.    1.
       column 11 to 15
   - 1.    0.    0.    5. - 3.
-->Y=X*P
 Y  =
       column  1 to 10
   - 3.    5.    0.    0. - 1.    1.    2.    3. - 3.    4.
       column 11 to 15
    1.    6.    7.    0.    3.
-->
```

M est donc la matrice qui porte des 1 sur la diagonale non-principale, des 0 ailleurs.

Remarques :

• Cet exercice fournit une nouvelle solution de l'exercice 4.2 : il suffit d'utiliser la permutation σ donnée par

```
sigma=[1:i-1,j,i+1:j-1,i,j+1:n];
```

(si l'on suppose $i < j$) ;

• cet exercice fournit aussi une nouvelle solution de l'exercice 6.18.

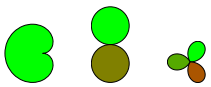
Solution n°3 : avec une matrice de permutation. C'est la meilleure solution dans le cas de calculs ultérieurs. Voir l'exercice 6.20 pour les matrices de permutation. La matrice `M` du script ci-dessous est la matrice de permutation de taille `n` qui réalise la transformation $X = [x_1, \ldots, x_{n-1}, x_n]$ donne $Y = [x_2, \ldots, x_n, x_1]$ par le produit matriciel $Y = X \cdot M$.

```
X=input("X=");
format('v',5);
afficher("X = "); afficher(X);
n=taille(X);
M=[zeros(1,n-1),1;eye(n-1,n-1),zeros(n-1,1)];
Y=X*M;
afficher("Y = "); afficher(Y);
```

```
Z=X*M^n;
afficher("Z = "); afficher(Z);
```

Nous avons appliqué ce script à `X=[-1,1/4,%e,%pi,5,log(2),tan(2*%pi/3)]` ($n = 7$). Nous avons calculé en plus $Z = X \cdot M^7$. On constate qu'en appliquant la transformation 7 fois, on trouve $Z = X \cdot M^7 = X$, ce qui était attendu car M^7 est évidemment la matrice-identité de taille 7.

```
-->exec('Chemin du script', -1)
X=[-1,1/4,%e,%pi,5,log(2),tan(2*%pi/3)]
 X =
   - 1.    0.25    2.72    3.14    5.    0.69  - 1.73
 Y =
    0.25    2.72    3.14    5.    0.69  - 1.73  - 1.
 Z =
   - 1.    0.25    2.72    3.14    5.    0.69  - 1.73
 -->
```



6.21 [***] Échanger deux lignes ou deux colonnes d'un tableau (produit d'une matrice par une matrice de permutation)

1 - M étant un tableau de nombres à n lignes et m colonnes,

1.a - soit σ_1 une permutation de $(1, \ldots, m)$ (au choix du lecteur) et P_1 la matrice de permutation qui lui est associée (voir l'exercice 6.20) ; comparer M et $M * P_1$ (effet sur une matrice de la multiplication à droite par une matrice de permutation) ;

1.b - soit σ_2 une permutation de $(1, \ldots, n)$ (au choix du lecteur) et P_2 la matrice de permutation qui lui est associée ; comparer M et $P_2 * M$ (effet sur une matrice de la multiplication à gauche par une matrice de permutation) ;

2 - Engendrer un tableau de nombres entiers M à n lignes et à m colonnes à l'aide de la commande `grand(n,m,"uin",-13,14)`.

2 .a - Échanger les lignes 2 et $n-1$ de M,

2.b - Échanger les colonnes 2 et $m-1$ de M.

Mots-clefs : matrices de permutation, multiplication à gauche et à droite.

Ce problème a déjà été traité, voir l'exercice 4.3. Ici, on utilise des matrices de permutation. Les matrices de permutation P_1 et P_2 seront calculées à l'aide de la fonction *scilab* `matpermut.sci` téléchargeable, voir l'exercice 6.20. Il convient donc de charger cette fonction dans la console. On utilisera le script suivant :

```
n=input('n=');
m=input('m=');
M=grand(n,m,"uin",-13,14);
s1=input('s1=');//Permutation de (1,...,m).
P1=matpermut(m,s1);
afficher(M);
afficher(M*P1);
s2=input('s2=');//Permutation de (1,...,n).
P2=matpermut(n,s2);
afficher(M);
afficher(P2*M);
```

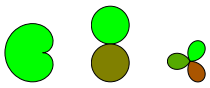
Lors de l'exécution de ce script, le logiciel demandera quelles valeurs ont été choisies pour `n`, `m`, `s1` (pour σ_1) et plus tard `s2` (pour σ_2). Ci-dessous, on a choisi respectivement 5, 9, $[1,8,3,4,5,6,7,2,9]$ et $[1,4,3,2,5]$. Cela donne finalement

```
-->exec('Chemin de matpermut.sci', -1)
-->exec('Chemin du script', -1)
n=5
m=9
s1=[1,8,3,4,5,6,7,2,9]

    2.   - 11.     9.     12.     8.     2.   - 1.     8.     4.
    9.   - 11.   - 2.     7.     8.     6.     6.   - 13.    14.
    1.     13.     1.     4.   - 2.     10.  - 6.   - 5.     7.
  - 5.     3.   - 4.     9.   - 10.    1.     0.   - 1.     6.
  - 8.   - 10.  - 10.   - 3.   - 10.    5.   - 3.   - 1.     10.

    2.     8.     9.     12.     8.     2.   - 1.   - 11.    4.
```

```
    9.   - 13.   - 2.     7.     8.     6.     6.   - 11.    14.
    1.   - 5.     1.     4.   - 2.     10.   - 6.     13.    7.
  - 5.   - 1.   - 4.     9.   - 10.     1.     0.     3.     6.
  - 8.   - 1.   - 10.   - 3.   - 10.     5.   - 3.   - 10.    10.
s2=[1,4,3,2,5]

    2.   - 11.    9.     12.    8.     2.   - 1.     8.     4.
    9.   - 11.   - 2.     7.     8.     6.     6.   - 13.    14.
    1.     13.    1.     4.   - 2.     10.   - 6.   - 5.     7.
  - 5.     3.   - 4.     9.   - 10.     1.     0.   - 1.     6.
  - 8.   - 10.   - 10.   - 3.   - 10.     5.   - 3.   - 1.     10.

    2.   - 11.    9.     12.    8.     2.   - 1.     8.     4.
  - 5.     3.   - 4.     9.   - 10.     1.     0.   - 1.     6.
    1.     13.    1.     4.   - 2.     10.   - 6.   - 5.     7.
    9.   - 11.   - 2.     7.     8.     6.     6.   - 13.    14.
  - 8.   - 10.   - 10.   - 3.   - 10.     5.   - 3.   - 1.     10.
-->
```

On constate que $M * P_1$ est obtenu à partir de M en échangeant les colonnes n°2 et n°8. De même, $P_2 * M$ est obtenu à partir de M en échangeant les lignes n°2 et n°8. Ce choix des permutations σ_1 et σ_2 fait que la deuxième question a été traitée en même temps que la première.

Il est facile de démontrer, en appliquant seulement la définition du produit des matrices, que multiplier à droite par une matrice de permutation revient à faire cette permutation sur les colonnes et que multiplier à gauche revient à faire la permutation sur les lignes. Dans le cas $n = 1$, on retrouve l'exercice 6.20.



6.22 [**] Calcul matriciel de l'espérance et de la variance d'une variable aléatoire finie

1 - Soit X une variable aléatoire ne prenant que les valeurs v_1, ..., v_n avec respectivement les probabilités p_1, ..., p_n (> 0). Exprimer matriciellement son espérance *moy* et sa variance *var* à l'aide des vecteurs $V = [v_1, \ldots, v_n]$ et $prob = [p_1, \ldots, p_n]$.

2 - Application : calculer l'espérance et la variance d'une variable aléatoire suivant la loi binomiale de taille 10 et de paramètres $\frac{1}{3}$.

Mots-clefs : calcul des probabilités, calcul matriciel, espérance, variance, commandes : `loi_binomiale(n,p)`, `afficher`.

Cet exercice montre le grand intérêt du calcul matriciel. Les calculs sont explicites et rapides. On n'a pas besoin d'utiliser les commandes `moyenne_ponderee` et `variance_ponderee` de *scilab*.

1 - Par définition de la moyenne d'une variable aléatoire finie,

$$moy = p_1 v_1 + \cdots + p_n v_n$$

ce qui signifie, par définition du produit des matrices, que

$$moy = prov \cdot V'$$

si l'on note V' la matrice transposée de V, qui est le vecteur V écrit en colonne. De même,

$$var = p_1(v_1 - moy)^2 + \cdots + p_n(v_n - moy)^2 \quad \text{ou} \quad var = prov \cdot W$$

où W est la matrice-colonne dont les éléments sont successivement $(v_1 - moy)^2, \ldots, (v_n - moy)^2$. Se rappelant que `.^2` est la commande d'élévation au carré terme à terme des éléments d'une matrice, on en déduit la fonction-*scilab* `MoyVar` suivante qui, à partir de toute variable aléatoire dont la loi est donnée par V et *prob*, retourne son espérance et sa variance.

Listing 6.1 – Fonction MoyVar

```
function [m,v]=MoyVar(V,prob)
    m=prob*V';
    v=prob*((V-m).^2)';
endfunction
```

Au lieu de `((V-m).^2)'`, on aurait pu écrire `((V-m*ones(n,1)).^2)'`. On aurait pu aussi intervertir la transposition et l'élévation au carré terme à terme. Cette question n'a évidemment rien à voir avec *scilab*.

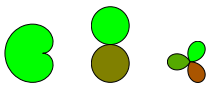
2 - Le script suivant répond à la deuxième question :

```
function [m,v]=MoyVar(V,prob)
    m=prob*V';
    v=prob*((V-m).^2)';
endfunction
n=10;
p=1/3;
proba=loi_binomiale(n,p);//Vecteur des probabilites individuelles de B(n,p).
val=0:10;
[Moyenne,Variance]=MoyVar(val,proba);
afficher('La moyenne et la variance de cette loi binomiale sont respectivement egales a');
```

```
afficher([Moyenne,Variance]);
mo=n*p;
va=n*p*(1-p);
afficher('Confirmation');
afficher([mo,va]);
```

Les 4 dernières lignes calculent directement la moyenne et la variance de la loi binomiale de taille n et de paramètre p[6]. On peut ainsi les comparer aux résultats précédents. L'exécution de ce script retourne :

```
-->exec('Chemin de la fonction MoyVar', -1)
 La moyenne et la variance de cette loi binomiale sont respectivement
 egales a
    3.3333333333333    2.2222222222222
 Confirmation
    3.3333333333333    2.2222222222222
-->
```



6. On sait que la moyenne et la variance de la loi binomiale de taille n et de paramètre p sont respectivement np et $np(1-p)$.

6.23 [***] Calcul matriciel de la covariance de 2 variables aléatoires finies

Soit X et Y deux variables aléatoires finies, $V_X = [x_1, \dots, x_n]$ et $V_Y = [y_1, \dots, y_m]$ les vecteurs des valeurs qu'elles prennent avec des probabilités > 0. La loi du couple aléatoire (X, Y) est décrite par la matrice $P_{(X,Y)} = (p_{i,j},\ 1 \leqslant i \leqslant n\,;\ 1 \leqslant j \leqslant m)$ où $p_{i,j} = P\left(X = x_i\,, Y = y_j\right)$.

1 - Comment calcule-t-on la loi P_X de X à partir de $P_{(X,Y)}$? Même question pour la loi P_Y de Y.

2 - Exprimer matriciellement les espérances m_X et m_Y de X et Y.

3 - Exprimer matriciellement la variance $\mathrm{var}\,(X)$ de X.

4 - Exprimer matriciellement la covariance $\mathrm{cov}\,(X, Y)$ de X et Y.

5 - Application : Calculer les quantités ci-dessus lorsque $P_{(X,Y)}$, V_X et V_Y sont respectivement égales à :

$P_{(X,Y)} =$

0.002	0.012	0.004	0.005	0.017	0.012	0.004	0.001	0.003	0.001	0.005	0.	0.009	0.002	0.001
0.011	0.002	0.003	0.01	0.009	0.002	0.009	0.002	0.002	0.006	0.002	0.003	0.011	0.007	0.006
0.003	0.001	0.007	0.001	0.009	0.003	0.009	0.005	0.006	0.004	0.002	0.004	0.003	0.011	0.01
0.001	0.002	0.007	0.012	0.002	0.	0.	0.002	0.003	0.008	0.006	0.009	0.005	0.002	0.008
0.01	0.011	0.001	0.01	0.004	0.004	0.	0.	0.004	0.	0.012	0.	0.005	0.01	0.009
0.001	0.	0.004	0.012	0.001	0.012	0.001	0.004	0.011	0.01	0.005	0.004	0.004	0.012	0.007
0.002	0.012	0.003	0.004	0.006	0.005	0.	0.01	0.004	0.012	0.003	0.004	0.	0.004	0.01
0.009	0.011	0.012	0.009	0.005	0.021	0.011	0.02	0.011	0.003	0.003	0.007	0.008	0.005	0.005
0.006	0.005	0.001	0.004	0.008	0.012	0.007	0.008	0.012	0.	0.	0.006	0.001	0.004	0.001
0.	0.006	0.012	0.01	0.002	0.011	0.006	0.011	0.023	0.023	0.009	0.003	0.011	0.01	0.007
0.003	0.013	0.	0.003	0.002	0.008	0.009	0.006	0.021	0.	0.004	0.003	0.005	0.002	0.007

$$V_X = [1., 1.3, 1.6, 1.9, 2.2, 2.5, 2.8, 3.1, 3.4, 3.7, 4.] \text{ et}$$
$$V_Y = [-4., -3.3, -2.6, -1.9, -1.2, -0.5, 0.2, 0.9, 1.6, 2.3, 3., 3.7, 4.4, 5.1, 5.8]$$

Mots-clefs : espérance, variance, covariance, lois marginales, commandes `sum(M,"c")`, `sum(M,"r")`, `diag`.

Cet exercice suppose connues certaines notions élémentaires du Calcul des probabilités. Pour l'espérance et la variance d'une variable aléatoire finie, on peut se reporter à 6.22.

1 - On peut supposer que Y ne prend pas d'autres valeurs que celles de V_Y. Dans ces conditions,

$$(X = x_i) = \bigcup_{j=1}^{m} (X = x_i\,, Y = y_j)$$

Cette réunion étant disjointe et la probabilité étant additive,

$$P(X = x_i) = \sum_{j=1}^{m} P\left(X = x_i\,, Y = y_j\right)$$

La $i^{\text{ème}}$ probabilité de la loi de X s'obtient donc en additionnant les éléments de la matrice $P_{(X,Y)}$ qui sont situés sur sa $i^{\text{ème}}$ ligne. Les sommes sur toutes les lignes formeront un vecteur-colonne qu'il faudra transposer pour obtenir un vecteur-ligne (commande `PX=(sum(PXY,"c"))';` dans le script ci-dessous). De même, la loi de Y s'obtient en additionnant les colonnes (commande `PY=sum(PXY,"r");` dans le script ci-dessous)[7].

2 et 3 - D'après l'exercice 6.22, $m_X = P_X * (V_X)'$, $m_Y = P_Y * (V_Y)'$. et $\mathrm{var}\,(X) = P_X * W$ où W est la matrice-colonne dont les éléments sont successivement $(x_1 - m_X)^2, \dots, (x_n - m_X)^2$.

4 - Par définition, la covariance de X et Y est l'espérance de la variable aléatoire $(X - m_X)\cdot(Y - m_Y)$. Comme cette variable aléatoire prend les valeurs $(x_i - m_X)(y_j - m_Y)$ avec les probabilités $p_{i,j}$,

$$\mathrm{cov}\,(X, Y) = \sum_{i=1}^{n}\sum_{j=1}^{m} p_{i,j}\cdot(x_i - m_X)(y_j - m_Y) = (X - m_X)\cdot P_{(X,Y)}\cdot(Y - m_Y)$$

7. P_X et P_Y s'appellent les lois marginales de (X, Y).

5 - Voir le script ci-dessous et son application.

Listing 6.2 – Covariance de X et Y (téléchargeable)

```
clear;
PXY=[2,12,4,5,17,12,4,1,3,1,5,0,9,2,1;
11,2,3,10,9,2,9,2,2,6,2,3,11,7,6;
3,1,7,1,9,3,9,5,6,4,2,4,3,11,10;
1,2,7,12,2,0,0,2,3,8,6,9,5,2,8;
10,11,1,10,4,4,0,0,4,0,12,0,5,10,9;
1,0,4,12,1,12,1,4,11,10,5,4,4,12,7;
2,12,3,4,6,5,0,10,4,12,3,4,0,4,10;
9,11,12,9,5,21,11,20,11,3,3,7,8,5,5;
6,5,1,4,8,12,7,8,12,0,0,6,1,4,1;
0,6,12,10,2,11,6,11,23,23,9,3,11,10,7;
3,13,0,3,2,8,9,6,21,0,4,3,5,2,7
]/1000;
disp(PXY);
VX=1:0.3:4;//10 valeurs.
disp('Les valeurs prises par X sont');
disp(VX);
VY=-4:0.7:6.4;//15 valeurs.
disp('Les valeurs prises par Y sont');
disp(VY);
PX=(sum(PXY,"c"))';//On obtient une ligne.
disp('La loi de X est');
disp(PX);
PY=sum(PXY,"r");//Idem
disp('La loi de Y est');
disp(PY);
EX=PX*(VX');
disp('L''esperance de X est');
disp(EX);
EY=PY*VY';
disp('L''esperance de Y est');
disp(EY);
VarX=PX*((VX-EX).^2)';
disp('La variance de X est');
disp(VarX);
CovXY=(VX-EX)*PXY*((VY-EY)');
disp('La covariance de X et Y est');
disp(CovXY);
// Autre calcul de la variance de X :
PXX=diag(PX);
A=(VX-EX)*PXX*((VX-EX)');
disp('La variance de X est');
disp(A);
```

Application :

Listing 6.3 – Calculs

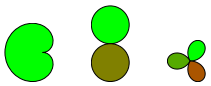
```
-->exec('Chemin du script', -1)
          column 1 to 7
    0.002     0.012     0.004     0.005     0.017     0.012     0.004
```

```
    0.011     0.002     0.003     0.01      0.009     0.002     0.009
    0.003     0.001     0.007     0.001     0.009     0.003     0.009
    0.001     0.002     0.007     0.012     0.002     0.        0.
    0.01      0.011     0.001     0.01      0.004     0.004     0.
    0.001     0.        0.004     0.012     0.001     0.012     0.001
    0.002     0.012     0.003     0.004     0.006     0.005     0.
    0.009     0.011     0.012     0.009     0.005     0.021     0.011
    0.006     0.005     0.001     0.004     0.008     0.012     0.007
    0.        0.006     0.012     0.01      0.002     0.011     0.006
    0.003     0.013     0.        0.003     0.002     0.008     0.009
         column  8 to 14
     0.001     0.003     0.001     0.005     0.        0.009     0.002
    0.002     0.002     0.006     0.002     0.003     0.011     0.007
    0.005     0.006     0.004     0.002     0.004     0.003     0.011
    0.002     0.003     0.008     0.006     0.009     0.005     0.002
    0.        0.004     0.        0.012     0.        0.005     0.01
    0.004     0.011     0.01      0.005     0.004     0.004     0.012
    0.01      0.004     0.012     0.003     0.004     0.        0.004
    0.02      0.011     0.003     0.003     0.007     0.008     0.005
    0.008     0.012     0.        0.        0.006     0.001     0.004
    0.011     0.023     0.023     0.009     0.003     0.011     0.01
    0.006     0.021     0.        0.004     0.003     0.005     0.002
          column 15
     0.001
    0.006
    0.01
    0.008
    0.009
    0.007
    0.01
    0.005
    0.001
    0.007
    0.007
 Les valeurs prises par X sont
         column 1 to 9
    1.     1.3     1.6     1.9     2.2     2.5     2.8     3.1     3.4
         column 10 to 11
    3.7     4.
Les valeurs prises par Y sont
         column 1 to 9
 - 4.  - 3.3  - 2.6  - 1.9  - 1.2  - 0.5     0.2     0.9     1.6
         column 10 to 15
    2.3     3.     3.7     4.4     5.1     5.8
La loi de X est
         column 1 to 7
    0.078     0.085     0.078     0.067     0.08     0.088     0.079
         column  8 to 11
     0.14     0.075     0.144     0.086
La loi de Y est
         column 1 to 7
     0.048     0.075     0.054     0.08     0.065     0.09     0.056
```

```
         column  8 to 14
    0.069    0.1    0.067    0.051    0.043    0.062    0.069
         column 15
    0.071
 L'esperance de X est
    2.624
 L'esperance de Y est
    0.881
 La variance de X est
    0.896
 La covariance de X et Y est
    0.039
 La variance de X est
    0.896
```

Remarque : $\mathrm{cov}(X,X) = \mathrm{E}((X - m_X)(X - m_X))$, soit $\mathrm{cov}(X,X) = \mathrm{var}(X)$.

Si on veut calculer la variance de X de cette manière, il faut calculer $P_{X,X}$. Comme $P(X = x_i, X = x_j)$ vaut 0 si $x_i \neq x_j$, $P(X = x_i)$ sinon, $P_{X,X}$ est la matrice diagonale dont les éléments diagonaux sont les éléments de P_X (commande `diag(PX)`). Ceci fait l'objet des 3 dernières lignes du script.



6.24 [***] Ranger n nombres donnés dans l'ordre croissant (tri à bulle)

Soit une suite de nombres $A = [a_1, \ldots, a_n]$ à ranger dans l'ordre croissant. On appellera *Aneuf* la suite obtenue. Dans le tri à bulles, on fait une suite de passages sur A. Au premier passage, on considère la paire (a_1, a_2) que l'on met dans l'ordre croissant, ce qui modifie A, puis (a_2, a_3), *etc.* À la fin du premier passage, le dernier terme de *Aneuf* est placé. On continue ces passages qui placent définitivement l'avant dernier terme de *Aneuf*, puis le précédent, *etc.* Après le $(n-1)^{\text{ème}}$ passage, *Aneuf* est obtenu.

On demande une fonction *scilab* qui range A dans l'ordre croissant par le tri à bulles.

Mots-clefs : « `tic();opération;toc()` », `tirage_entier`, `taille`, boucles `pour` imbriquées.

On peut utiliser la fonction suivante :

```
function Aneuf=TriBulles(A)
    B=A;
    n=taille(A);
    for k=1:(n-1)
        for j=1:(n-k)
            B([j,j+1])=[min(B(j),B(j+1)),max(B(j),B(j+1))];
        end
    end
    Aneuf=B;
//afficher(A);
//afficher(Aneuf);
endfunction
```

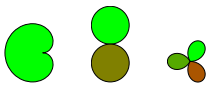
Ce script tient compte des remarques suivantes : à la fin du $(k-1)^{\text{ème}}$ passage, les $k-1$ derniers termes de *Aneuf* sont placés. Pour le $k^{\text{ème}}$ passage, il suffit donc d'examiner les paires (a_1, a_2), ..., (a_{n-k-1}, a_{n-k}). Et bien sûr, $k-1$ passages suffisent. Voici une application de cette fonction.

```
-->exec('Chemin du script', -1)
-->A=tirage_reel(10000,0,1);
-->tic();Aneuf=TriBulles(A);temps=toc();
-->temps
 temps  =
     391.183
 -->A(1:5)
 ans  =
        column 1 to 2
    0.8147236919031    0.1354770041071
        column 3 to 4
    0.9057919341139    0.8350085897837
        column 5
    0.1269868118688
-->Aneuf(1:5)
 ans  =
        column 1 to 2
    0.0000121421181    0.0002310688142
        column 3 to 4
    0.0003260960802    0.0003414633684
        column 5
    0.0005223751068
```

```
-->A(9996:10000)
 ans  =
         column 1 to 2
     0.1593243696261     0.9299520466011
         column 3 to 4
     0.4512528912164     0.2819604331162
         column 5
     0.9601144108456
-->Aneuf(9996:10000)
 ans  =
         column 1 to 2
     0.9994916191790     0.9997731263284
         column 3 to 4
     0.9997925662901     0.9998612964991
         column 5
     0.9999415774364
-->
```

Ces résultats font apparaître les 5 premiers (et les 5 derniers) termes de `A` et de `Aneuf`. Le tri seul a pris 391.183 secondes, ce qui est abominablement long. Il aurait été pratiquement instantané avec la commande `tri` de *scilab*.

Autres solutions : Voir les exercices 6.25, 8.9 et 8.10. Ils proposent d'autres tris, tous plus rapides que le tri à bulles, qui n'est pas un bon tri.



6.25 [****] Ranger n nombres donnés dans l'ordre croissant

Soit une suite d'au moins 3 nombres `A=[a_1,…,a_n]` à ranger dans l'ordre croissant. On appellera `Atri` la suite obtenue. Si l'on ne considère que les 2 premiers termes de la suite, `Atri=[min(A([1:2])),max(A([1:2]))]`. Si on tient compte du terme suivant, on va l'introduire dans `Atri` comme suit : on écrit d'abord les termes de `Atri` qui lui sont strictement inférieurs, puis ce nombre, puis les termes de `Atri` qui lui sont supérieurs ou égaux. Ceci nous fournit la nouvelles valeur de `Atri`. Et ainsi de suite. On demande une fonction *scilab* qui range `A` dans l'ordre croissant par ce procédé.

Mots-clefs : « `tic();opération;toc()` », `tirage_reel`, `taille`, `find`, boucle `pour`.

On peut utiliser la fonction suivante :

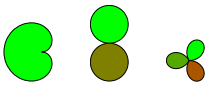
```
function Bof=Trier3(A)// A est une suite d'au moins 3 nombres
    t=taille(A);
    Atri=[min(A([1,2])),max(A([1,2]))];
    for i=3:t
        B=Atri;
        C=find(B<A(i));
        B(C)=[];
        D=find(Atri>=A(i));
        Atri(D)=[];
        Atri=[Atri,A(i),B];
    end
    Bof=Atri;
endfunction
```

Voici deux applications de cette fonction (tri de 10000 nombres entre 0 et 1, puis tri de 100000 nombres également entre 0 et 1 ; on a affiché pour chacun de ces deux tris le temps mis, les 10 premiers et les 10 derniers nombres de `A` et de `Atri`).

```
-->clear;
-->exec('Chemin de Bof', -1)
-->A=tirage_reel(10000,0,1);
-->tic();Atri=Trier3(A);temps=toc();afficher(temps);afficher(Atri(1:10));
afficher(Atri(9990:10000));
    5.092

          column 1 to 5
    0.0000110080000    0.0001033612061    0.0002221900504    0.0003911894746
0.0008971169591
           column 6 to 10
    0.0009051819798    0.0010192452464    0.0010214892682    0.0011630158406
0.0011716762092

          column 1 to 5
    0.9992581047118    0.9993214600254    0.9993323672097    0.9994070082903
0.9995183185674
          column 6 to 10
    0.9996201740578    0.9996497405227    0.9996856176294    0.9997958824970
0.9999444757123
```

```
          column 11
    0.9999806743581
-->A=tirage_reel(100000,0,1);
-->tic(); Atri=Trier3(A); temps=toc(); afficher(temps); afficher(Atri(1:10));
afficher(Atri(99990:100000));
    602.528

          column 1 to 5
    0.0000210860744    0.0000264451373    0.0000351499766    0.0000357083045
0.0000372761860
          column  6 to 10
    0.0000668293796    0.0000717386138    0.0000732627232    0.0000770098995
0.0000860642176

          column 1 to 5
     0.9998367871158    0.9998749606311    0.9998786845244    0.9998902815860
0.9998920490034
           column  6 to 10
     0.9998978190124    0.9999217658769    0.9999401415698    0.9999512080103
0.9999596350826
          column 11
     0.9999762303196
-->
```

Autres solutions : Voir les exercices 6.24, 8.9 et 8.10. Ce tri est un mauvais tri, très lent. La commande `trier` de *scilab* est de loin plus rapide.



6.26 [**] Trier un tableau suivant l'une de ses lignes

Voici la liste des fréquences des 26 lettres de l'alphabet dans la langue française :
7.68, 0.80, 3.32, 3.60, 17.76, 1.06, 1.10, 0.64, 7.23, 0.19, 0.00, 5.89, 2.72, 7.61, 5.34, 3.24, 1.34, 6.81, 8.23, 7.30, 6.05, 1.27, 0.00, 0.54, 0.21, 0.07.
Le problème est de ranger ces lettres par ordre des fréquences décroissantes.

Plus généralement, étant donné une matrice numérique M à n lignes et m colonnes et un entier j, $1 \leqslant j \leqslant n$, ranger les colonnes de `M` de manière que la ligne j soit rangée dans l'ordre décroissant.

Mots-clefs : fonction *scilab*, boucles `pour`, manipulations élémentaires de tableaux, commandes `min`, `max`, `taille`, `find`, `exec`, `asciimat`.

Commençons par rentrer les données dans un tableau. Nous optons pour un tableau numérique en reportant non pas les lettres de l'alphabet mais leurs codes ASCII [14]. Nous transformons ainsi la liste A,..., Z en le vecteur 65 : 90.

Listing 6.4 – Les lettres et leurs fréquences

```
let=[65:90];
Freq=[7.68,0.80,3.32,3.60,17.76,1.06,1.10,0.64,7.23,0.19,0.00,5.89,2.72,7.61,
5.34,3.24,1.34,6.81,8.23,7.30,6.05,1.27,0.00,0.54,0.21,0.07];
T=[let;Freq];
```

On voit que la cinquième colonne[8] devra prendre la place de la première, que la dix-neuvième[9] devra venir en deuxième position, *etc.* Le mieux est de traiter le problème général : étant donné une matrice numérique M à n lignes et m colonnes et un entier j, $1 \leqslant j \leqslant n$, de ranger les colonnes de `M` de manière que la ligne j soit rangée dans l'ordre décroissant. La fonction `TriCol` ci-dessous réalise cette tâche :

Listing 6.5 – Fonction TriCol

```
//Le fonction TriCol ordonne les colonnes d'une matrice numerique de maniere
//qu'une ligne donnee soit rangee dans l'ordre decroissant.
function MTriee=TriCol(M,ligne)
    //M=Matrice dont les colonnes doivent etre ordonnees.
    //ligne=indice de la ligne a ordonner.
    MTriee=[];
    A=M(ligne,:);//Ligne a ranger.
    minA=min(A);
    t=taille(A);
    for j=1:t
        maxA=max(A);
        k=max(find(A==maxA));//maxA peut apparaitre plusieurs fois.On choisit
        //arbitrairement celui qui a le rang le plus eleve.
        MTriee=[MTriee,M(:,k)];//On place la colonne selectionnee a la suite
        //des autres.
        A(k)=minA-1;// Cette colonne n'interviendra plus. On aurait pu
        //l'effacer.
        j=j+1;
    end
endfunction
```

8. colonne de code ASCII 69, soit E
9. colonne de code ASCII 83, soit S

Ci-dessous, nous avons regroupé toutes les commandes, y compris le chargement de la fonction `TriCol` dans un seul script appelé `TriColonnes` ;

Listing 6.6 – Script TriColonnes

```
let=[65:90];
Freq=[7.68,0.80,3.32,3.60,17.76,1.06,1.10,0.64,7.23,0.19,0.00,5.89,2.72,7.61,5.34,3.
T=[let;Freq];
exec('Chemin de la fonction TriCol.sci', -1);
TTriee=TriCol(T,2);
LetTriees=TTriee(1,:);
alphTrie=asciimat(LetTriees);
afficher('Voici l''alphabet range dans l''ordre des frequences decroissantes :');
afficher(alphTrie)
```

qui a conduit au résultat suivant :

Listing 6.7 – Résultat

```
-->clear
-->exec('Chemin du script TriColonnes.sce', -1)
 Voici l'alphabet range dans l'ordre des frequences decroissantes :
 ESANTIRULODCPMQVGFBHXYJZWK
-->
```

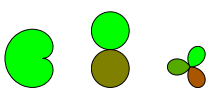
Variante de la fonction `TriCol`

Dans la fonction `TriCol`, les termes déjà traités étaient rendus inopérants par la commande `A(k)=minA-1`. Dans `TriCol2`, on les efface à l'aide de la commande `A2(k)=[]`. La boucle `pour` a été remplacée par une boucle `tant que`.

Listing 6.8 – Fonction `tricol`

```
//Le fonction TriCol ordonne les colonnes d'une matrice numerique de maniere
//qu'une ligne donnee soit rangee dans l'ordre decroissant.
function MTriee=TriCol2(M,ligne)
    //M=Matrice dont les colonnes doivent etre ordonnees.
    //ligne=indice de la ligne a ordonner.
    MTriee=[];
    A2=M(ligne,:);//Ligne a ranger.
    while taille(A2)>0
        k=max(find(A2==max(A2)));//maxA peut apparaitre plusieurs fois.
        //On choisit arbitrairement celui qui a le rang le plus eleve.
        MTriee=[MTriee,M(:,k)];
        A2(k)=[];
    end
endfunction
```

Pour appliquer cette fonction, il suffit de remplacer `TriCol` par `TriCol2` dans le script 6.6.



Chapitre 7

Instructions conditionnelles

Syntaxe de l'instruction conditionnelle `if-then-else`

✓ Instruction conditionnelle sans alternative

```
if conditions then
instructions ;
end
```

✓ Instruction conditionnelle avec alternative simple

```
if conditions then
instructions ;
else
instructions ;
end
```

✓ Instruction conditionnelle avec alternative complexe

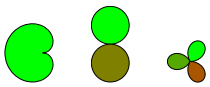
```
if conditions then
instructions ;
elseif conditions then
instructions ;
....
else
instructions ;
end
```

Liste des exercices :

énoncé n° 7.8 : [**] Calcul et représentation graphique de fréquences (exercice type du Calcul des probabilités).

énoncé n° 7.9 : [**] Le point M est-il à l'intérieur du triangle ABC ?

énoncé n° 7.10 : [**] Ce triangle est-il isocèle ?



7.1 [*] Fonction valeur absolue

1 - Programmer une fonction-scilab qui donne la valeur absolue de tout nombre réel
- à l'aide d'une instruction conditionnelle avec alternative,
- à l'aide d'une instruction conditionnelle sans alternative.

2 - En déduire la valeur absolue de -0.87 et de $\sqrt{3}$.

Mots-clefs : instruction conditionnelle, commandes `sqrt`, `abs`.

La commande `abs` de *scilab* a pour argument un vecteur ou liste de nombres réels et retourne la liste des valeurs absolues de ces nombres, par exemple :

```
-->abs([-0.87,sqrt(3)])
 ans  =
    0.87    1.7320508075689
-->
```

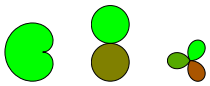
1 - La fonction valeur absolue se programme avec une instruction conditionnelle sans alternative (fonction `valabs1` ci-dessous). On peut aussi l'écrire avec une instruction conditionnelle avec une alternative tout à fait inutile (fonction `valabs2`). Ces fonctions admettent comme argument un nombre réel et retourne sa valeur absolue. Elles font donc moins bien que la commande `abs`.

```
function y = valabs1(x)
  y=x;
  if x<0 then
    y=-x;// pas d'alternative
  end
endfunction
function y = valabs2(x)
  if x<0 then
    y=-x;
  else // alternative
    y=x;
  end
endfunction
```

2 - Chargeons ces fonctions dans la console et appliquons les fonctions précédentes au calcul des valeurs absolues de -0.87 et de $\sqrt{3}$. On obtient :

```
-->exec('Chemin du fichier', -1)
-->a1=valabs1(-0.87);a2=valabs2(-0.87);
-->afficher(a1,a2);
    0.87
    0.87
-->b1=valabs1(sqrt(3));b2=valabs2(sqrt(3));
-->afficher(b1,b2);
    1.7320508075689
    1.7320508075689
-->
```

On aurait pu programmer la fonction `valabs1` de manière à ce qu'elle admette une liste de nombres comme argument.



7.2 [*] Calculer le maximum de 2 ou 3 nombres donnés

Écrire une fonction *scilab* qui retourne le maximum de 2 nombres `x` et `y`, puis une fonction *scilab* qui retourne le maximum de 3 nombres `x`, `y` et `z` donnés.

Mots-clefs : instruction conditionnelle `if`.

Les fonctions *scilab* `max2` et `max3` du script ci-dessous conviennent. Elles retournent `m` qui est le maximum de `x` et `y` dans le premier cas, le maximum de `x`, `y` et `z` dans le second. On utilise une condition `if` sans alternative.

```
function m=max2(x,y)
    m=x;
    if y>m then
        m=y;
    end
endfunction

function m=max3(x,y,z)
    m=max2(x,y);
    if z>m then
        m=z
    end
endfunction
```

Application numérique :

```
-->exec('Chemin du script', -1)
-->x=sin(%pi/5);y=exp(-0.15);z=-0.590;
-->m=max2(x,y)
 m  =
    0.8607080
-->m=max3(x,y,z)
 m  =
    0.8607080
-->
```

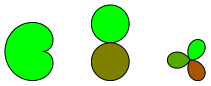
Voici le même problème traité avec *Python* 3 (cf. [3], Exercice 1, p.17). On utilise une condition `if` avec une seule alternative pour la première fonction ; la seconde utilise la première.

```
def max2(x,y):
    if y>x: return y
    else: return x

def max3(x,y,z):# utilise la fonction max2.
    return max2(x,max2(y,z))
```

Application numérique :

```
========== RESTART: /Users/raymondmoche/Desktop/cbcc_1.py ==========
>>> x,y,z=-1/2,0.7,-4
>>> max2(x,y)
0.7
```

```
>>> max3(x,y,z)
0.7
>>>
```



7.3 [*] Ranger 2 nombres dans l'ordre croissant

Écrire une fonction *scilab* qui, étant donné deux nombres réels, les retourne rangés dans l'ordre croissant.

Mots-clefs : instruction conditionnelle, `min`, `max`, `trier`.

La commande `trier` de *scilab* permet de trier très vite, dans l'ordre croissant, de très longues listes de nombres. C'est une commande opaque, c'est à dire qu'on ne sait pas comment elle effectue ces tris. Elle est très efficace. Ci-dessous, nous allons programmer explicitement une fonction qui permet de ranger deux nombres dans l'ordre croissant.

Solution n° 1, avec une instruction conditionnelle :

```
function A=ranger(x,y);
    if x>y then// instruction conditionnelle sans alternative.
        z=x;
        x=y;
        y=z;
    end;
    A=[x,y];
endfunction
```

On charge la fonction dans la console. Si par exemple on désire ranger dans l'ordre croissant `%pi/3` et `sqrt(2)` que l'on veut afficher avec 10 décimales (car *scilab* effectuera automatiquement les calculs), on tapera directement dans la console :

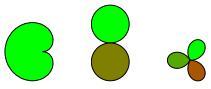
```
-->format('v',13);x=%pi/3;y=sqrt(2);ranger(x,y)
 ans  =
    1.0471975512    1.4142135624
-->
```

Solution n° 2 : Si on s'autorise les commandes `min` et `max`, il y a une solution explicite évidente. Pour ranger 3 nombres dans l'ordre croissant, on pourra biaiser de même (voir l'exercice 7.4).

```
-->format('v',13);x=%pi/3;y=sqrt(2);B=[min(x,y),max(x,y)]
 B  =
    1.0471975512    1.4142135624
-->
```

Solution n° 3 : Utiliser la commande `trier` de *scilab* :

```
-->x=%pi/3;y=sqrt(2);trier([x,y])
 ans  =
    1.0471975512    1.4142135624
-->
```



7.4 [*] Ranger 3 nombres dans l'ordre croissant

Ranger trois nombres réels donnés dans l'ordre croissant.

Mots-clefs : instruction conditionnelle, `min`, `max`, `find`, `sum`, `trier`.

Les données sont ici fabriquées par la commande `tirage_reel(3,a,b)`, $a < b$, qui produit 3 nombres réels entre a et b. Ranger 3 nombres dans l'ordre croissant est aussi traité dans l'exercice 8.8.

Solution n° 1 : Résoudre le problème posé à l'aide d'instructions conditionnelles est évidemment maladroit car cela revient à passer en revue tous les cas possibles. Voici un tel script :

```
L=input('L=');
R=L;
if (L(1)<=L(2))&(L(2)>L(3)) then
    if L(3)<=L(1) then
        R=[L(3),L(1),L(2)];
    else
        R=[L(1),L(3),L(2)];
    end
elseif (L(1)>L(2))&(L(2)<=L(3)) then
    if L(3)<=L(1) then
        R=[L(2),L(3),L(1)];
    else
        R=[L(2),L(1),L(3)];
    end
elseif (L(1)>L(2))&(L(2)>L(3)) then
    if L(1)>L(3) then
        R=[L(3),L(2),L(1)];
    end
end
afficher(R,L)
```

C'est pénible à écrire et d'exécution lente. Cette solution ne peut pas se développer pour ranger dans l'ordre croissant plus de 3 nombres. En voici un exemple d'application :

```
-->clear
-->exec('Chemin du script', -1)
L=tirage_reel(3,-1.97,3.44)
    2.4376551731955   - 1.2370694077807     2.9303343635565
  - 1.2370694077807     2.4376551731955     2.9303343635565
-->
```

Solution n° 2 : Comme on l'a signalé à l'exercice 7.3, il y a une solution explicite. Dans le script qui suit, on a de plus demandé un affichage des nombres avec 2 décimales :

```
format('v',5)
A=tirage_reel(3,-1,2);
B(1,1)=min(A);
B(1,3)=max(A);
B(1,2)=sum(A)-(B(1)+B(3));
afficher(A,B);
```

Voici une exécution de ce script :

```
-->exec('Chemin du script', -1)
  - 0.34    0.90    1.74
    1.74  - 0.34    0.90
 -->
```

On rappelle que `afficher(A,B)` affiche `B` puis `A`. Cette solution ne conduit pas non plus à une solution qui permettrait de ranger au moins 4 nombres dans l'ordre croissant.

Solution n° 3 : il s'agit de la solution n° 1 sans calcul. En effet, dans la solution n° 1, le terme médian a été calculé. Il peut en résulter une erreur infime, ce que l'on évite ici :

```
A=tirage_reel(3,-1,2);
B=A;
I1=find(B==min(B));
I2=find(B==max(B));
B(1,[min(I1),max(I2)])=[];
C=[min(A),B,max(A)];
afficher(C,A);
```

Voici une exécution de ce script avec les réglages par défaut de l'affichage :

```
-->clear
-->exec('Chemin du script', -1)
    1.74  - 0.34    0.90
  - 0.34    0.90    1.74
-->
```

L'affichage est cette fois dans le bon ordre : `A` puis `B`.

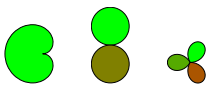
Solution n° 4 : Voir le tri à bulles, 6.24.

Solution n° 5 : Voir l'exercice 6.25.

Solution n° 6 : Voir l'exercice 8.9.

Solution n° 7 : Voir l'exercice 8.10.

Solution n° 5 : Utiliser la commande `trier` de *scilab*. C'est évidemment la bonne solution.



7.5 [**] Sauts de kangourou

Un kangourou fait habituellement des bonds de longueur aléatoire comprise entre 0 et 9 mètres.

1 - Combien de sauts devra-t-il faire pour parcourir 2000 mètres ?

2 - Fabriquer une fonction-scilab qui à toute distance d exprimée en mètres associe le nombre aléatoire N de sauts nécessaires pour la parcourir.

3 - Calculer le nombre moyen de sauts effectués par le kangourou quand il parcourt T fois la distance d.

Mots-clefs : boucles `tant que` et `pour`, `tirage_reel( , , )`, `moyenne`, `afficher`, `tic()`, `toc()`.

1 - On remarquera que l'énoncé est incorrect car il est incomplet. Il est en fait sous-entendu que la longueur de chaque bond est la valeur prise par une variable aléatoire qui suit la loi uniforme entre 0 et 9. Il est aussi sous-entendu, comme d'habitude, que si on considère des sauts successifs, les variables aléatoires qui leur sont associées sont indépendantes.

```
D=0;// distance parcourue avant le premier saut.
N=0;// nombre de sauts effectues avant le premier saut.
while D<2000 then
  D=D+tirage_reel(1,0,9);
  N=N+1;
end
afficher('Le nombre de sauts necessaires pour parcourir 2000 m a ete '
+string(N)+'. La distance reelle parcourue a ete ' +string(D)+' metres.');
```

Exécution du script :

```
-->exec('Chemin du script', -1)
 Le nombre de sauts necessaires pour parcourir 2000 m
       a ete 459. La distance reelle parcourue a ete 2003.3149584748 metres.
-->
```

2 - Compte tenu du script précédent, on choisira évidemment comme fonction la fonction `kangourou` suivante :

```
function N=kangourou(d)
    D=0;
    N=0;
    while D<d then
        D=D+tirage_reel(1,0,9);
        N=N+1;
    end;
endfunction;
```

3 - Utilisons le script suivant (qui produit aussi le temps de calcul) :

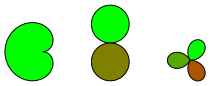
```
d=input('La distance a parcourir est egale a ');
T=input('Le nombre de trajets est ');
tic();
NSauts=[];
for j=1:T
  NSauts=[NSauts,kangourou(d)];
```

```
end;
Moyenne=moyenne(NSauts);
toc();
afficher('Le nombre moyen de sauts quand le kangourou parcourt '+string(T)+
' fois au moins '+string(d)+' metres est '+string(Moyenne)+'.');
afficher('Duree du calcul : '+string(toc()));
```

Exécution du script dans le cas de 600 parcours de 2500 m :

```
-->clear
-->exec('Chemin de la fonction kangourou', -1)
-->exec('Chemin du script', -1)
La distance a parcourir est egale a 2500
Le nombre de trajets est 600
 Le nombre moyen de sauts quand le kangourou parcourt 600 fois au
       moins 2500 metres est 556.25166666667.
 Duree du calcul : 15.79
-->
```

On pourrait évidemment prolonger cet exercice par une illustration de la loi des grands nombres.



7.6 [**] Addition en base 2

Soit deux entiers n, $m \geqslant 0$ notés en base 2 sous la forme $[a_0, \ldots, a_{k-1}, a_k]$ et $[b_0, \ldots, b_{k-1}, b_l]$.
Cela signifie que ce sont deux suites de 0 et de 1 qui vérifient

$$n = a_0 \cdot 2^0 + a_1 \cdot 2^1 + \cdots + a_k \cdot 2^k \text{ et } m = b_0 \cdot 2^0 + b_1 \cdot 2^1 + \cdots + b_l \cdot 2^l$$

Le problème posé est de calculer leur somme en travaillant dans la base 2.

Mots-clefs : boucle `pour`, instruction conditionnelle `if (conditions) then (instructions) elseif (conditions) then (instructions) elseif (conditions) then (instructions) else (instructions) end`, opérateurs logiques `&` (et), `|` (ou).

Il n'y a pas d'inconvénient à supposer que ces listes sont de même longueur, parce que l'on peut toujours ajouter des 0 à la liste la plus courte, si nécessaire. Par exemple, si on veut additionner 5 et 3 en base 2, on écrira 5 sous la forme $[1, 0, 1]$ et 3 sous la forme $[1, 1, 0]$ au lieu de $[1, 1]$.

Rappelons la table d'addition en base 2 :

+	0	1
0	0	1
1	1	10

Posons

$$n + m = s_0 \cdot 2^0 + s_1 \cdot 2^1 + \cdots + s_{k+1} \cdot 2^{k+1}$$

Au début du calcul, il n'y a pas de retenue. On conviendra qu'il y en a une qui est $r_0 = 0$. On calcule ensuite le coefficient s_i de 2^i dans la somme $n + m$ et la retenue à venir r_{i+1} successivement pour $i = 1, \ldots, k$ en tenant compte de la retenue r_i. Les différents résultats possibles sont résumés dans le tableau suivant :

Nbre de 1	s_{i+1}	r_{i+1}
0	0	0
1	1	0
2	1	0
3	1	1

Cela permet de programmer la somme de 2 entiers en base 2, donnés sous la forme de listes de 0 et de 1. La fonction proposée ci-dessous est appelée `somme2`. Elle est téléchargeable sur le site. Les premières instructions ont pour but d'ajouter si nécessaire des zéros à droite de la liste la plus courte afin qu'elles aient la même longueur. Le résultat `S` sera de longueur minimum, c'est à dire sans zéros à droite inutiles.

```
function S=somme2(n,m)// N et M sont donnes sous la forme de 2 listes
de 0 et de 1 : N=[a0,...,a_k] et M=[b0,...,bl].
    k=taille(N)-1;l=taille(M)-1;
    N=[N,zeros(1,l-k)];M=[M,zeros(1,k-l)];// egalisation des longueurs.
    k=max(k,l);// N et M ont k comme longueur commune.
// Debut de l'addition en base 2.
    r=0;//La premiere retenue est nulle.
    S=[];// Pour le stockage de la somme.
    for i=0:k
        a=N(i+1);
        b=M(i+1);
        c=a+b+r;
        if c==0 then
            s=0;r=0;
        elseif c==1 then
            s=1;r=0;
```

```
        elseif c==2 then
            s=0;r=1;
        else
            s=1;r=1;
        end
        S=[S,s];
    end
    if r==1 then
        S=[S,r];
    end
endfunction
```

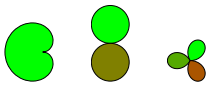
Utilisons cette fonction à titre d'exemple :

```
-->clear
-->exec('Chemin de la fonction somme2', -1)
-->N=[0,1,1,0,1,1,1,0,0,0,1];M=[1,1,0,0,0,0,1,0,1,0,1];
-->S=somme2(N,M)
 S  =
          column 1 to 10
    1.    0.    0.    1.    1.    1.    0.    1.    1.    0.
          column 11 to 12
    0.    1.
-->n=sum(N.*2.^(0:taille(N)-1))
 n  =
    1142.
-->m=sum(M.*2.^(0:taille(M)-1))
 m  =
    1347.
-->s=sum(S.*2.^(0:taille(S)-1))
 s  =
    2489.
-->
```

Cela veut dire que

$$[0,1,1,0,1,1,1,0,0,0,1] + [1,1,0,0,0,0,1,0,1,0,1] = [1,0,0,1,1,1,0,1,1,0,0,1]$$

Le passage de l'écriture en base 2 à l'écriture en base 10 a fait l'objet de l'exercice 3.3. On vérifie de tête, en base 10, que le résultat est exact !



7.7 [**] L'année est-elle bissextile ?

1 - L'année n est-elle bissextile ?

2 - Soit n et m deux millésimes ($n \leqslant m$). Combien y a-t-il d'années bissextiles de l'année n à l'année m (années n et m comprises) ?

Mots-clefs : fonction-scilab, instructions conditionnelles, compteur, `%f` (faux), `%t` (vrai), `reste`, opérateurs logiques `|` (ou), `&` (et), `==` (=).

L'intérêt de cet exercice est surtout de prendre la négation d'une proposition (logique).

1 - On définit ci-dessous une fonction `estbissextile` dont l'argument est n, qui retourne `%t` ou `%f` suivant que l'année n est bissextile ou non. Ce programme traduit la définition des années bissextiles :

$$n \text{ est bissextile si et seulement si } (r = 0 \text{ et } s > 0) \text{ ou } (t = 0)$$

où r, s et t désignent successivement les restes dans la division euclidienne de n par 4, 100 et 400 (c'est la définition des années bissextiles).

```
function [truefalse]=estbissextile(n)
  r=reste(n,4);
  s=reste(n,100);
  t=reste(n,400);
  truefalse=%f;
  if (((r==0)&(s>0))|(t==0)) then
      truefalse=%t;
  end;
endfunction;
```

Utilisons cette fonction pour savoir si les 2047, 2048, 2100 et 2400 sont bissextiles :

```
-->exec('Chemin de la fonction estellebissextile', -1)
-->estbissextile(2047)
 ans  =
  F
-->estbissextile(2048)
 ans  =
  T
-->estbissextile(2100)
 ans  =
  F
-->estbissextile(2400)
 ans  =
  T
-->
```

En prenant la négation de la définition des années bissextiles :

$$n \text{ n'est pas bissextile si et seulement si } (r > 0 \text{ ou } s = 0) \text{ et } (t = 0)$$

on peut programmer une fonction `estvraimentbissextile` qui a la même entrée et la même sortie que `estbissextile` :

```
function [truefalse]=estvraimentbissextile(n)// fonction identique a la
//precedente.
  r=reste(n,4);
```

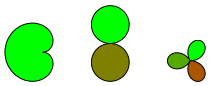
```
  s=reste(n,100);
  t=reste(n,400);
  truefalse=%t;
  if ((r>0)|((s==0)&(t>0))) then
    truefalse=%f;
  end;
endfunction;
```

2 - Pour traiter la seconde question, il suffit d'un compteur, ce qui donne le script suivant :

```
n=input('n=');
m=input('m=');// n<=m.
C=0;// Initialisation du compteur
for i=n:m
    if estbissextile(i)==%t then
        C=C+1;
    end
end
afficher('Le nombre d''annees bissextiles recherche est '+string(C));
```

Exemple : trouver le nombre d'années bissextiles de 2014 à 2500.

```
-->clear
-->exec('Chemin de la fonction estbissextile', -1)
-->exec('Chemin du script', -1)
n=2014
m=2500
 Le nombre d'annees bissextiles recherche est 118
-->
```



7.8 [**] Calcul et représentation graphique de fréquences (exercice type du Calcul des probabilités)

On tire 10000 nombres au hasard dans l'intervalle $[0,1]$. À chaque tirage, on se demande si l'événement

$$\text{A : « le nombre tiré au hasard appartient à l'intervalle } [\frac{1}{7}\,,\,\frac{5}{6}] \text{ »}$$

est réalisé.

1 - Combien vaut la probabilité p de A ?

2 - Calculer la fréquence de réalisation de A après chaque tirage. Tracer le graphe des fréquences.

3 - Choisir la dernière fréquence calculée comme valeur approchée de p.

Mots-clefs : boucle `pour`, commandes `tirage_reel(p,a,b)`, `plot(X)`, `plot(X,Y,'r')`, `plot(X,Y,'r+')` et commandes analogues, `grand(m,n,'bin',N,p)`, compteur, nuage de points.

Ceci est un exercice standard sur l'*illustration par le calcul* de la loi des grands nombres du Calcul des Probabilités. Nous insistons sur les différentes représentations possibles, correctes ou non, de la suite des fréquences calculées. Dans la section **Pour aller plus loin**, nous montrons comment on peut raccourcir les algorithmes avec un peu plus de Calcul des probabilités.

1 - L'énoncé est clairement incorrect car il ne précise pas *selon quel hasard* les nombres sont tirés dans l'intervalle $[0,1]$. En fait, il est sous-entendu qu'ils sont tirés selon la loi uniforme sur $[0,1]$. Dans ces conditions, par définition de cette loi, la probabilité de l'événement A est la longueur de $[1/7\,,\,6/7]$, soit $p = 5/6 - 1/7 = 29/42$.

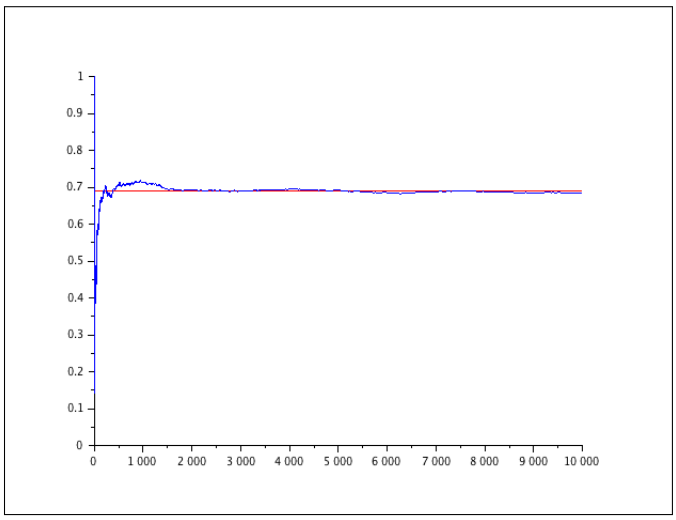
2 & 3 - On peut utiliser le script commenté ci-dessous.

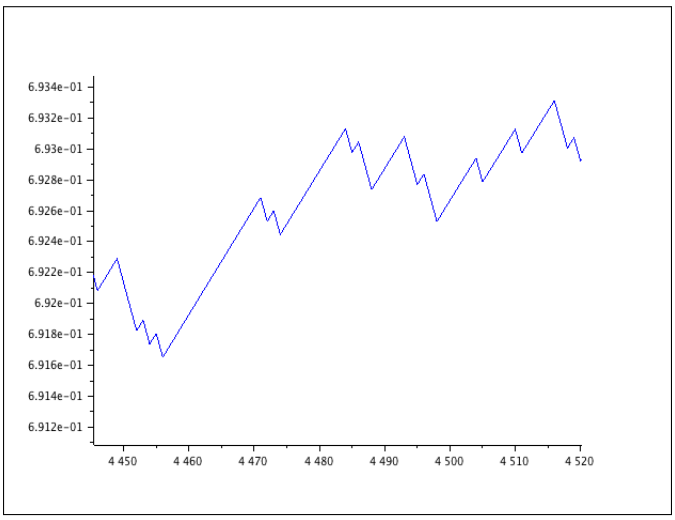
```
clear;
clf;
Tirages=tirage_reel(10000,0,1);// Liste de 10000 nombres tires
//au hasard suivant la loi uniforme sur [0,1].
p=5/6-1/7;
afficher('La valeur exacte de p est '+string(p));
C=0;// Initialisation du compteur des realisations de A.
Frequences=[];// Initialisation du vecteur des frequences.
for i=1:10000
    if (1/7<=Tirages(i))&(Tirages(i)<=5/6) then
        C=C+1;//Si A se realise, le compteur augmente de 1.
    end;
    Frequences=[Frequences,C/i];
end;
plot([0,10000],[p,p],'r');// Ligne horizontale d'ordonnee p, en rouge.
plot(Frequences);
afficher('La valeur approchee de p par les frequences est '
+string(Frequences(10000)));
```

On remarquera que l'appartenance de `Tirages(i)` à l'intervalle $[0,1]$ est décrite comme la conjonction (`&`) des conditions `1/7<=Tirages(i)` et `Tirages(i)<=5/6`. Voici ce que l'on obtient sur la console :

```
-->exec('Chemin du script', -1)
La valeur exacte de p est 0.6904761904762
La valeur approchee de p par les frequences est 0.6904
-->
```

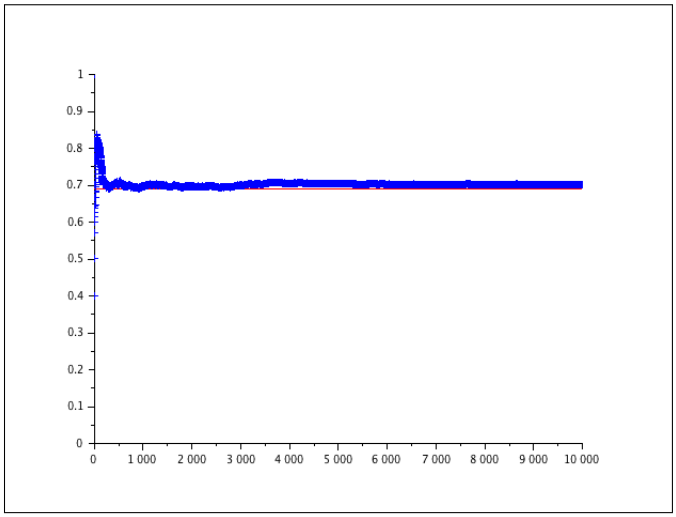
Le graphique correspondant est (à gauche) :

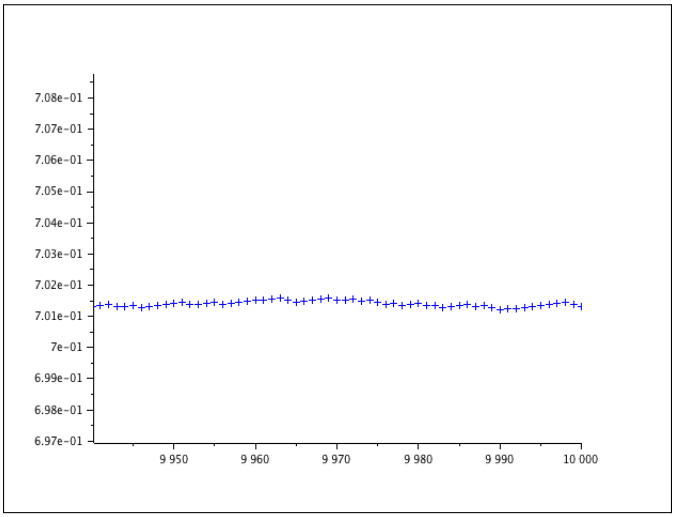




Ce graphique est parfois considéré comme satisfaisant par des professeurs. Mais un agrandissement (à droite) rappelle qu'il s'agit d'une ligne polygonale (la commande `plot` trace des lignes polygonales, voir l'aide en ligne). Or ce qu'on demande est *un nuage de points*, formé des points d'abscisse `i` et d'ordonnée `C/i`, pour `i` variant de 1 à 10000. Le graphique obtenu est donc mauvais.

Pour obtenir le nuage de points désiré, on peut remplacer dans le script précédent `plot(Frequences)` par la commande `plot(Frequences,'+b')`. Les points du nuage sont alors représentés par des `+` en bleu et ne sont pas joints par des segments, voir l'aide en ligne. Cela donne une grosse limace bleue qui ne fait pas penser à un nuage de points. On peut se convaincre que c'en est un en agrandissant plusieurs fois une portion du graphe, voir la figure de droite.



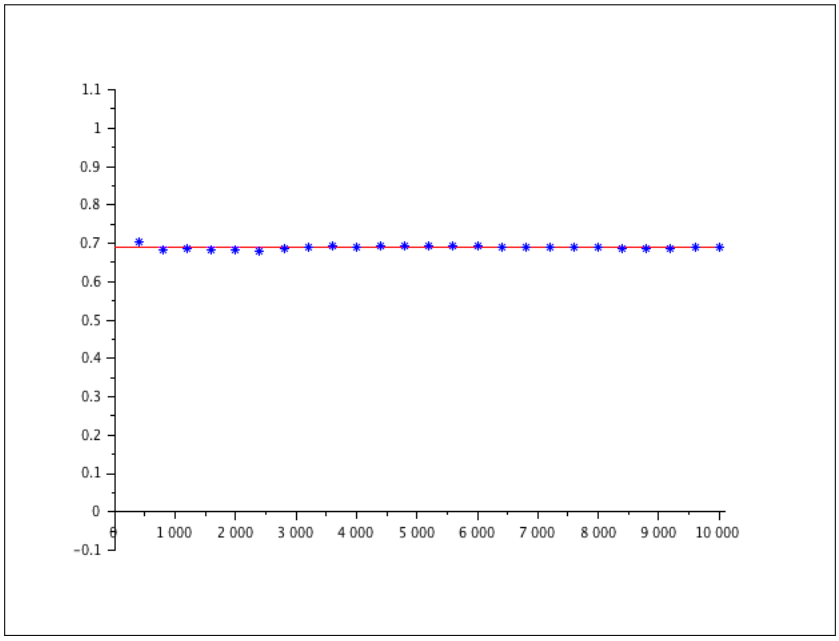


Finalement, la meilleure solution est sans doute de choisir les points que l'on va tracer, mais c'est plus difficile à expliquer aux élèves. Ci-dessous, on a tracé seulement les points d'abscisse 400, 800, ...,10000 à l'aide du script :

```
clear
clf
Tirages=tirage_reel(10000,0,1);
p=5/6-1/7;
afficher('La valeur exacte de p est '+string(p));
C=0;
Frequences=[];
```

```
for i=1:10000
    if (1/7<=Tirages(1,i))&(Tirages(1,i)<=5/6) then
        C=C+1;
    end;
    Frequences=[Frequences,C/i];
end;
afficher('La valeur approchee de p par les frequences est '
+string(Frequences(10000)));
abscisses=400:400:10000;
frequences=Frequences(abscisses);
a=gca();a.data_bounds=[-0.1,-0.1;10100,1.1];//reglage du graphe.
plot([0,10000],[p,p],'r');
plot(abscisses,frequences,'b*');
```

(nuage de $*$ bleus). On obtient le graphe suivant, qui est très satisfaisant :



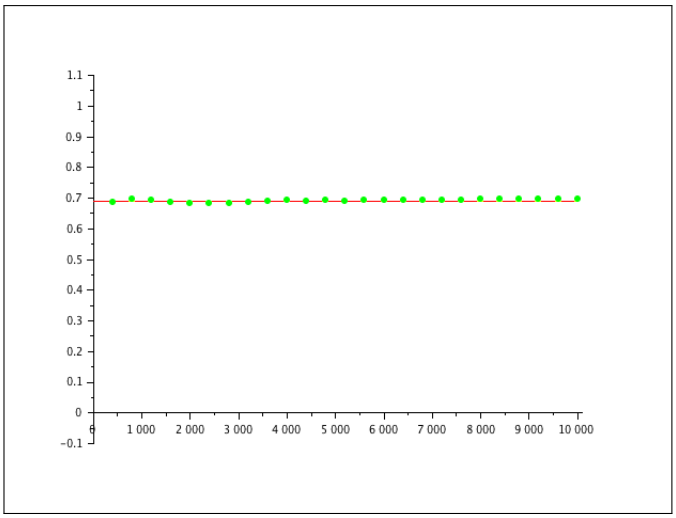
Pour aller plus loin :

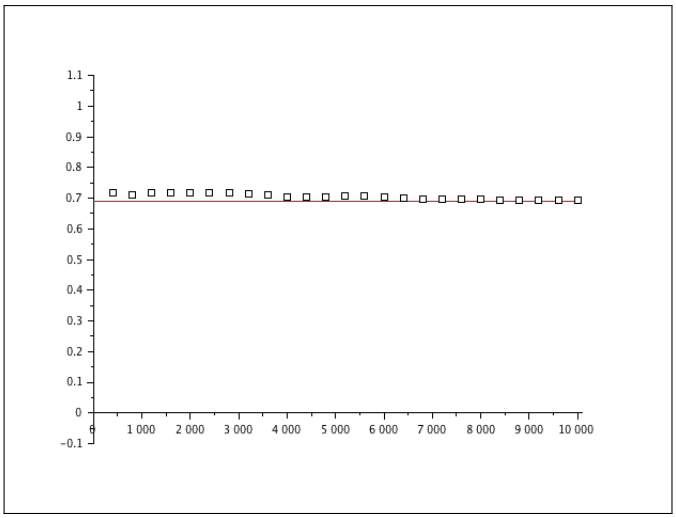
1 - On est ici dans le schéma de la répétition à l'identique d'une expérience aléatoire, les répétitions étant indépendantes les unes des autres. Plus précisément, on a fait intervenir 10000 variables aléatoires indépendantes et suivant la loi uniforme sur l'intervalle $[0\,,\,1]$. Ces variables aléatoires étant notées $X_1, \ldots, X_{10000}$, si pour $i = 1, \ldots, 10000$, on appelle Y_i la variables aléatoire qui vaut 1 si l'événement $\{1/7 \leqslant X_i \leqslant 5/6\}$ est réalisé, 0 sinon (Y_i s'appelle la fonction indicatrice de cet événement), les variables aléatoires $Y_1, \ldots, Y_{10000}$ sont indépendantes et suivent la même loi : elles prennent la valeur 1 avec la probabilité $p = 29/42$, la valeur 0 sinon. Comme on s'intéresse en réalité aux variables Y_i (plutôt qu'aux variables X_i), on peut simuler directement celles-ci et utiliser un script sans instruction conditionnelle.

```
clear;
clf;
Tirages=grand(1,10000,"bin",1,29/42);
p=29/42;
C=0;
Frequences=[];
for i=1:10000
    C=C+Tirages(i);
    Frequences=[Frequences,C/i];
```

```
end;
abscisses=400:400:10000;
frequences=Frequences(abscisses);
a=gca();a.data_bounds=[-0.1,-0.1;10100,1.1];//reglage du graphe.
plot([0,10000],[p,p],'r');
plot(abscisses,frequences,'g.');
```

(nuage de points verts). `grand(m,n,'bin',N,p)` est une commande de *scilab* qui fonctionne avec *scilab pour les lycées*. Elle produit une matrice à `m` lignes et `n` colonnes de nombres tirés selon la loi binomiale de taille `N` et de paramètre p. C'est la loi commune des variables aléatoires Y_i pour `N=1` et `p=29/42`. Évidemment, `m=1` et `n=10000`. On obtient le graphe de gauche :

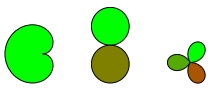




2 - En fait, les lois binomiales de taille 1 sont appelées habituellement lois de Bernoulli. Mais cela donne une nouvelle idée de simplification. Les sommes des paquets successifs de 400 variables aléatoires $Y_1, \ldots, Y_{10000}$ forment une famille de 25 variables aléatoires $Z_1, \ldots, Z_{25}$ indépendantes et suivant la loi binomiale de taille 400 et de paramètre `p`. Comme en fait on ne s'intéresse qu'à elles, on peut les simuler directement, selon le script suivant :

```
clear;
clf;
Tirages=grand(1,25,"bin",400,29/42);
p=29/42;
C=0;
Frequences=[];
for i=1:25
    C=C+Tirages(i);
    Frequences=[Frequences,C/(400*i)];
end;
abscisses=400:400:10000;
a=gca();a.data_bounds=[-0.1,-0.1;10100,1.1];//reglage du graphe.
plot([0,10000],[p,p],'r');
plot(abscisses,Frequences,'ks');
```

ce qui produit le graphe de droite ci-dessus (nuage de carrés noirs).



7.9 [**] Le point M est-il à l'intérieur du triangle ABC ?

Étant donné un triangle $A_1A_2A_3$ et un point M (par leurs coordonnées), fabriquer une fonction qui indique si M est à l'intérieur du triangle ou non.

Mots-clefs : fonction-scilab, instruction conditionnelle, « input », « exec ».

1 - Rappels : Soit A et B deux points distincts (on ne peut avoir en même temps $x_A = x_B$ et $y_A = y_B$). Posons

$$f(x, y) = (y - y_A)(x_B - x_A) - (y_B - y_A)(x - x_A)$$

L'équation de la droite $[AB]$ est $f(x, y) = 0$ (un point M de coordonnées (x, y) appartient à $[AB]$ si et seulement si $f(x, y) = 0$). De plus, M se trouve dans l'un des demi-plans (ouverts) définis par $[AB]$ si $f(x, y) > 0$; sinon, il se trouve dans l'autre. Par conséquent, deux points donnés M et M' appartiennent au même demi-plan si et seulement si $f(x, y) \cdot f(x', y') > 0$.

On en déduit une fonction *scilab* : `memecote` retourne 1 si deux points P et Q sont strictement du même côté de $[AB]$, 0 sinon (un point `M` quelconque se note maintenant `[M(1),M(2)]`).

```
function R=memecote(A,B,P,Q)
    R=0;
    if (((P(2)-A(2))*(B(1)-A(1))-(B(2)-A(2))*(P(1)-A(1)))
    *((Q(2)-A(2))*(B(1)-A(1))-(B(2)-A(2))*(Q(1)-A(1)))>0) then
    R=1;
    end;
endfunction
```

2 - Solution algorithmique :

On propose ci-dessous une fonction *scilab* `inttriangle` qui, étant donné un vrai triangle $A_1A_2A_3$ (ces points sont distincts) et un point M qui n'est sur aucun côté de ce triangle retourne `M est à l'intérieur du triangle` ou `M n'est pas à l'intérieur du triangle` suivant le cas. Pour exécuter cette fonction, on aura chargé les fonctions `memecote` et `inttriangle` dans la console.

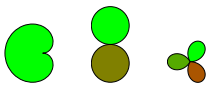
```
function A=inttriangle(A1,A2,A3,M)
    c=memecote(A1,A2,A3,M);
    a=memecote(A2,A3,A1,M);
    b=memecote(A3,A1,A2,M);
    if (a*b*c==1) then
        A='M est a l''interieur du triangle';
    else
        A='M n''est pas a l''interieur du triangle';
    end;
endfunction;
```

Le script qui suit permet de saisir les données, d'exécuter la fonction `inttriangle` et d'afficher la réponse à la question posée :

```
A1=input('A1=');
A2=input('A2=');
A3=input('A3=');/
M=input('M=');
Verdict=inttriangle(A1,A2,A3,M);
afficher(Verdict);
```

Application :

```
-->clear
-->exec('Chemin de la fonction memecote', -1)
-->exec('Chemin de la fonction inttriangle', -1)
-->exec('Chemin du script', -1)
A1=[1,1]
A2=[3,3.7]
A3=[2,-4.3]
M=[1.5,1]
    M est a l'interieur du triangle
 -->
```



7.10 [**] Ce triangle est-il isocèle ?

Un triangle ABC est donné par les coordonnées de ses sommets dans un repère orthonormé. Écrire un algorithme qui permette d'en déduire qu'il est isocèle (sans être équilatéral), équilatéral ou quelconque (c'est à dire, ici, ni isocèle, ni équilatéral).

Mots-clefs : instruction conditionnelle `if then elseif then end`, test `==`, commande `trier`, opérateurs logiques `&` et `|`, afficher une chaîne de caractères.

Tester si un triangle ABC donné est isocèle ou équilatéral revient à comparer les longueurs de ses côtés, donc à tester plusieurs fois l'égalité de deux nombres. Le script ci-dessous permet de saisir les coordonnées des sommets de ABC, de calculer les carrés des longueurs des côtés AB, BC et CA, carrés notés x, y et z dans le script et de tester de deux façons l'égalité de ces longueurs.

Remarques

• Comparer les carrés des longueurs des côtés (au lieu de comparer les longueurs) permet d'éviter le calcul de racines carrées, ce qui induirait des approximations de calcul supplémentaires.

• Le script ci-dessous contient deux solutions.

• La première solution serait impeccable si *scilab* était capable de calculer x, y et z exactement. Comme ce n'est pas le cas, *scilab* peut se tromper, voir l'exécution du script ci-dessous.

• La deuxième solution considère que $x = y$ si $|x - y|/(x + y)$ est $<$ *%eps* (et de même pour les autres couples de côtés), autrement dit si le nombre $|x - y|/(x + y)$ est considéré comme nul par *scilab*, ce qui ne veut pas dire qu'il est effectivement nul. C'est pratiquement ce que l'on peut faire de mieux.

• *%eps* est très petit : c'est en effet le plus petit nombre > 0 reconnu par *scilab*.

• %eps=2.220446049D-16, voir un manuel.

Le script :

```
A=input("Coordonnees de A = ");// [abscisse, ordonnee]
B=input("Coordonnees de B = ");// idem
C=input("Coordonnees de C = ");// idem
x=(B(1)-A(1))^2+(B(2)-A(2))^2;// x=AB^2
y=(C(1)-A(1))^2+(C(2)-A(2))^2;//y=AC^2
z=(C(1)-B(1))^2+(C(2)-B(2))^2;//z=BC^2
// Test ne tenant pas compte des approximations de calcul.
if x==y & y==z then
    disp("ABC est equilateral")
elseif x==y |y==z |z==x then
    disp("ABC est isocele")
else
    disp("ABC n''est ni equilateral, ni isocele")
end
// Test tenant compte des approximations de calcul et de %eps.
if abs(x-y)/(x+y)<%eps & abs(y-z)/(y+z)<%eps then
    disp("ABC est equilateral")
elseif abs(x-y)/(x+y)<%eps |abs(y-z)/(y+z)<%eps |abs(z-x)/(z+x)<%eps then
    disp("ABC est isocele")
else
    disp("ABC n''est ni equilateral, ni isocele")
end
```

Une exécution du script :

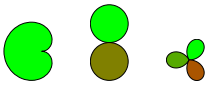
```
-->exec('Chemin du script', -1)
Coordonnees de A = [-23/15,-36/35]
Coordonnees de B =  [13/15,76/35]
Coordonnees de C = [-89/15,167/35]
 ABC n'est ni equilateral, ni isocele
 ABC est isocele
```

La première réponse est fausse, la seconde est exacte car un calcul à la main ou avec *Xcas*, qui fait des calculs exacts en nombres rationnels, donne

$$x = 16, \quad y = 53, \quad z = 53$$

Ce qu'il faut retenir est que *les réponses données par le script ne sont pas sûres*, deux longueurs en réalité égales pouvant être considérées comme différentes par *scilab*, deux longueurs en réalité différentes pouvant être considérées comme égales.

Finalement, le principal intérêt de cet exercice est le maniement d'opérateurs logiques.



Chapitre 8

Boucle tant que

Syntaxe des boucles `tant que`

```
while conditions
instructions ;
end
```

Tant que les conditions sont vérifiées, les instructions sont répétées.

Liste des exercices :

Énoncé n° 8.1 : [*] Transformer une boucle `pour` en une boucle `tant que`.

Énoncé n° 8.2 : [*] Calculer le maximum d'une liste de nombres.

Énoncé n° 8.3 : [*] Conversion du système décimal vers le système à base b.

Énoncé n° 8.4 : [*] Sommes de nombres impairs.

Énoncé n° 8.5 : [*] Partie entière d'un nombre réel.

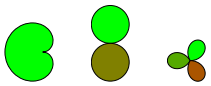
Énoncé n° 8.6 : [*] Formule de la somme des carrés.

Énoncé n° 8.7 : [**] Nombres factoriels.

Énoncé n° 8.8 : [***] Ranger 3 nombres donnés dans l'ordre croissant (tri à bulles).

Énoncé n° 8.9 : [***] Ranger n nombres donnés dans l'ordre croissant.

Énoncé n° 8.10 : [****] Ranger n nombres donnés dans l'ordre croissant.



8.1 [*] Transformer une boucle `pour` en une boucle `tant que`

Calculer le maximum d'une liste de nombres d'après la fonction `Min` de 6.2, puis remplacer la boucle `pour` par une boucle `tant que`.

Mots-clefs : Maximum, liste de nombres, commandes `max`, `tic`, `toc`, `disp` (afficher).

On peut toujours transformer une boucle `pour` (fonction `maxxx1`, analogue à la fonction `Min` de 6.2) en une boucle `tant que` (fonction `maxxx2` ci-dessous). Il suffit d'ajouter un compteur. C'est l'objet de cet exercice.

```
// Solution avec une boucle pour.
function m=maxxx1(L)// L est une liste de nombres ; m sera son maximum.
    m=L(1);
    n=length(L);
    for i=1:n-1
        if m<L(i+1) then
            m=L(i+1);
        end
    end
endfunction
// Transformation de la boucle pour en une boucle tant que.
function m=maxxx2(L)// L est une liste de nombres ; m sera son maximum.
    j=1;// Compteur.
    m=L(1);
    while j<length(L)
        if m<L(j+1) then
            m=L(j+1);
        end
        j=j+1;// Incrément de 1 du compteur.
    end
endfunction
```

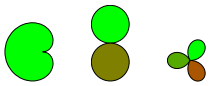
Essai numérique :

```
-->clear
-->exec('Chemin des fonctions maxxx1 et maxxx2', -1)
-->L=grand(1,80000,'uin',-250000,100000);//Échantillon de taille 80000 de
// la loi uniforme sur l'intervalle d'entiers [-250000,100000].
-->tic();disp(maxxx1(L));disp(toc());
    99999.
    0.066
-->tic();disp(maxxx2(L));disp(toc());
    99999.
    0.176
-->tic();disp(max(L));disp(toc());
    99999.
    0.003
-->
```

Cet essai suggère que la boucle `pour` est plus rapide que la boucle `tant que`, sans doute parce qu'il y a, en gros, deux fois plus de tests dans ce cas. La commande *scilab* `max` est bien sûr hors concours. Cette remarque

est confirmée par l'essai suivant, où L est un échantillon de la loi uniforme sur l'intervalle $[-250000, 100000[$.

```
-->exec('Chemin des fonctions maxxx1 et maxxx2', -1)
-->L = grand(1,80000, 'unf', -250000,100000);
-->tic();disp(maxxx1(L));disp(toc());
    99993.236
    0.06
-->tic();disp(maxxx2(L));disp(toc());
    99993.236
    0.177
-->tic();disp(max(L));disp(toc());
    99993.236
    0.003
```



8.2 [*] Calculer le maximum d'une liste de nombres

On veut calculer le maximum d'une liste de nombres `L` donnée, de longueur au moins 2, de la manière suivante : si `L(2)` est plus grand que `L(1)`, on échange ces nombres ; ensuite, on efface tous les termes qui sont inférieurs ou égaux à `L(1)` à partir du rang 2. Ce faisant, la longueur de `L` diminue mais reste supérieure ou égale à 1. Si elle est encore supérieure ou égale à 2, on recommence. Quand elle est égale à 1, le terme restant est le maximum recherché.

Mots-clefs : boucle `tant que`, maximum, appel d'une fonction *scilab*, instruction conditionnelle `if…then …end`, commandes `length`, `find`, `tic() …toc()`.

La solution normale est d'utiliser la commande `max` de *scilab*. Ce qui suit est un exercice sur les boucles `tant que`.

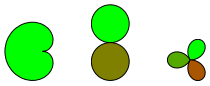
La méthode imposée dans l'énoncé, appelée `maxxi` ci-dessous, devrait améliorer la vitesse de la fonction `maxxx2` de l'exercice 8.1.

```
function m=maxxi(L)//L : liste d'au moins 2 nombres ; m sera son maximum.
    n=length(L);
    while n>=2
        if L(1)<L(2) then
            L([1,2])=L([2,1]);// Si nécessaire, on échange les 2 premiers
            // termes de L.
        end
        M=L(2:n);//On va effacer tous les termes de M qui sont <= L(1)
        M(find(M<=L(1)))=[];// Au moins un terme est effacé.
        L=[L(1),M];// La longueur de L reste >= 1.
        n=length(L);// La boucle while s'arrêtera lorsque n=1.
    end
m=L(1);
endfunction
```

Les essais ci-dessous (voir aussi l'exercice 8.1) montrent que la fonction `maxxi` est plutôt rapide.

```
clear
-->exec('Chemin de la fonction maxxi', -1)
-->exec('Chemin de la fonction maxxx2', -1)
-->L = grand(1,80000, 'uin', -250000,100000);
-->tic();disp(maxxi(L));disp(toc());
    99999.
    0.008
-->tic();disp(maxxx2(L));disp(toc());
    99999.
    0.173
-->L = grand(1,80000, 'unf', -250000,100000);
-->tic();disp(maxxi(L));disp(toc());
    99998.171
    0.008
-->tic();disp(maxxx2(L));disp(toc());
    99998.171
```

```
    0.179
-->
```



8.3 [*] Conversion du système décimal vers le système à base b

Un nombre entier $n \geqslant 0$ est donné sous forme décimale. Le problème posé est de l'écrire dans le système à base $b \geqslant 2$, c'est à dire sous la forme d'une liste $[a_0, a_1, \ldots, a_p]$ de 0, de 1, ..., de $b-1$ de sorte que $n = a_0 \cdot b^0 + a_1 \cdot b^1 + \cdots + a_p \cdot b^p$.

Mots-clefs : boucle `tant que`, commandes `input`, `reste`, `quotient`.

Cas $b = 2$: Ce cas n'est pas un cas particulier. Voici une ancienne rédaction consacrée à ce cas. On sait que les coefficients $a_0, \ldots, a_p$ s'obtiennent par des divisions successives par 2, ce qui donne :

```
n=input('n=');// n est un entier >0.
A=[];
m=0;
c=0;
p=n;
while m<n
    a=reste(p,2);
    A=[A,a];
    m=m+a*2^c;
    p=quotient(p-a,2);
    c=c+1;
end
afficher(A,m)
```

La commande `input` permet de saisir la valeur de `n` dans la console. Dans l'exemple ci-dessous, on a choisi `n=123 456 789`. Afficher `m` permet de vérifier le calcul car `m` est le résultat de la conversion de `n` du système à base 2 vers le système décimal. On revient au point de départ. Voici ce que l'on obtient :

```
-->exec('Chemin du script', -1)
n=123456789
    123456789.
          column  1 to 14
!1  0  1  0  1  0  0  0  1  0  1  1  0  0  !
          column 15 to 27
!1  1  1  1  0  1  1  0  1  0  1  1  1  !
-->
```

En fait, on a affiché non pas le vecteur `A` qui ne serait pas bien lisible (une ligne, 27 colonnes), mais la chaîne de caractères associée. Bien sûr, c'est `A=[1 0 1 0 1 0 0 0 1 0 1 1 0 0 1 1 1 1 0 1 1 0 1 0 1 1 1]` que l'on aurait aimé afficher.

Cas général $b \geqslant 2$: utilisons le script suivant

```
// Convertit en base b un entier ecrit en base 10
function B=convertion10b(n,b)// n>=0,b>=2, entiers.
    q=n,
    B=[];
    while q>=b
        B=[B,reste(q,b)];
        q=quotient(q,b);
    end
```

```
    B=[B,q];
    afficher(B);
endfunction
```

Exemples : 6 882 309 est-il divisible par 7 ?

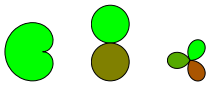
```
-->clear
-->exec('Chemin de convertion10b.sci', -1)
-->B=convertion10b(6882309,7);
    0.    2.    0.    3.    3.    3.    2.    1.    1.
-->n=sum(B.*7.^(0:taille(B)-1))
 n  =
    6882309.
-->
```

Le chiffre des unités dans la base b (à gauche) est 0, donc 6 882 309 est divisible par 7. Bien sûr, si on sait comment faire une division par 7, c'est quand même plus rapide. Le calcul de `n` ci-dessus (voir l'exercice 3.3) est une vérification inutile.

On peut aussi se demander si 6 882 309 est divisible par 17 comme précédemment :

```
-->clear
-->exec('Chemin de convertion10b.sci', -1)
-->B=convertion10b(6882309,17);
    12.    3.    14.    6.    14.    4.
-->
```

Le reste de la division de 6 882 309 par 17 est 12.



8.4 [*] Sommes des premiers nombres impairs

Calculer la somme `S(n)` des nombres impairs inférieurs ou égaux à un entier donné n.

Mots-clefs : boucle `tant que`, commandes `input`, `sum`.

Cet exercice est élémentaire. On peut utiliser le script ci-dessous. On remarquera l'affichage : il est constitué de 4 chaînes de caractères (deux chaînes limitées par le signe `'` et 2 chaînes `string()`). Les 3 signes `+` concatènent ces 4 chaînes.

```
n=input('n=');
S=0 ;// Initialisation de la somme
i=1;
while (2*i-1)<= n// debut de la boucle.
  S=S+2*i-1;
  i=i+1;
end// Fin de la boucle.
afficher('La somme des impairs <= '+string(n)+ ' est '+string(S));
```

La commande `input` permet de saisir la valeur de n, ici 500, dans la console.

```
-->exec('Chemin du script', -1)
n=500
  La somme des impairs <= 500 est 62500
-->
```

On peut faire beaucoup mieux, par exemple directement dans la console :

```
-->sum(1:2:500)
 ans  =
    62500.
-->
```

En effet, la commande `1:2:n` retourne la liste des nombres $(1, 3, \ldots, n)$ si `n` est impair, $(1, 3, \ldots, n-1)$ sinon, tandis que `sum(V)` retourne la somme d'une liste (ou d'une matrice) de nombres `V`.

Solution mathématique : soit $\mathbb{S}(n)$ la somme des n premiers entiers. En général, on sait que

$$\mathbb{S}(n) = \frac{n(n+1)}{2}.$$

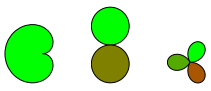
On en déduit, dans le cas où n est impair, noté $n = 2p+1$:

$$S(n) = \mathbb{S}(2p+1) - 2\cdot\mathbb{S}(p) = \frac{(n+1)^2}{2}$$

d'où l'on déduit, si n est pair :

$$S(n) = S(n-1) = \frac{n^2}{2}.$$

On ne peut pas démontrer de telles formules littérales avec *scilab*. C'est possible avec un logiciel de calcul formel comme *Xcas*.



8.5 [*] Partie entière d'un nombre réel

Calculer la partie entière d'un nombre réel donné.

Mots-clefs : instruction conditionnelle avec alternative, boucles `tant que`, commandes `input`, `floor`, `afficher`.

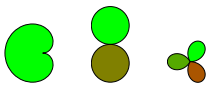
La partie entière d'un réel donné x est le plus grand entier n tel que $n \leqslant x$, autrement dit, l'unique entier n qui vérifie $n \leqslant x < n+1$. Le script ci-dessous est très intéressant du point de vue de l'apprentissage de *scilab*.

```
clear
x=input('x=');
y=0;//On va distinguer les cas x<0 et x>=0.
if y>x// Cas x<0.
  while y>x then
    y=y-1;
  end;
else// Cas x>=0.
  while y<=x then
    y=y+1;
  end
  y=y-1;
end;
afficher('La partie entiere de x est '+string(y));
```

L'exécution de ce script pour trouver la partie entière de $x = -2.117$ donne :

```
-->exec('Chemin du script', -1)
x=-2.117
 La partie entiere de x est -3
-->floor(-2.117)
 ans  =
  - 3.
-->
```

Nous avons confirmé le résultat obtenu à l'aide de la commande `floor` qui est justement la fonction *partie entière* de *scilab*. Nous aurions pu transformer le script proposé en fonction-*scilab* et oublier `floor`. Ce serait idiot, évidemment.



8.6 [*] Formule de la somme des carrés

La formule $1^2 + \cdots + n^2 = \dfrac{n(n+1)(2n+1)}{6}$ est-elle vraie pour tout entier $n \geqslant 1$?

Mots-clefs : boucle `tant que`, abandon d'un calcul.

On sait que cette formule est vraie quel que soit l'entier $n \geqslant 1$, c'est à dire que si l'on pose

$$\forall n \in \mathbb{N}^*, \quad S(n) = 1^2 + \cdots + n^2 \quad \text{et} \quad T(n) = \frac{n(n+1)(2n+1)}{6}$$

$S(n) = T(n)$ quel que soit l'entier $n \geqslant 1$. Essayons de vérifier cela à l'aide d'un script :

- Il est évident que c'est vrai pour $n = 1$.
- Augmentons n de 1 et vérifions que $S(n) = T(n)$.
- Si oui, on itère l'étape précédente ; sinon, on arrête.

Le script suivant cherche en fait le premier entier n pour lequel $S(n) \neq T(n)$:

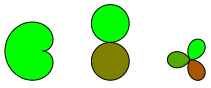
```
n=1;
S=1;
T=1;
while S=T
    n=n+1;
    S=S+n^2;
    T=n*(n+1)*(2*n+1)/6;
end
afficher(+string(n)+' est le plus petit entier pour lequel la formule
proposee est fausse');
```

C'est angoissant parce que l'on sait que les calculs ne vont pas s'arrêter ! Lançons (quand même) l'exécution de l'algorithme, sachant que l'on pourra de toute façon abandonner les calculs (Contrôle>Abandonner). Voilà ce que l'on obtient :

```
-->exec('Chemin du script', -1)
 208065 est le plus petit entier pour lequel la
     formule proposee est fausse
-->
```

Cette erreur de *scilab* est probablement dûe au fait que sa capacité à traiter exactement ces nombres entiers a été dépassée (ils ont été transformés en nombres flottants approximatifs).

De plus, il est clair a priori qu'un calculateur numérique ne peut pas être utilisé pour démontrer qu'une infinité d'égalités sont satisfaites.



8.7 [**] Comptage de nombres factoriels

1 - Combien y a-t-il de nombres factoriels inférieurs ou égaux à 10^6, à 10^9, à 10^{12}, à $(17!)$?
2 - 3 629 300 et 39 916 800 sont-ils des nombres factoriels ?

Mots-clefs : boucles `pour` et `tant que`, instruction conditionnelle, `prod`, `factorielle`, `format`.

Les scripts ci-dessous, rédigés de manière un peu compacte, appellent des commentaires :

• Comme on demande 4 fois de compter des nombres factoriels, on a intérêt à programmer une fonction-*scilab* qui associe à tout entier $p \geqslant 1$ le nombre $n_f(p)$ de nombres factoriels $\leqslant p$. C'est l'objet du premier script. Cette fonction a été appelée `nombfact`.

• Le nombre factoriel `p !` est calculé à l'aide de la commande `prod([1:p])`. On aurait pu utiliser directement la commande *scilab* `factorielle(p)`.

• `format('v',25)` empêche *scilab*, dans cet exercice, d'afficher certains grands entiers comme `17 !` en notation scientifique standard. Sinon, `17 !` serait affiché `3.556874281D+14` au lieu de `355687428096` (en fait, `355687428096.` car *scilab* ne connaît pas les nombres entiers : le point qui suit `355687428096` est le point qui sépare la partie entière de la partie décimale de `17 !` en notation scientifique standard).

• $p \geqslant 2$ est un nombre factoriel si et seulement si $n_f(p-1) \neq n_f(p)$.

Listing 8.1 – Fonction nombfact

```
function n=nombfact(p)
C=1; // C vaudra 1, 2, 3, ...
fact=1; // les valeurs de fact seront les nombres factoriels successifs
while fact<=p then
  C=C+1;
  fact=fact*C;
end
n=C-1;
endfunction
```

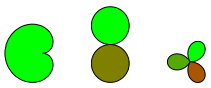
Chargeons `nombfact` dans la console. Nous répondrons aux deux questions à l'aide du script :

Listing 8.2 – Compter et reconnaître des factoriels

```
// Comptage de factorielles
format('v',25);
P=[10^6,10^9,10^(12),prod([1:17])];
for i=1:4
    NF=nombfact(P(i));
    afficher('Le nombre de nombres factoriels <= '+string(P(i))+' est '
    +string(NF));
end;
// n est-il un nombre factoriel ?
n=[3629200,39916800];
for i=1:2
    if (nombfact(n(i)-1)==nombfact(n(i))) then
        afficher(string(n(i))+' n''est pas un nombre factoriel.');
    else
        afficher(string(n(i))+' est un nombre factoriel')
    end;
end;
```

ce qui donne :

```
-->exec('Chemin de nombfact', -1)
 Le nombre de nombres factoriels <= 1000000 est
      9
 Le nombre de nombres factoriels <= 1000000000 e
      st 12
 Le nombre de nombres factoriels <= 100000000000
      0 est 14
 Le nombre de nombres factoriels <= 355687428096
      000 est 17
 3629200 n'est pas un nombre factoriel.
 39916800 est un nombre factoriel
-->
```



8.8 [***] Ranger 3 nombres donnés dans l'ordre croissant (tri à bulle)

Pour ranger 3 nombres donnés x, y et z dans l'ordre croissant, on considère d'abord x et y que l'on échange si $x > y$; puis on considère y et z et on fait de même ; on compte aussi le nombre de permutations de 2 nombres consécutifs que l'on a réalisées au cours de ce premier passage. Si c'est 0, c'est terminé. Sinon, on fait un deuxième passage et ainsi de suite, éventuellement. On demande d'écrire la suite x, y, z dans l'ordre croissant et d'afficher le nombre de permutations faites.

Mots-clefs : instructions conditionnelles `if …then …end`, boucle `tant que`, `trier`, « exécution pas à pas ».

Ce problème a déjà été traité (mieux) dans l'exercice 7.4.

Solution n° 1 : Le script ci-dessous traduit exactement le procédé de tri décrit dans l'énoncé. C'est un script maladroit. On peut passer directement à la solution n° 2.

```
x=input("x=")
y=input("y=")
z=input("z=")
S=1;
NP=0;
while S>0 then
  if x>y then
    S1=1;
    t=x;
    x=y;
    y=t;
  else
    S1=0;
  end;
  if y>z then
    S2=1;
    t=y;
    y=z;
    z=t;
  else
    S2=0;
  end;
  S=S1+S2;
  NP=NP+S;
end;
A=[x,y,z];
afficher(A);
afficher('Le nombre de permutations effectuees est '+string(NP));
```

• L'intérêt du script ci-dessus est qu'on peut le généraliser facilement pour un nombre quelconque de nombres (à partir de 4 nombres, trier est compliqué à programmer). Le tri utilisé ici s'appelle « tri à bulle ».
• Il existe beaucoup d'autres types de tri, dont la fonction `trier` de *scilab*, voir l'aide en ligne ou un manuel.

Voici une exécution du script ci-dessus :

```
-->exec('Chemin du script', -1)
x=-1.17
y=-2.34
```

```
z=-4.71
    - 4.71   - 2.34   - 1.17
    Le nombre de permutations effectuees est 3
-->
```

En utilisant la commande `trier` de *scilab*, on aurait obtenu :

```
-->trier([-1.17,-2.34,-4.71])
 ans  =
    - 4.71   - 2.34   - 1.17
-->
```

Solution n° 2 : En fait, on peut compacter le script ci-dessus. On est sûr qu'après 2 passages, les nombres donnés sont rangés dans l'ordre croissant. Par conséquent, quitte à perdre un peu de temps, faisons ces 2 passages sans compter le nombre des permutations. De plus, remplaçons systématiquement les couples de nombres successifs examinés par leur minimum suivi de leur maximum. Cela donne le script :

```
A=input("A=")// A est une liste de 3 nombres constituant les donnees.
Atri=A;// Pour conserver la valeur de A.
for i=1:2
Atri([1,2])=[min(Atri(1),Atri(2)),max(Atri(1),Atri(2))]
Atri([2,3])=[min(Atri(2),Atri(3)),max(Atri(2),Atri(3))]
end
afficher('Le triplet initial est ')
afficher(A)
afficher('Le triplet final est ')
afficher(Atri)
```

Traitons l'exemple où `A` est obtenu avec la commande `tirage_entier(2,-15,8)` et choisissons le mode d'exécution 7 (exécution pas à pas). Cela donne :

```
-->exec('Chemin du script', 7)
step-by-step mode: enter carriage return to proceed
>>
>>A=input("A=")// A est une liste de 3 nombres constituant les donnees.
A=tirage_entier(3,-15,8)
 A  =
    - 6.   - 11.   - 4.
>>
>>Atri=A;// Pour conserver la valeur de A.
>>
>>for i=1:3
>>Atri([1,2])=[min(Atri(1),Atri(2)),max(Atri(1),Atri(2))]
>>Atri([2,3])=[min(Atri(2),Atri(3)),max(Atri(2),Atri(3))]
>>end
 Atri  =
    - 11.   - 6.   - 4.
 Atri  =
    - 11.   - 6.   - 4.   // Fin du premier passage.
 Atri  =
    - 11.   - 6.   - 4.
 Atri  =
    - 11.   - 6.   - 4. // Fin du deuxieme passage.
>>
>>afficher('Le triplet initial est ')
```

```
Le triplet initial est
>>
>>afficher(A)
   - 6.  - 11.  - 4.
>>
>>afficher('Le triplet final est ')
Le triplet final est
>>
>>afficher(Atri)
   - 11.  - 6.  - 4.
>>
-->
```

Pour aller plus loin (un peu de Calcul des probabilités) :

1 - Quelles valeurs NP peut-elle prendre ?

Pour compter les permutations, on peut imaginer que l'on a tiré 1, 2 et 3 (seules interviennent les inégalités), ce qui donne le tableau :

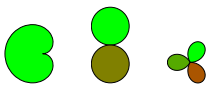
			NP
1	2	3	0
1	3	2	1
2	1	3	1
2	3	1	2
3	1	2	2
3	2	1	3

2 - Avec quelles probabilités ces valeurs sont-elles prises ?

De simples considérations de symétrie indiquent que la loi de NP est

NP	0	1	2	3
probabilités	1/6	1/3	1/3	1/6

Plus précisément, le volume de $((x, y, z)\,;\, 0 < x < y < z < 1)$ est 1/6 parce qu'il y a 6 tétraèdres de ce type qui remplissent le cube unité (à des ensembles de volume nul près) et que ces tétraèdres ont le même volume par raison de symétrie.



8.9 [***] Ranger n nombres donnés dans l'ordre croissant

Programmer une fonction *scilab* dont l'entrée est une liste de nombres `L`, qui retourne cette liste rangée dans l'ordre croissant, notée `R`, fabriquée comme suit : le premier terme de `R` est `min(L)`. Ensuite, on efface de la liste `L` un terme égal à `min(L)` (il peut y en avoir plusieurs). Le second terme de `R` est le minimum des termes de `L` restants, et ainsi de suite.

Mots-clefs : commandes `min`, `find`, `trier`.

La fonction *scilab* qui range selon la méthode imposée dans l'énoncé peut être programmée ainsi :

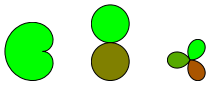
```
function R=ttrier(L)// L : liste de nombres.
    R=[];// Future liste ordonnee.
    while taille(L)>0
        R=[R,min(L)];
        L(min(find(L==min(L))))=[];
    end
endfunction
```

Utilisons cette fonction ainsi que la commande `trier` et comparons les temps de calcul du tri :

```
-->clear
-->exec('Chemin de la fonction ttrier', -1)
-->L=tirage_reel(100000,0,1);
-->tic();
-->R=ttrier(L);toc();// tri de L avec ttrier.
-->temps=toc()// temps de calcul du tri avec ttrier.
 temps  =
    236.
-->format('v',8);// affichage a 5 decimales.
-->afficher(R(1:10),L(1:10))// 10 premiers termes de L et de R.
         column 1 to 5
    0.96769     0.15761     0.72584     0.97059     0.98111
         column  6 to 10
    0.95717     0.10986     0.48538     0.79811     0.80028
         column 1 to 5
    0.00001     0.00001     0.00005     0.00005     0.00007
         column  6 to 10
    0.00007     0.00007     0.00007     0.00007     0.00009
-->afficher(R(99991:100000),L(99991:100000))// 10 derniers termes de L et de R.
         column 1 to 5
    0.58883     0.41749     0.14204     0.16188     0.29871
         column  6 to 10
    0.02526     0.15916     0.71615     0.49549     0.51918
          column 1 to 5
     0.99991     0.99992     0.99993     0.99993     0.99994
         column  6 to 10
    0.99995     0.99997     0.99997     0.99997     0.99998
-->tic();S=trier(L);toc();// tri avec trier.
-->afficher(toc())// temps de calcul du tri avec trier.
    39.225
-->
```

On constate que la commande `trier` de *scilab* est beaucoup plus rapide.

Autres solutions : Voir les exercices 6.24, 6.25 et 8.10.



8.10 [***] Ranger n nombres donnés dans l'ordre croissant

Progammer une fonction de tri exécutable par *scilab* à partir de la description suivante :
Un vecteur de nombres noté `A` étant donné ; on appellera `Aneuf` ce vecteur rangé dans l'ordre croissant. On détermine d'abord le plus petit nombre `m` de `A`, on compte combien de fois `n` le nombre `m` figure dans `A`, on ajoute à `Aneuf` (vide au début) la suite `m`, ..., `m` (`m` écrit `n` fois), on efface tous les `m` qui figurent dans `A`. Si `A` n'est pas vide, on recommence.

Mots-clefs : commande `find`.

Cet exercice est une variante de 8.9

L'énoncé se traduit par l'algorithme suivant :

```
function Aneuf=Trier(A)// A est une suite de nombres.
    Aneuf=[];
    while size(A)>0
        m=min(A);
        B=find(A==m);
        A(1,B)=[];
        Aneuf=[Aneuf,m*ones(B)];
    end
    // afficher("La liste A dans l''ordre croissant s''ecrit :");
    // afficher(Aneuf);
endfunction
```

Appliqué par exemple à une suite de 10 entiers tirés au hasard entre -1 et 4 selon la loi uniforme, cela donne :

```
exec('Chemin de Trier', -1)
-->A=tirage_entier(10,-1,4)
 A  =
    4.    0.    4.    0.  - 1.    1.    3.    2.    3.    4.
 -->Aneuf=Trier(A);
  La liste A dans l'ordre croissant s'ecrit :
   - 1.    0.    0.    1.    2.    3.    3.    4.    4.    4.
-->
```

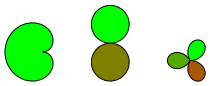
On pourrait aussi bien trier des nombres réels. Dans l'exemple qui suit, on trie 100000 nombres réels avec la fonction `Trier` puis avec la commande `trier` de *scilab*. On affiche le temps pris par le tri en secondes dans chaque cas ainsi que les 5 derniers termes triés.

```
-->clear;
-->exec('Chemin de Trier', -1)
-->X=tirage_reel(100000,-1,4);
-->tic();Xneuf=Trier(X);temps=toc();afficher(temps);
265.225
-->tic();Xtri=trier(X);temps=toc();afficher(temps);
    0.053
-->Xneuf(99996:100000)
 ans  =
    3.99976324616    3.9999442009721    3.9999521544669    3.999977289699
3.9999918183312
-->Xtri(99996:100000)
```

```
 ans  =
    3.99976324616    3.9999442009721    3.9999521544669    3.999977289699
3.9999918183312
-->
```

La commande `trier` de *scilab* est beaucoup plus rapide.

Autres solutions : Voir les exercices 6.24, 6.25 et 8.9.



Chapitre 9

Graphes

Pour `plot2d`, on consultera de préférence [11], chapitre 19. Sauf erreur, [12] ne parle pas de cette commande. [1] est très indigeste.

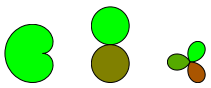
Liste des exercices :

Énoncé n° 9.1 : [*] Tracer un triangle.

Énoncé n° 9.2 : [***] Tracés de polygones réguliers inscrits dans un cercle.

Énoncé n° 9.3 : [***] Tracer les N premiers flocons de Koch.

Énoncé n° 9.4 : [***] Visualisation du tri à bulles.



9.1 [*] Tracer un triangle

Tracer le triangle ABC, ces points ayant respectivement pour coordonnées $(1,3)$, $(-2,4)$ et $(3,8)$.

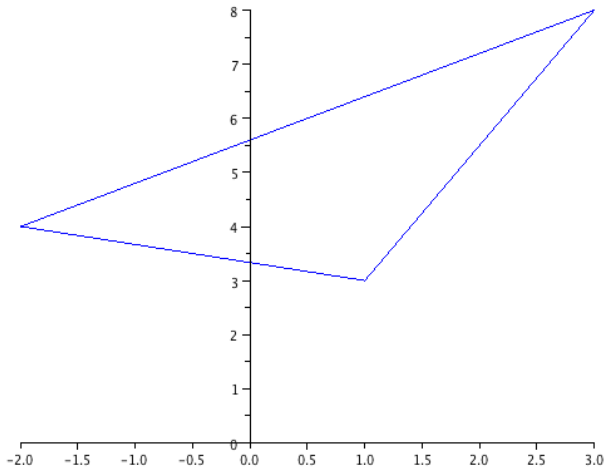
Mots-clefs : commandes `clf`, `plot`, `orthonorme(), plot2d`, `a=gca ( )`, `a.isoview="on"`, `axesflag =0`, `a.axes\_visible=["off","off","off"]`.

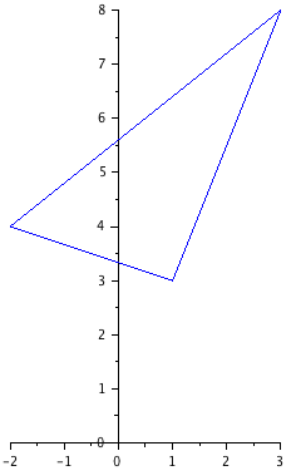
On utilise le vecteur des abscisses des points A, B, C et A puis le vecteur de leurs ordonnées, dans le même ordre. Le repère est orthonormé.

```
clf;
X=[1,-2,3,1];
Y=[3,4,8,3];
plot(X,Y);
```

Ce script particulièrement simple conduit au tracé de gauche. Le repère est choisi par *scilab*. On constate qu'il n'est pas orthonormé. On peut imposer qu'il le soit en imposant la commande `orthonorme()` [1], ce qui donne le script suivant et le trac de droite :

```
clf;
X=[1,-2,3,1];
Y=[3,4,8,3];
orthonorme();
plot(X,Y);
```





Remarques sur les scripts précédents :

• La commande `clf` nettoie la fenêtre graphique. Elle est indispensable si on a déjà fait des tracés dans la même session. Sans elle, les tracés se superposeraient.

• On remarque que le triangle a été fermé en répétant les coordonnées de A.

On pourrait aussi faire disparaître les axes, encadrer le tracé par souci esthétique, épaissir les traits, les colorer, ajouter des légendes, *etc.* Pour cela, il vaut mieux utiliser les commandes `plot2d` et `gca()`, cette dernière ayant énormément d'options, voir l'aide en ligne, bon courage.

1. Ne pas oublier `()`.

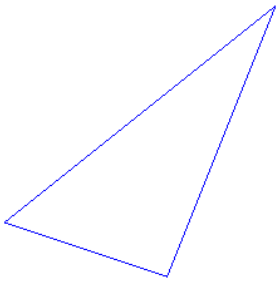
Dans le graphe ci-dessous, le repère orthonormé a été imposé par l'instruction `a.isoview="on";` au lieu de la commande `orthonorme()`, l'effacement des axes par la commande `a.axes_visible=["off","off","off"];`.

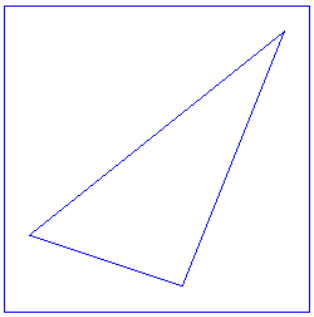
```
clf;
X=[1,-2,3,1];
Y=[3,4,8,3];
plot(X,Y);
a=gca();
a.axes_visible=["off","off","off"];// Les axes sont effaces.
a.isoview="on";// Le repere sera orthonorme.
```

Ce script retourne la figure de gauche ci-dessous. Pour encadrer le triangle par un carré (déterminé en bricolant, à la main), on peut exécuter le script qui suit :

```
U=[-2.5,3.5,3.5,-2.5,-2.5];
V=[2.5,2.5,8.5,8.5,2.5];
plot(U,V);
```

Ce script ne comportant pas la commande `clf`, on constate comme prévu que le carré vient se superposer au tracé existant.





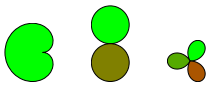
Remarque :

On peut aussi obtenir la figure de droite ci-dessus avec le listing

```
clf;
f=scf();
orthonorme();
X=[1,-2,3,1];
Y=[3,4,8,3];
plot2d(X,Y,axesflag=0);
U=[-2.5,3.5,3.5,-2.5,-2.5];
V=[2.5,2.5,8.5,8.5,2.5];
plot2d(U,V,axesflag=0);
```

scilab et la géométrie :

scilab n'est pas un logiciel de géométrie. Les logiciels de géomśtrie sont bien meilleurs. Néanmoins, dans certains calculs avec illustration géom'etrique, les capacit s graphiques de *scilab* pourront ´viter d'ouvrir une application de géomśtrie.



9.2 [***] Tracés de polygones réguliers inscrits dans un cercle

Tracer dans le même repère orthonormé le cercle de centre O et de rayon 1 ainsi que le triangle équilatéral, le carré et l'heptagone régulier inscrits dans ce cercle et dont un sommet est le point $(1, 0)$.

Mots-clefs : `plot`, `plot2d`, `orthonorme()`, `help getcolor`, `style=i`, `axesflag=0`.

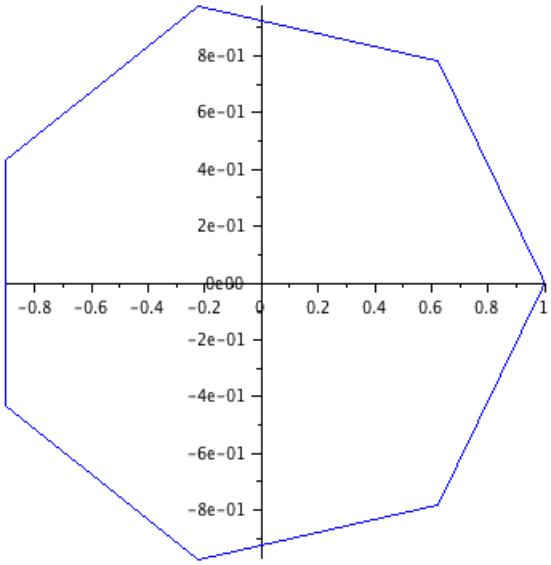
Cet exercice est intéressant. Pour tracer un polygone régulier à n côtés inscrits dans le cercle unité et admettant $(1, 0)$ comme sommet, il suffit d'exécuter le script suivant :

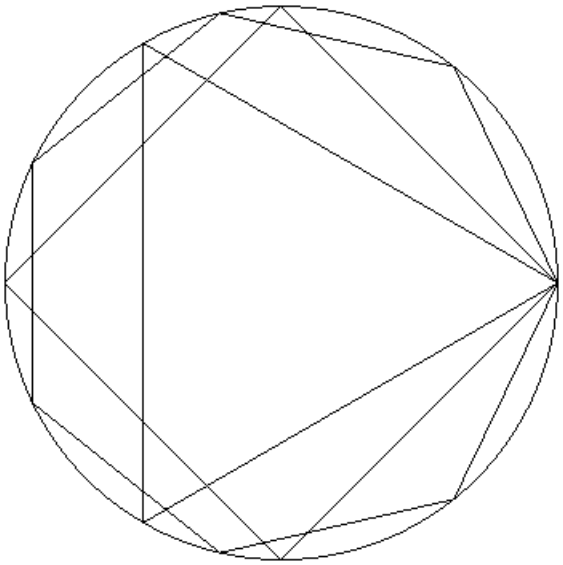
```
clf;
orthonorme();
n=input('n=');
X=cos(2*%pi/n*[0:n]);
Y=sin(2*%pi/n*[0:n]);
plot(X,Y);
```

Ce polygone sera bien fermé. On aura sans doute envie de rendre le repère orthonormé et d'effacer les axes, voir plus loin. En effet, quand on exécute ce script,

```
-->exec('Chemin du script', -1)
n=7
-->
```

on obtient la figure de gauche, très peu satisfaisante (il s'agit d'un heptagone) :





Lorsque `n` sera grand, ce polygone sera indistinguable à nos yeux du cercle-unité. On pourrait construire le cercle-unité autrement, mais ce sera toujours en réalité un tel polygone régulier. Il peut être utile de faire grandir le nombre de côtés et de trouver à partir de quelle valeur de `n` on voit un cercle, puis de vérifier que c'est un polygone à l'aide du zoom.

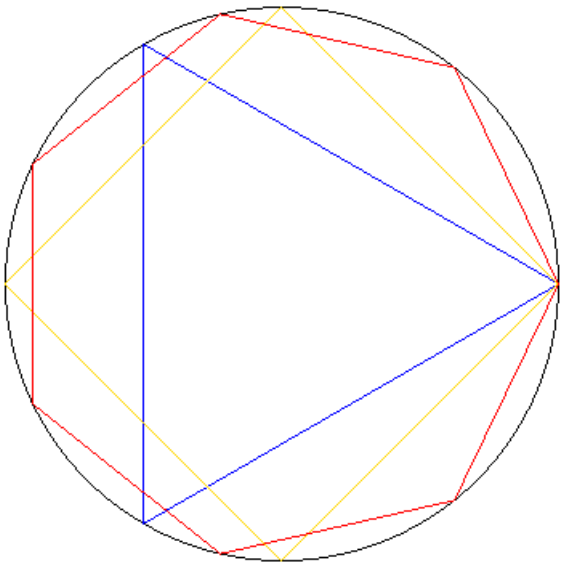
Pour le tracé de la figure de droite ci-dessus, nous avons choisi `n=100` et nous avons allongé `X` et `Y` en ajoutant les coordonnées des sommets du triangle, du carré et de l'heptagone demandés.

```
clf;
orthonorme();
X=cos([2*%pi/100*[0:100],2*%pi/3*[0:3],2*%pi/4*[0:4],2*%pi/7*[0:7]]);
Y=sin([2*%pi/100*[0:100],2*%pi/3*[0:3],2*%pi/4*[0:4],2*%pi/7*[0:7]]);
plot2d(X,Y,axesflag=0);
```

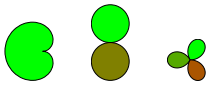
Colorions les 4 courbes tracées à l'aide du script :

```
clf;
orthonorme();
X100=cos(2*%pi/100*[0:100])';
Y100=sin(2*%pi/100*[0:100])';
plot2d(X100,Y100,style=1,axesflag=0);
X3=cos(2*%pi/3*[0:3]);
Y3=sin(2*%pi/3*[0:3]);
plot2d(X3,Y3,style=2,axesflag=0);
X4=cos(2*%pi/4*[0:4]);
Y4=sin(2*%pi/4*[0:4]);
plot2d(X4,Y4,style=32,axesflag=0);
X7=cos(2*%pi/7*[0:7]);
Y7=sin(2*%pi/7*[0:7]);
plot2d(X7,Y7,style=5,axesflag=0);
```

On obtient :



On peut obtenir beaucoup plus simplement cette figure à l'aide d'un logiciel de géométrie comme, par exemple, *GeoGebra*.



9.3 [***] Tracer les N premiers flocons de Koch

Tracer les N premiers flocons de Koch définis comme suit à partir de l'hexagone régulier ABCDEFA (ou K_0) inscrit dans le cercle unité dont le point A de coordonnées $(1, 0)$ est un sommet :
- Construction du premier flocon de Koch K_1 :
• d'abord on remplace le côté AB de l'hexagone par la ligne brisée AMGNB, M étant au tiers de AB à partir de A, N étant au tiers de AB à partir de B, le triangle MNG étant équilatéral, le point G étant à droite quand on se déplace de A à B ;
• puis on fait de même avec les 5 autres côtés de K_0.
Le polygone à 18 côtés ainsi obtenu est appelé premier flocon de Koch, noté K_1.
- Construction des flocons de Koch suivants : on passe de K_n à K_{n+1} comme on est passé de K_0 à K_1.

Mots-clefs : `plot`, `plot2d`, `orthonorme()`, `help getcolor`, `style=i`, `axesflag=0`.

L'énoncé étant un peu rapide, on pourra se reporter à [4] pour plus de précisions. Le script ci-dessous produit le tracé de K_0 en noir, sans les axes, dans un repère orthonormé :

```
clf;
orthonorme();
X=cos(%pi/3*[0:6]);
Y=sin(%pi/3*[0:6]);
plot2d(X,Y,style=1,axesflag=0);
```

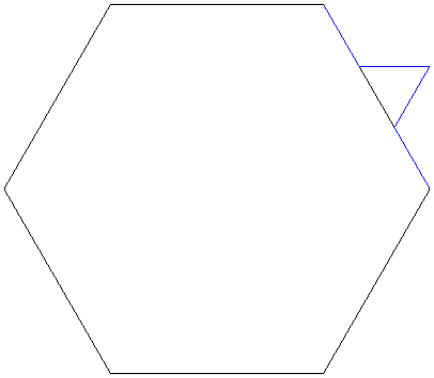
Pour remplacer le segment AB par la ligne brisé AMGNB, opération qui sera ensuite répétée sans cesse, on utilisera la fonction-*scilab* que nous avons appelée `Machinekoch` ainsi définie :

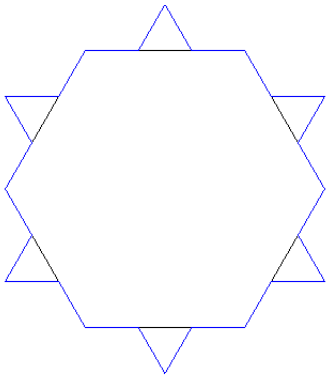
```
function Q=Machinekoch(P)// P a pour colonnes les coordonnees de A, puis de B.
    R=[P(:,1),2/3*P(:,1)+1/3*P(:,2),1/3*P(:,1)+2/3*P(:,2),P(:,2)];
    S=1/3*[1/2,sqrt(3)/2;-sqrt(3)/2,1/2]*(P(:,2)-P(:,1))+R(:,2);
    Q=[R(:,1:2),S,R(:,3:4)];
endfunction
```

Cette fonction étant chargée dans la console, le script qui vient superposera à K_0 la ligne brisée AMGNB en bleu :

```
P=[X(1:2);Y(1:2)];
Q=Machinekoch(P);
plot2d(Q(1,:),Q(2,:),style=2,axesflag=0);
```

ce qui donne la figure de gauche ci-dessous :





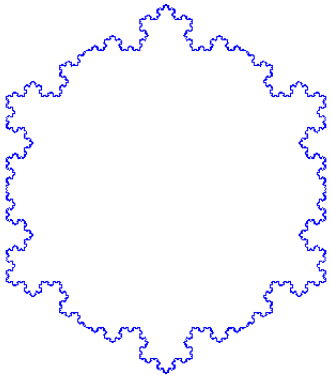
Il est dès lors facile d'obtenir K_1 en ajoutant une boucle `pour` au script précédent qui devient :

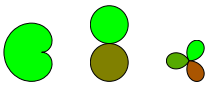
```
for i=1:6
    P=[X(i:i+1);Y(i:i+1)];
    Q=Machinekoch(P);
    plot2d(Q(1,:),Q(2,:),style=2,axesflag=0);
end
```

et retourne la figure de droite ci-dessus. Finalement, pour calculer K_N, $N \geqslant 1$ donné, on calcule successivement K_1, ..., K_N, à l'aide d'une nouvelle boucle `pour`. Le programme suivant rassemble toutes les étapes de la construction de K_N. Il retourne aussi le temps de calcul de K_N :

```
clf;
N=input('N=');
tic();
XY=[cos(%pi/3*[0:6]);sin(%pi/3*[0:6])];
function Q=Machinekoch(P)
    R=[P(:,1),2/3*P(:,1)+1/3*P(:,2),1/3*P(:,1)+2/3*P(:,2),P(:,2)];
    S=1/3*[1/2,sqrt(3)/2;-sqrt(3)/2,1/2]*(P(:,2)-P(:,1))+R(:,2);
    Q=[R(:,1:2),S,R(:,3:4)];
endfunction
for j=1:N
    n=6*4^(j-1);// Nombre de cotes de K_(j-1).
    xy=[];
    for i=1:n
        P=XY(:,i:i+1);
        Q=Machinekoch(P);
        xy=[xy,Q(:,1:4)];
    end
    XY=[xy,Q(:,5)];
end
orthonorme();
plot2d(XY(1,:),XY(2,:),style=2,axesflag=0);
temps=toc();
afficher(temps);
```

Les calculs sont assez rapides jusque'à K_6 (moins de une seconde pour K_6). Le calcul de K_7 nécessite presque 20 secondes ; on peut remarquer à ce sujet que K_7 a 98304 côtés. Voici ce flocon :





9.4 [***] Visualisation du tri à bulles

On applique le tri à bulles à une suite donnée de n nombres $X = (x_1, \ldots, x_n)$ obtenue, pour fixer les idées, à l'aide de la commande `tirage_reel(n,0,1)`, n pouvant éventuellement varier. Représenter l'état de la suite X après chaque passage, y compris le dernier qui produit une illustration de la suite triée, en traçant, dans un repère orthonormé, les $(n-1)$ points (x_1, x_2), (x_2, x_3), ..., ainsi que la droite $y = x$.

Mots-clefs : `clf`, `sleep`, repère orthonormé, nuage de points : commande `plot(X,Y,"*b")`.

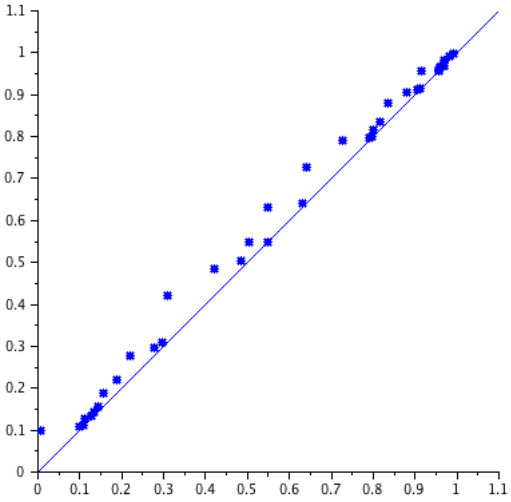
L'exercice 6.24 est consacré au tri à bulles que nous allons illuster maintenant, suivant l'exemple de [16]. Pour visualiser le tri à bulles, on peut utiliser le script commenté suivant :

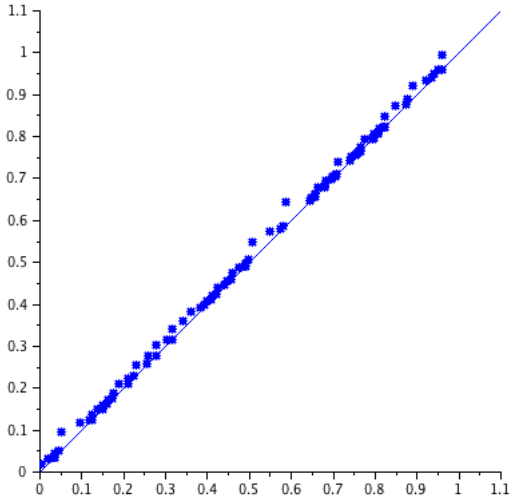
```
clear;// Effacement des variables non-protegees.
clf;// Effacement de la fenetre graphique.
n=input('n=');// Choix de n.
L=tirage_reel(n,0,1);
c=1;// Pour decrire au moins une fois la boucle suivante.
while c>0
    c=0;
    for k=1:(n-1)// K indexe les passages.
        for j=1:(n-k)
            if L(j)>L(j+1) then
                c=c+1;
                aux=L(j);
                L(j)=L(j+1);
                L(j+1)=aux;
            end
        end
        plot([0,1.1],[0,1.1],"b");// Graphe de y=x.
        a=gca();
        a.isoview="on";// Repere orthonorme.
        X=L(1:n-1)// Vecteur des abscisses.
        Y=L(2:n);// Vecteur des ordonnees.
        plot(X,Y,"*b")// Chaque point est une etoile bleue.
        sleep(1000);// On attend une seconde avant l'eventuel passage suivant.
        if c>0 then
            clf
        end
    end
end// Le dernier graphe qui apparait et reste represente la suite triee.
```

On obtient sur la console, en donnant la valeur 40 à n :

```
-->exec('Chemin du fichier charge', -1)
n=40
-->
```

Voici le graphe qui représente la suite triée issue de `X`, à gauche :

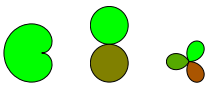




Remarques :

• Tous les points sont évidemment au-dessus du graphe de la droite $y = x$ parce que les nombres étant rangés dans l'ordre croissant, l'ordonnée est toujours supérieure ou égale à l'abscisse.

• S'il y a beaucoup de points (si n est grand), il y a peu d'écart entre l'abscisse et l'ordonnée des points représentés. On a l'impression qu'ils sont alignés sur la droite $y = x$. On peut alors faire usage du zoom. Le graphe de droite ci-dessus correspond au choix $n = 100$.



Chapitre 10

scilab vs Xcas

Liste des exercices :

Énoncé n° 10.1 : [*] Calcul numérique avec *scilab* et *Xcas* (test d'égalité de deux nombres).

Énoncé n° 10.2 : [*] Calculs en nombres rationnels avec *scilab* et avec *Xcas*.

Énoncé n° 10.3 : [*] Les fonctions racine carrée et élévation au carré sont-elles inverses l'une de l'autre ?

Énoncé n° 10.4 : [*] Calcul de factorielles.

Énoncé n° 10.5 : [*] Longueur d'un nombre entier.

Énoncé n° 10.6 : [**] Somme des chiffres d'un nombre entier.

Énoncé n° 10.7 : [*] Sommes de puissances des premiers entiers (démontrer une infinité d'égalités)

Énoncé n° 10.8 : [**] Test d'isocélité et d'équilatéralité d'un triangle.

Énoncé n° 10.9 : [***] Sommes d'inverses.

Ce qui suit est une comparaison très sommaire des *capacités de calcul et d'affichage* de *scilab* et de *Xcas*. Ce sont des logiciels de calcul très vastes qui, même si on se limite au calcul numérique, fonctionnent très différemment.

• *scilab* est perfectionné et étendu depuis plusieurs dizaines d'années. Il contient de nombreux développements spécialisés qui ne nous concernent pas. Il est employé notamment dans des bureaux d'étude industriels. Il est rapide et doué pour le calcul matriciel. Il travaille essentiellement en format scientifique standard. Par exemple, calculons $\left(\frac{\pi-1}{\pi+1}\right)^2$:

```
-->((%pi-1)/(%pi+1))^2
 ans  =
    0.2673861903466
-->b=ans^2
 b  =
    0.0714953747880
-->
```

• Même en calcul numérique pur, *Xcas* ne travaille pas du tout comme *scilab*. Il utilise ses capacités de calcul formel pour calculer en mode exact. Par exemple, recalculons les quantités précédentes avec *Xcas* :

```
a:=(%pi-1)/(%pi+1);b:=a**2;evalf(b);
```

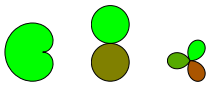
$$\frac{\pi-1}{\pi+1},\ \left(\frac{\pi-1}{\pi+1}\right)^2,\ 0.267386$$

(on peut afficher plus de décimales). On a d'abord obtenu b sous forme exacte puis en notation scientifique standard. Cette forme n'est pas une valeur exacte. C'est cette valeur approchée que *scilab* donne directement.

Xcas travaille en mode exact quand les données sont fractionnaires, ce qui est pratiquement toujours le cas au lycée, notamment en calcul des probabilités. C'est, pour le professeur, un avantage fabuleux.

scilab et *Xcas* n'ont pas non plus les mêmes *capacités d'affichage*. Un logiciel peut parfaitement calculer un nombre sans pouvoir l'afficher ou en l'affichant de manière déficiente. Par exemple, *scilab* affichera en notation scientifique standard un nombre entier qu'il vient de calculer exactement parce qu'il est trop grand. *Xcas* peut calculer tout nombre entier. Mais il y aura aussi des restrictions d'affichage. Les capacités de *Xcas* dans ce domaine sont impressionnantes.

Cette comparaison de *scilab* et de *Xcas* s'approfondira dans les exercices suivants. Il sera évidemment nécessaire de se reporter aux manuels. Nous recommandons [11] pour *scilab* (en plus de [12]) et [10] pour *Xcas*.



10.1 [*] Calcul numérique avec *scilab* et *Xcas* (test d'égalité de deux nombres)

Vérifier si *scilab* et *Xcas* donnent à $\tan\left(\frac{\pi}{4}\right)$ et à $\left(\sqrt{2}\right)^2$ les valeurs 1 et 2 respectivement.

Mots-clefs : Test « == ».

scilab

scilab ne peut pas vraiment tester l'égalité de deux nombres, sauf cas particulier. En effet, si deux nombres sont considérés par *scilab* comme des nombres réels et s'ils ne sont pas décimaux, ils ont une infinité de décimales. *scilab* n'en gardera qu'une partie en tronquant leurs développements décimaux. Sauf cas particulier, *scilab* ne donne donc pas aux nombres réels leur valeur exacte mais une valeur approchée. Cela amène à des erreurs. Voici deux exemples de test d'égalité de deux nombres où *scilab* se trompe :

```
-->tan(%pi/4)==1
 ans  =
    F
-->(sqrt(2))^2==2
 ans  =
    F
```

Ce constat est assez accablant. Mais quand on a un outil, on essaie de l'utiliser au mieux. On peut voir dans l'exercice 10.8 comment on teste quand même et de manière imparfaite l'égalité de 2 nombres avec *scilab*.

Xcas

Xcas ne calcule pas $\tan\left(\frac{\pi}{4}\right)$, sauf erreur, car ce logiciel contient les formules usuelles de trigonométrie ainsi que les valeurs remarquables des sin, cos, *etc.* Cela donne

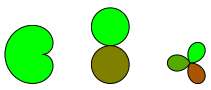
```
1 tan(%pi/4);
```

$$1$$

Xcas donnera aussi la valeur exacte de $\sqrt{2}^2$. En effet,

```
2 sqrt(2)^2;
```

$$2$$



10.2 [*] Calculs en nombres rationnels avec *scilab* et avec *Xcas*

Calculer $\dfrac{\frac{1}{2}+\frac{1}{3}+\frac{1}{4}}{\left(\frac{1}{2}\right)^2+\left(\frac{1}{3}\right)^2+\left(\frac{1}{4}\right)^2}$ avec *scilab* puis *Xcas*.

Mots-clefs : Calcul numérique, nombres rationnels.

Solution *scilab*

```
-->(1/2+1/3+1/4)/((1/2)^2+(1/3)^2+(1/4)^2)
 ans  =
     2.5573770491803
```

Solution *Xcas*

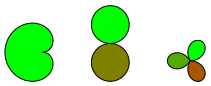
1 `a:=(1/2+1/3+1/4)/((1/2)^2+(1/3)^2+(1/4)^2);`

$$\frac{156}{61}$$

2 `evalf (a);`

$$2.557377$$

Xcas donne une solution exacte puis, à la demande (commande `evalf`), une solution approchée (on peut afficher plus de décimales). On pourrait obliger *scilab* à effectuer exactement le calcul, mais il n'est pas fait pour ça.



10.3 [*] Les fonctions racine carrée et élévation au carré sont-elles inverses l'une de l'autre ?

Calculer $\sqrt{x}^2$ et $\sqrt{x^2}$.

Mots-clefs : Commande `@` (composition des fonctions).

Solution *scilab*

Il n'y a pas de solution *scilab*. Tout au plus peut-on vérifier que pour des valeurs particulières de `x`, $\sqrt{x^2} = x$ et $\sqrt{x}^2 = x$.

Solution *Xcas*

Xcas sait que les fonctions considérées sont des fonctions réciproques :

3
```
f(x):=sqrt(x);
```

```
(x)->sqrt(x)
```

(f est définie comme la fonction « racine carrée »).

4
```
g(x):=x^2;
```

```
(x)->x^2
```

(g est définie comme la fonction « élévation au carré »)

5
```
(g@f)(x);
```

$$x$$

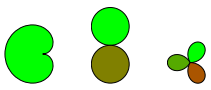
• L'utilisateur est sensé connaître le domaine de validité de ce résultat (à savoir $]0, +\infty[$).

6
```
(f@g)(x)
```

$$\text{abs}\,(x)$$

• L'utilisateur est sensé savoir que ceci est vrai pour tout réel x.

• Bien sûr, la fin de ce calcul est typiquement du calcul formel.



10.4 [*] Calcul de factorielles

Calculer $n!$ pour $n = 10, 20, 50, 100, 500, 1000$ à l'aide de *scilab* puis de *Xcas*.

Mots-clefs : calcul de grands nombres entiers, calcul exact, calcul approché, notation scientifique standard.

Calcul *scilab*

```
// Calcul et affichage de nombres factoriels.
clear;
format(25);//Cette commande permet l'affichage le plus long.
factorielle(10)
factorielle(20)
factorielle(23)
factorielle(24)
factorielle(50)
factorielle(100)
factorielle(500)
factorielle(1000)
```

Nous avons calculé en plus 23! et 24! parce que 23! est le plus grand nombre factoriel qui peut être affiché exactement (au format d'affichage le plus long), à savoir : $23! = 25852016738884978212864$. Les réponses montreront pourquoi nous avons aussi ajouté le calcul de 170! et de 171!

Voici la réponse de *scilab* à l'exécution de ce script :

```
-->// Calcul et affichage de nombres factoriels.
-->clear;
-->format(25);//Cette commande permet l'affichage le plus long.
-->factorielle(10)
 ans  =
     3628800.
-->factorielle(20)
 ans  =
     2432902008176640000.
-->factorielle(23)
 ans  =
     2585201673888497821 2864.
-->factorielle(24)
 ans  =
     6.20448401733239410 0D+23
-->factorielle(50)
 ans  =
     3.04140932017133755 8D+64
-->factorielle(100)
 ans  =
     9.33262154439441021D+157
 -->factorielle(170)
 ans  =
    7.25741561D+306
-->factorielle(171)
 ans  =
```

```
      Inf
-->factorielle(500)
 ans  =
     Inf
-->factorielle(1000)
 ans  =
     Inf
```

On peut imaginer (?) que *scilab* calcule exactement 24 !, 50 !, 100 !, 170 ! mais ne les affiche pas bien (passage en notation scientifique standard) et qu'à partir du calcul de 171 !, ses capacités ont été dépassées (les calculs de 500 ! et de 1000 ! étaient, par conséquent, inutiles). Au contraire, *Xcas* n'a pas de limitation de calcul en nombres entiers.

Calcul *Xcas*

1 `10!`

3628800

2 `100!`

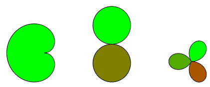
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000

3 `1000!`

402387260077093773543702433923003985719374864210714632543799910429938512398629020592044208486969404800479988610197196058631666872994808558901323829669944590997424504087073759918823627727188732519779505950995276120874975462497043601418278094646496291056393887437886487337119181045825783647849977012476632889835955735432513185323958463075557409114262417474349347553428646576611667797396668820291207379143853719588249808126867838374559731746136085379534524221586593201928090878297308431392844403281231558611036976801357304216168747609675871348312025478589320767169132448426236131412508780208000261683151027341827977704784635868170164365024153691398281264810213092761244896359928705114964975419909342221566832572080821333186116811553615836546984046708975602900950537616475847728421889679646244945160765353408198901385442487984959533191017233555566021394503997362807501378376153071277619268490343526252000158885351473316117021039681759215109077880193931781141945452572238655414610628921879602238397147608850627686296714667469756291123408243920816015378088989396451826324367161676217916890977991190375403127462228998800519544441428201218736174599264295658174662830295557029902432415318161721046583203678690611726015878352075151628422554026517048330422614397428693306169089796848259012545832716822645806652676995865268227280707578139185817888965220816434834482599326604336766017699961283186078838615027946595513115655203609398818061213855860030143569452722420634463179746059468257310379008402443243846565724501440282188525247093519062092902313649327349756551395872055965422874977401141334696271542284586237738753823048386568897646192738381490014076731044664025989949022222176590433990188601856652648506179970235619389701786004081188972991831102117122984590164192106888438712185564612496079872290851929681937238864261483965738229112312502418664935314397013742853192664987533721894069428143411852015801412334482801505139969429015348307764456909907315243327828826986460278986432113908350621709500259738986355427719674282224875758676575234422020757363056949881626

2508796892816275384886339690995982628095612145099487170124451646126037902930912088
9086942028510640182154399457156805941872748998094254742173582401063677404595741785
1608292301353580818400969963725242305608559037006242712434169090041536901059339838
3577793941097002775347200000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000

C'est étonnant !



10.5 [*] Longueur d'un nombre entier

Avec combien de chiffres s'écrit 500 ! ?

Mots-clefs : calcul de grands nombres entiers.

Il vaut mieux programmer la solution du problème plus général : avec combien de chiffres s'écrit un entier naturel $A \geqslant 1$ donné ?

Solution *scilab*

L'exercice 10.4 laisse prévoir que *scilab* n'aura pas de problème tant que $A \leqslant 170$ et qu'il sera impuissant sinon.

```
-->// La fonction qui suit donne le nombre de chiffres avec lesquels
//s'ecrit un entier donne A.
-->function nn=NC(A);
-->     n=0;
-->while 10^n<=A
-->     n=n+1;
-->end
-->nn=n;
-->afficher("A s'ecrit avec "+string(nn)+" chiffres");
-->endfunction
```

On charge cette fonction, puis on l'applique pour diverses valeurs de A :

```
-->NC(factorielle(23))
 A s'ecrit avec 23 chiffres
 ans  =
    23.
 -->NC(factorielle(24))
  A s'ecrit avec 24 chiffres
 ans  =
    24.
-->NC(factorielle(50))
  A s'ecrit avec 65 chiffres
 ans  =
     65.
-->NC(factorielle(100))
  A s'ecrit avec 158 chiffres
 ans  =
    158.
-->NC(factorielle(150))
  A s'ecrit avec 263 chiffres
 ans  =
    263.
-->NC(factorielle(160))
   A s'ecrit avec 285 chiffres
 ans  =
    285.

-->NC(factorielle(170))
```

```
A s'ecrit avec 307 chiffres
 ans  =
    307.
-->NC(factorielle(171))
  Abandon du calcul.
```

Le calcul de 171 ! a été abandonné au bout de 6 minutes.

Solution *Xcas*

1

```
// Avec combien de chiffres s'ecrit 500!?
NombredeChiffres(A):={
local n;
n:=1;
tantque 10^n<=A faire
n:=n+1;
ftantque;
retourne n;
}:;
```

// Parsing NombredeChiffres
// Success compiling NombredeChiffres

Done

2 `NombredeChiffres(170!)`

307

3 `NombredeChiffres(200!)`

375

4 `NombredeChiffres(500!)`

1135

5 `NombredeChiffres(1000!)`

2568

6 `NombredeChiffres(5000!)`
Evaluation time : 2.34

16326

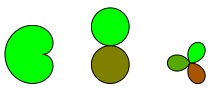
7 `NombredeChiffres(10000!)`
Evaluation time : 14.54

35660

8 `NombredeChiffres(15000!)`
Evaluation time : 42.8

56130

On peut ajouter que le calcul du nombre de chiffres de 171 ! est instantané.



10.6 [**] Somme des chiffres d'un nombre entier.

Un entier $n \geqslant 1$ étant donné, calculer son chiffre des unités noté $u(n)$ et la somme de ses chiffres notée $s(n)$.
Application : démontrer que $u\left(\frac{3^{2014}-1}{2}\right) = s\left(s\left(s\left(s\left(\frac{3^{2014}-1}{2}\right)\right)\right)\right)$.

Mots-clefs : calcul de grands nombres entiers.

Solution *scilab*

• On ne peut calculer `s(n)` avec *scilab* que si `n` est donné, bien entendu. Aussi, dans le script ci-dessous, on commence par demander la valeur de `n`. Ce script est élémentaire. Il donne au passage le nombre de chiffres de `n`, autrement dit, il résout au passage l'exercice 10.5.

```
n=input('n=');
r=reste(n,10);//chiffre des unites de n.
C=[];//future liste des chiffres de n.
while n>0
    r=reste(n,10);
    C=[C,r];
    n=(n-r)/10;
end
t=taille(C);
afficher('Le nombre de chiffres de n est '+string(t))
s=sum(C);
afficher('La somme des chiffres de n est '+string(s))
```

• Appliquons ce script quand `n` vaut $\frac{3^{2014}-1}{2} = s\left(s\left(s\left(s\left(\frac{3^{2014}-1}{2}\right)\right)\right)\right)$ [1]. Cela donne :

```
->exec('Chemin du script', -1)
n=(3^2014-1)/2
 !--error 10000
Les arguments de la fonction "reste" doivent etre des entiers.
at line       5 of function reste called by :
r=reste(n,10);//chiffre des unites de n.
at line       2 of exec file called by :
'Chemin du script', -1
->
```

À la suite de ce message d'erreur bizarre, calculons `n`. On obtient :

```
->n=(3^2014-1)/2
 n  =
    Inf
->
```

Effectivement, `inf` n'est pas un nombre entier.

• On peut démontrer que $u\left(\frac{3^{2014}-1}{2}\right) = s\left(s\left(s\left(s\left(\frac{3^{2014}-1}{2}\right)\right)\right)\right)$ de manière presque purement arithmétique. Évidemment, cela demande de l'astuce. En fait, la bonne solution (la plus facile) de ce problème est la solution *Xcas* :

1. Ceci est un exercice de la rubrique « De-ci, de-là » du Bulletin vert n°510 posé par Michel Lafond.

Solution *Xcas*

Nous proposons l'algorithme suivant qui donne la somme des chiffres, le nombre des chiffres (dont on n'a pas besoin) et le chiffre des unités de `n`.

1

```
SommedesChiffres(n):={
  local s,C,r,u,l;
  s:=0;//Future somme des chiffres de n.
  C:=[];//Future liste des chiffres de n.
  tantque n>0 faire
    r:=irem(n,10);// Chiffre successifs.
    u:=r;// Chiffre des unites de n.
    C:=append(C,r);
    s:=s+r;
    n:=(n-r)/10;
  ftantque;
  l:=size(C);
retourne(s,l,u);
}:;
```

// Parsing SommedesChiffres
// Success compiling SommedesChiffres

Done

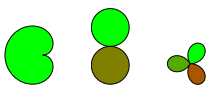
2 `SommedesChiffres((3^2014-1)/2);`

$$4306, 961, 4$$

3

On applique ensuite cet algorithme à $n = \frac{3^{2014}-1}{2}$. On trouve que la somme de ses chiffres est 4306 et que 4 est son chiffre des unités. On calcule enfin de tête la somme des chiffres de 4306 qui est 13, puis la somme des chiffres de 13 qui est 4 et la somme des chiffres de 4 qui est 4, ce qui est bien le chiffre des unités de `n`.

En se basant sur cet algorithme *Xcas*, on peut se poser d'autres problèmes arithm tiques.



10.7 [*] Sommes de puissances des premiers entiers (démontrer une infinité d'égalités)

Peut-on démontrer que $1+\cdots+n=\dfrac{n\,(n+1)}{2}$ avec *scilab* ? avec *Xcas* ?
Quelles formules donnent la somme des carrés ? des cubes ? des puissances suivantes ?

Mots-clefs : Calcul formel.

Solution *scilab*

scilab ne peut trouver la formule demandée mais on peut chercher à la vérifier. Cependant, comme n prend une infinité de valeurs, il faudrait effectuer une infinité de calculs, ce qui est impossible. La formule donnée n'est donc pas démontrable avec *scilab*. Le script ci-dessous permet de vérifier l'égalité quand n varie de p à q :

```
//Quelques verifications de la formule 1+..+n=n(n+1)/2.
function S=SommesDentiers(p,q)
    s=sum(linspace(1,p-1,p-1));// s=1+...+p-1.
    test=1;
    for j=p:q
        s=s+j;
        if 2*s<>j*(j+1) then
            test=0;
            S='La formule est fausse';
            break;
        end
    end
    if test==1 then
        S='La formule est exacte pour n variant de '+string(p)+' a '
        +string(q);
    end;
endfunction
```

Vérifions ainsi que la formule est exacte quand $n \leqslant 10^6$:

```
-->SommesDentiers(1,1000000)
 ans  =
 La formule est exacte pour n variant de 1 a 1000000
```

Solution *Xcas*

Nous avons calculé ci-dessous successivement $\sum_{k=1}^{n} k$, $\sum_{k=1}^{n} k^2$, $\sum_{k=1}^{n} k^3$, $\sum_{k=1}^{n} k^4$, $\sum_{k=1}^{n} k^5$, $\sum_{k=1}^{n} k^{10}$, $\sum_{k=1}^{n} k^{20}$. Les formules obtenues doivent être considérées comme démontrées[2]. Le calcul de $\sum_{k=1}^{n} k^{50}$, trop long, a été abandonné. La première formule est en général connue. La deuxième et la troisième sont un peu moins connues. Elles peuvent toutes être démontrées à la main par récurrence, à condition de deviner la formule de récurrence !

2. Ceux qui douteraient pourraient utiliser les formules données ci-dessus par *Xcas* pour faire une démonstration par récurrence dans les règles de l'art. Ce serait facile car le plus difficile est justement de deviner la relation de récurrence.

1 `factor(simplify(sum(n,n,1,n)));`
n : recursive definition

$$\frac{n(n+1)}{2}$$

2 `factor(simplify(sum(n^2,n,1,n)))`
n : recursive definition

$$\frac{n(n+1)(2\cdot n+1)}{6}$$

3 `factor(simplify(sum(n^3,n,1,n)))`
n : recursive definition

$$\frac{n^2\,(n+1)^2}{4}$$

4 `factor(simplify(sum(n^4,n,1,n)))`
n : recursive definition

$$\frac{n(n+1)(2\cdot n+1)(3n^2+3\cdot n-1)}{30}$$

5 `factor(simplify(sum(n^5,n,1,n)))`
n : recursive definition

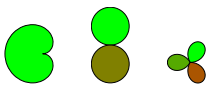
$$\frac{n^2\,(n+1)^2\,(2n^2+2\cdot n-1)}{12}$$

6 `factor(simplify(sum(n^10,n,1,n)))`
n : recursive definition
Evaluation time : 0.45

$$\frac{n(n+1)(2\cdot n+1)(n^2+n-1)(3n^6+9n^5+2n^4-11n^3+3n^2+10\cdot n-5)}{66}$$

7 `factor(simplify(sum(n^20,n,1,n)))`
n : recursive definition
Evaluation time : 31.64

$$\frac{n(n+1)}{6930}(2\cdot n+1)(165n^{18}+1485n^{17}+3465n^{16}-5940n^{15}-25740n^{14}+41580n^{13}+163680n^{12}$$
$$-266310n^{11}-801570n^{10}+1335510n^9+2806470n^8-4877460n^7-6362660n^6+11982720n^5$$
$$+7591150n^4-17378085n^3-1540967n^2+11000493\cdot n-3666831)$$



10.8 [**] Test d'isocélité et d'équilatéralité d'un triangle *

Écrire un script *scilab* et un script *Xcas* qui indiquent si un triangle donné est équilatéral, isocèle (sans être équilatéral) ou quelconque (c'est à dire non-isocèle).

Mots-clefs : Test « == », test conditionnel.

Tester si un triangle ABC donné est isocèle ou équilatéral revient à comparer les longueurs de ses côtés. On a donc à tester plusieurs fois l'égalité de deux nombres.

Script *scilab*

Le script *scilab* ci-dessous permet de saisir les coordonnées des sommets de ABC, calcule les carrés des longueurs AB, BC et CA notés x, y et z et teste de deux façons l'égalité de ces longueurs.

• Comparer les carrés des longueurs des côtés permet d'éviter le calcul de racines carrées, qui induirait de nouvelles approximations.

• Le premier test serait impeccable si *scilab* était capable de calculer ces longueurs exactement. Comme ce n'est pas le cas, il se trompe quand on l'applique aux points $A = [-23/15, -36/35]$, $B = [13/15, 76/35]$ et $C = [-89/15, -167/35]$ (parce que ABC est un triangle isocèle).

• *Le deuxième test considère que $x = y$ si $|x - y|/(x + y)$ est < %eps (et de même pour les autres couples de côtés), autrement dit si le nombre $|x - y|/(x + y)$ est considéré comme nul par scilab, ce qui ne veut pas dire qu'il est effectivement nul.*

• %eps est très petit : c'est en effet le plus petit nombre > 0 reconnu par *scilab*.

• %eps=2.220446049D-16, voir un manuel.

```
A=input("Coordonnees de A = "); // sous la forme [abscisse, ordonnee]
B=input("Coordonnees de B = "); // idem
C=input("Coordonnees de C = "); // idem
x=(B(1)-A(1))^2+(B(2)-A(2))^2; // x=AB^2
y=(C(1)-A(1))^2+(C(2)-A(2))^2; //y=AC^2
z=(C(1)-B(1))^2+(C(2)-B(2))^2; //z=BC^2
// Test ne prenant pas en compte les troncatures de reels
if x==y & y==z then
    disp("ABC est equilateral")
elseif x==y |y==z |z==x then
    disp("ABC est isocele")
else
    disp("ABC n''est ni equilateral, ni isocele")
end
// Test tenant compte des troncatures de reels et de %eps
if abs(x-y)/(x+y)<%eps & abs(y-z)/(y+z)<%eps then
    disp("ABC est equilateral")
elseif abs(x-y)/(x+y)<%eps |abs(y-z)/(y+z)<%eps |abs(z-x)/(z+x)<%eps then
    disp("ABC est isocele")
else
    disp("ABC n''est ni equilateral, ni isocele")
end
```

Voici ce que répond *scilab* pour le triangle ABC défini ci-dessus :

```
//Test d'isocelite et d'equilateralite d'un triangle ABC
-->A=input("Coordonnees de A = ");
```

```
Coordonnees de A = [-23/15,-36/35]
-->B=input("Coordonnees de B = ");// idem
Coordonnees de B = [13/15,76/35]
-->C=input("Coordonnees de C = ");// idem
Coordonnees de C = [-89/15,167/35]
-->// Test ne prenant pas en compte les troncatures de reels
 ABC n'est ni equilateral, ni isocele
-->// Test tenant compte des troncatures de reels et de %eps
 ABC est isocele
```

La réponse du premier test est fausse, la réponse du second test est exacte mais à prendre avec réserve parce que l'égalité des longueurs est testée de manière approximative.

Script *Xcas*

```
NatureDuTriangle( a,b,c,d,f,g):={
// a,b,c,d,f,g signifie x_A,y_A,x_B,y_B,x_C,y_C
local u,v,w;
u:=(a-c)^2+(b-d)^2;v:=(c-f)^2+(d-g)^2;w:=(f-a)^2+(g-b)^2;
if simplify(u)==simplify(v) and simplify(v)==simplify(w) then
    afficher("Ce triangle est \'{e}quilat\'{e}ral");
elif simplify(u)==simplify(v)or
 simplify(v)==simplify(w)or
simplify(w)==simplify(u)then
    afficher("Ce triangle est isocele");
else
    afficher("Ce triangle n'est ni equilateral, ni isocele");
end
return simplify(u),simplify(v),simplify(w);}
 :;
```

// Parsing NatureDuTriangle
// Success compiling NatureDuTriangle

Done

2 `NatureDuTriangle(-23/15,-36/35,13/15,76/35,-89/15,167/35)`
Ce triangle est isocèle

$$16, 53, 53$$

3 `NatureDuTriangle(evalf(-23/15,-36/35,13/15,76/35,-89/15,167/35))`
Ce triangle est isocèle

$$16.000000, 53.000000, 53.000000$$

Dans ce calcul, les coordonnées rationnelles des points A, B et C ont tout de suite été transformées en nombres réels. Il en résulte que *Xcas* a travaillé en nombres réels mais répond correctement.

4 `NatureDuTriangle(-1/2,2/3,9/10,77/15,-24/5,-7/8*17^(1/7))`
Ce triangle n'est ni équilatéral, ni isocèle

$$\frac{986}{45}, \frac{11025\left(17^{\frac{1}{7}}\right)^2+129360\cdot 17^{\frac{1}{7}}+847312}{14400}, \frac{11025\left(17^{\frac{1}{7}}\right)^2+16800\cdot 17^{\frac{1}{7}}+272656}{14400}$$

La réponse est exacte. Pour s'en convaincre, voir l'essai suivant.

5 `NatureDuTriangle(evalf(-1/2,2/3,9/10,77/15,-24/5,-7/8*17^(1/7)))`
Ce triangle n'est ni équilatéral, ni isocèle

$$21.911111, 74.026584, 22.403361$$

Dans ce calcul, les coordonnées rationnelles des points A, B et C ont tout de suite été transformées en nombres réels. *Xcas* a travaillé en nombres réels. On constate que les 3 longueurs sont différentes.

6 `NatureDuTriangle(4,3,6,5,5-sqrt(3),4+sqrt(3))`
Ce triangle est équilatéral

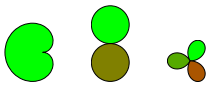
$$8, 8, 8$$

Xcas n'a pas transformé $\sqrt{3}$ en nombre réel. Il a calculé formellement. Le résultat est exact et est exprimé en nombres entiers.

7 `NatureDuTriangle(evalf(4,3,6,5,5-sqrt(3),4+sqrt(3)))`
Ce triangle est équilatéral

$$8.000000, 8.000000, 8.000000$$

On a obligé à travailler *Xcas* en nombres réels. Le résultat est exact et est exprimé en nombres réels.



10.9 [***] Sommes d'inverses *

On calcule $s_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$ pour des valeurs croissantes de n.
1 - Pour quelle valeur de n *scilab* cesse-t-il de donner un résultat exact ?
2 - Même question pour *Xcas*.

Mots-clefs : calcul exact, calcul approché, calcul en nombres rationnels, notation scientifique standard.

Solution *scilab*

scilab considérant par défaut tous les nombres comme des réels, le résultat retourné n'est pas exact dès $n = 3$, mais approché, exprimé comme un nombre décimal (de 16 signes dont le + qui n'apparaît pas, cf. la commande `format(16)`), à cause du $\frac{1}{3}$. Ci-dessous, on a calculé s_n pour n valant 10, 10^2, ..., 10^6 à l'aide de l'algorithme :

```
function y=SomInv(n)
    y=0;
    for j=1:n
        y=y+1/j;
    end
endfunction;
S=[];
format(16);
for j=1:6
    S=[S,SomInv(10^j)];
end
disp(S)
```

scilab a retourné les résultats suivants :

```
        column 1 to 4

 2.9289682539683    5.1873775176396    7.4854708605503    9.7876060360443

        column 5 to 6

 12.090146129863    14.392726722865
```

Solution *Xcas*

Dans le programme *Xcas* qui suit, $f(n)$ désigne la somme $1 + \frac{1}{2} + \cdots + \frac{1}{n}$.

1

```
f(n):={
local j,s;
s:=0;
pour j de 1 jusque n faire
s:=s+1/j;
fpour;
retourne s;
}:;
```

// Parsing f
// Success compiling f

Done

Nous avons calculé successivement $f(100)$, $f(1000)$, $f(10000)$ et $f(100000)$. On obtient des résultats exacts sous forme de fraction (sauf $f(100000)$ qui n'a pas pu être affiché) dont on a demandé une évaluation (à l'aide de la commande `evalf`).

2 `s100:=f(100)`

$$\frac{1446663627952035116022151804310413144771 1}{2788815009188499086581352357412492142272}$$

3 `evalf(s100)`

5.1873775176396202

4 `s1000:=f(1000)`

Xcas retourne une fraction `s1000` dont le numérateur est

533629132822947850455910456240429804096524722803842600971013492484562688894971017575060979019850356914090887315504680983784421721178850094643023443265660225021002784256328520814055449412104425101426727702947747127089179639677796104532246924268664688882815820719848971051107968732493191555293970175089315645199760857344730141832840117244122806490743077037366831700558002936592350885893602352858528081607595747378366554131755081315225 17

et le dénominateur est

712886527466509305316638415571427292066835886188589304045200199115432408758111149947644415191387158691171781701957525651298026406762100925146587100430513107268626814320019660997486274593718834370501543445252373974529896314567498212823695623282379401106880926231770886197954079124775455804932647573782992335275179673524804246363805113703433121478174685087845348567802188807537324992199567205693202909939089168748767269795093160352000 0

5 `evalf(s1000)`

7.4854708605503451

6 `s10000:=f(10000)`
Evaluation time : 1.18

$$\frac{\text{Integer too large for display}}{\text{Integer too large for display}}$$

7 `evalf(s10000)`

(10.9.1) 9.7876060360443820

8 `s10000:=f(100000)`
Evaluation time : 91.91

$$\frac{\text{Integer too large for display}}{\text{Integer too large for display}}$$

9 `evalf(s100000)`

(10.9.2) s100000

On constate que

• - Pour $n = 10000$, la valeur exacte de la somme est calculée mais ne s'affiche pas ; une valeur approchée est donnée sous forme décimale. On pourrait demander plus de décimales.

• - Pour $n = 100000$, la valeur exacte de la somme est calculée mais ne s'affiche pas ; *Xcas* ne donne pas non plus de valeur approchée.

• - Il est difficile d'après ces tests numériques d'imaginer que $s_n \xrightarrow[n\to+\infty]{} +\infty$!

• - *scilab* est meilleur ici que *Xcas*. Il donne des valeurs approchées des termes de rang 10000, 100000 et 1000000 de la suite $(f(n),\, n \geqslant 1)$.

• - Quand on étudie la convergence d'une suite numérique avec très peu de connaissances mathématiques, une idée naturelle est de calculer ses premiers termes et de voir si on peut conjecturer le comportement asymptotique de ladite suite. On voit qu'ici ça ne marche pas du tout. Après avoir ajouté les inverses des entiers de 1 à 10^6, on obtient moins de 15. Difficile d'imaginer que la suite tend vers $+\infty$.

• - à noter (en exécutant soi-même l'algorithme) la rapidité des calculs de *scilab*.

• - Un peu de réflexion montre que la contre-performance de « Xcas » est due au fait que le calcul a été fait en nombres rationnels. Si on supprime cette contrainte, quitte à perdre l'exactitude des résultats, on retrouve des performances comparables à celles de *scilab*. Il suffit d'utiliser la commande `evalf` :

1

```
f(n):={
local j,s;
s:=0;
pour j de 1 jusque n faire
s:=evalf(s+1/j);
fpour;
retourne s;
}:;
```

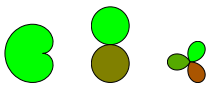
// Parsing f
// Success compiling f

Done

2 `f(1000000)`
Evaluation time : 16.24

$$0.14392726722865995e2$$

La réponse a été donnée en notation scientifique standard. Il s'agit de 14,392726722865995 sous forme décimale.



10.10 [****] Table ronde *

Quelle est la probabilité p pour que n couples placés au hasard autour d'une table ronde soient tous séparés ?

Mots-clefs : très grands nombres entiers.

• On considérera que deux configurations sont identiques quand chacune des $2n$ personnes a le même voisin à droite et le même voisin à gauche. Il s'agit des configurations qui se déduisent l'une de l'autre par une rotation autour de la table ronde.

• Ce difficile problème de combinatoire connu comme « le problème des ménages » a reçu une solution ainsi que des problèmes voisins comme « Quelle est la probabilité pour que les sexes soient alternés quand on place n couples au hasard autour d'une table ronde (à couples séparés ou non ? ».

• Nous allons tenter une solution triviale qui consiste à écrire toutes les configurations possibles et à compter celles qui ne séparent pas tous les couples.

• Appelons les participants $1, -1, 2, -2, \ldots, n, -n$. Nous verrons plus loin l'intérêt de ce codage.

• Pour fabriquer toutes les configurations possibles, nous commencerons toujours par placer 1 devant une chaise donnée, puis nous placerons les $2n-1$ personnes restantes en tournant autour de la table, dans le sens direct par exemple. Cela montre que le nombre total de configurations est $N = (2n-1)!$ (c'est un nombre d'arrangements).

• Bien sûr, ces configurations sont équiprobables puisque les personnes sont placées au hasard autour de la table. C'est pourquoi ce problème est un problème de combinatoire.

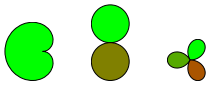
Solution *scilab*

• Dans l'algorithme *scilab* ci-dessous, nous partons de la suite $2, \ldots, n, -n, \ldots, -1$ dont nous écrivons toutes les permutations à l'aide de la commande « perms », puis nous écrivons toutes les configurations possibles en ajoutant à gauche une colonne de 1, *etc.*

```
n=input('n=');
stacksize('max');
Assemblee=[2:n,-n:-1];
N=factorielle(2*n-1);
A=[ones(N,1),perms(Assemblee)];// Perms(Assemblee) est une matrice a N
// lignes et 2n-1 colonnes.
B=[A(:,2:2*n),ones(N,1)];
C=A+B;// Si une ligne de C contient un zero, un couple de la
// configuration concernee n'est pas separe.
compteur=0;
for i=1:N
    if prod(C(i,:))==0 then
        compteur=compteur+1;
    end;
end;
p=1-compteur/N;
afficher("La probabilite pour que les "+string(n)+" couples soient
separes est egale a "+string(p));
```

Cet algorithme est correct mais ne pourra s'appliquer qu'à de petites valeurs de n car il nécessite énormément de calcul puisqu'il utilise N boucles « pour » contenant un test.

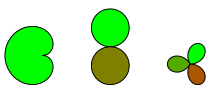
n	1	2	3	4	5	6
N	1	6	120	5040	362880	39916800
p	0	1/3	0.267	0.295	0.310	?



Chapitre 11

Plus de scilab

Liste des exercices :



11.1 [****] Crible d'Ératosthène avec *scilab* et *Xcas*

On demande une fonction *scilab* qui, à tout nombre entier $n \geqslant 2$ associe la liste des nombres premiers qui lui sont inférieurs ou égaux, rangés dans l'ordre croissant.

Mots-clefs : boucles `tant que`, commandes `taille`, `pmodulo`, `find`, `reste`, `liste_premiers` ou `primes`, `A($)`.

Cet exercice (et quelques exercices suivants) concerne les nombres premiers, plus généralement l'arithmétique. C'est une branche extraordinaire des mathématiques qui impose des calculs en nombres premiers. Les nombres premiers, concernés par le crible d'Ératosthène, restent assez mystérieux. Par exemple, si on les note $p_1 = 2$, $p_2 = 3$, p_3, ..., il n'existe pas de formule qui, étant donné n, permette de calculer p_n.

Solution n° 1 : la commande `liste_premiers(n)` (`n` spécifié, $\geqslant 2$) fournit en sortie la liste des nombres premiers compris au sens large entre 2 et `n` et répond donc exactement au problème posé. C'est une commande de *scilab pour les lycées*. Elle est identique à la commande `primes` de *scilab*. L'argument de ces commandes ne peut pas être trop grand. Sinon, `scilab` envoie le message d'erreur : `Taille de la pile dépassée`.

Exemples : pour $n = 100$,

```
-->liste_premiers(100)
 ans  =
         column  1 to 11
    2.    3.    5.    7.    11.    13.    17.    19.    23.    29.    31.
         column 12 to 21
    37.    41.    43.    47.    53.    59.    61.    67.    71.    73.
         column 22 to 25
    79.    83.    89.    97.
 -->
```

On voit qu'il y a 25 nombres premiers entre 2 et 100 et que $p_{25} = 97$. Pour $n = 10^7$, puis $n = 10^8$:

```
-->clear
-->primes(10000000);
-->size(ans,'c')
 ans  =
    664579.
-->clear;stacksize('max');primes(100000000);
 !--error 17
Taille de la pile depassee
Utilisez la fonction stacksize pour l'augmenter.
Memoire utilisee pour les variables  : 100009787
Memoire intermediaire requise  : 200000006
Memoire totale disponible  : 268435455
at line       31 of function primes called by :
clear;stacksize('max');primes(100000000);
-->
```

On lit qu'il y a 664579 nombres premiers inférieurs ou égaux à 10^7.

Solution n° 2 : crible d'Ératosthène À la main, lorsque `n` est petit, on détermine habituellement cette liste à l'aide du crible d'Ératosthène[1]. Tant que `n` n'est pas trop grand, on pourra utiliser la fonction

1. Le crible d'Ératosthène supposé connu.

`Erato.sci` ci-dessous dont l'algorithme est explicite mais qui est moins rapide que `liste_premiers(n)`, comme on le constatera plus tard.

Le crible d'Ératosthène est typiquement un algorithme.

Objet du crible d'Ératosthène : le crible d'Ératosthène associe à tout entier n supérieur ou égal à 2 la liste des nombres premiers inférieurs ou égaux à n.

Description du crible d'Ératosthène : il consiste en un certain nombre de passages au crible, nombre contrôlé par une boucle `tant que`.

• A désignant la liste des entiers de 2 à n, on pose $k = 1$.

• *Début de la boucle* : Si $A(k)^2 \leqslant A(\$)$ et tant que cette inégalité reste vraie, on efface de A les multiples stricts de $A(k)$ (il y en a au moins un, à savoir $A(k)^2$). On continue d'appeler A la liste ainsi obtenue, on remplace k par $k + 1$ et on revient au début de la boucle.

• *Fin de la boucle*

• Si B désigne l'état final de A, B est la liste des nombres premiers $\leqslant n$.

Justification : On sait que $B(1) = 2$ est premier. De plus, si l'on a démontré que $B(1), B(2), \ldots, B(k-1)$ sont des nombres premiers, ce sont les seuls nombres premiers $< B(k)$ parce que tout nombre qui a été effacé est un multiple, donc n'est pas premier. Si $B(k)$ n'était pas premier [2], il serait un multiple de l'un des nombres $B(1), B(2), \ldots, B(k-1)$, ce qui est absurde car il aurait été effacé. Le même raisonnement montre que le premier multiple de $B(k)$ qui ne peut pas avoir été effacé est $B(k)^2$. Par conséquent, on peut arrêter la boucle si $B(k)^2 > A(\$)$ car on n'effacera plus rien. Finalement, B est bien la liste des nombres premiers $\leqslant n$.

La fonction téléchargeable `Erato.sci` ci-dessous reproduit exactement ce crible.

Remarques sur le script :

• Pour repérer la liste `I` des rangs des multiples d'un entier `a` dans une liste d'entiers `L`, on utilise la commande `find` et la commande `pmodulo`, ligne 5.

• La commande `pmodulo` est une commande de *scilab* qui contrairement à `reste` admet un vecteur d'entiers comme premier argument. `reste` est une commande de *scilab pour les lycées*.

• On efface les termes de `L` dont le rang appartient à `I` à l'aide de la commande `L(I)=[ ]` (ligne 10).

Listing 11.1 – Erato.sci

```
function A=Erato(n)
A=2:n;// n entier >=2.
k=1;// k compte les nombres premiers decouverts.
while A(k)^2<=A($)
    I=find(pmodulo(A(k+1:taille(A)),A(k))==0);
    // On repere les multiples de A(k) apres A(k).
    A(k+I)=[];// On a efface les multiples de A(k).
    k=k+1;
end;
endfunction
```

On s'attend bien sûr à ce que `liste_premiers` soit plus rapide que `Erato.sci` (et s'applique à des entiers `n` plus grands). Calculons le temps de calcul de la liste des nombres premiers $\leqslant 10^6$ à l'aide de `Erato.sci`, puis à l'aide de `liste_premiers` :

```
-->clear
-->exec('Chemin de Erato.sci', -1)
-->tic();A=Erato(1000000);temps=toc();afficher(temps);
    2.612
-->tic();A=liste_premiers(1000000);temps=toc();afficher(temps);
```

2. Tout entier $\geqslant 2$ est soit premier, soit divisible par un nombre premier strictement plus petit que lui.

```
    0.122
-->
```

Le plus grand nombre premier $\leqslant 10^6$ est :

```
-->A($)
 ans  =
    999983.
-->
```

Si l'on tente $n = 10^7$, il faut augmenter la taille de la mémoire affectée au calcul quand on se sert de la fonction `Erato.sci` ou de la commande `primes` :

```
-->clear
-->exec('Chemin de Erato.sci', -1)
-->stacksize('max');tic();A=Erato(10000000);temps=toc();afficher(temps);
    62.529
-->clear
-->tic();A=liste_premiers(10000000);temps=toc();afficher(temps);
    1.279
-->
```

On obtient aussi :

```
-->taille(A)
 ans  =
    664579.
-->A($)
 ans  =
    9999991.
-->
```

On constate que le crible d'Ératosthène a établi la liste des nombres premiers inférieurs à 10^7 en 63 secondes, la commande `liste_premiers` en 1.3 secondes. Il y a 664579 nombres premiers $\leqslant 10^7$. Le plus grand est $p_{664579} = 9999991$.

La conclusion attendue est qu'il vaut lieux utiliser la commande `liste_premiers` ou la commande `primes` !

Notes diverses

On sait qu'il y a une infinité de nombres premiers, depuis longtemps, voir les Élements d'Euclide ([18]). Ces nombres deviennent très vite difficiles à calculer. Le 25 janvier 2013, le plus grand nombre premier connu était $p = 2^{57\,885\,161} - 1$, qui s'écrit avec 17 425 170 chiffres. L'exposant 57 885 161 est lui-même un nombre premier (voir l'exercice 11.2), si bien que p est un nombre de Mersenne (les nombres premiers records sont souvent des nombres de Mersenne).

Établir qu'un nombre est premier nécessitait autrefois de nombreuses divisions à la main. Les progrès sont venus de l'utilisation de machines : calculatrices mécaniques, ordinateurs et enfin réseaux d'ordinateurs.

Solution n°3, avec *Xcas*

Nous ne détaillerons pas cette solution qui consiste en une fonction-*Xcas* appelée encore `Erato` qui est testée ci-dessous pour calculer la liste des nombres premiers de 2 à `n`, `n` prenant successivement les valeurs 10^2, 10^3, 10^4, 10^5 10^6 et 10^7. Elle n'a pas pu venir à bout du cas $n = 10^8$. La fonction `crible` de [8], qui a le même rôle, paraît moins bonne qu'`Erato`.

1

```
Erato(n):={
local A,j,k,t,a,B,x;
```

```
A:=[j$(j=2..n)];
k:=0;
t:=length(A)-1;
while(A[k]^2<=A[t]){
a:=A[k];
B:=A[(k+1)..t];
B:=remove(x->(irem(x,a)==0),B);
A:=concat(A[0..k],B);
k:=k+1;
t:=length(A)-1;
}
return A;
}
 :;
```

// Warning : a, declared as global variable(s)
// Parsing Erato
// Success compiling Erato

Done

2 `P100:=Erato(100);`

$$[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]$$

3 `P1000:=Erato(1000);`

$[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]$

4 `P10000:=Erato(10000);`

Done

5 `P100000:=Erato(100000);`
Evaluation time : 6.43

Done
6 `l:=length(P100000);P100000[l-1];`

$$9592, 99991$$

7 `P1000000:=Erato(1000000);`
Evaluation time : 126.29

Done

8 `r:=length(P1000000);P1000000[r-1];`

$$78498,999983$$

9 `P10000000:=Erato(10000000);`
Evaluation time : 2682.93

Done
10 `s:=length(P10000000);P10000000[s-1];`

$$664579,9999991$$

11

```
//renvoie la liste des nombres premiers<=n selon eratosthene
crible(n):={
  local tab,prem,p,j;
  //on ecrit dans trab la liste des entiers de 0 a n
  tab:=makelist(x->x,0,n);
  //1 n'est pas premier donc on met 0 ds tab[1]
  tab[1]:=0;
  p:=2;
  //on barre les multiples de p qui sont compris entre p*p et n
  //en mettant 0 dans la case correspondante
  tantque (p*p<=n) faire
    pour j de p*p jusque n pas p faire
      tab[j]:=0;
    fpour;
    p:=p+1;
    //on cherche le premier element non barre dans tab
    tantque ((p*p<=n) et (tab[p]==0)) faire
      p:=p+1;
    ftantque;
  ftantque;
  //on remplit la liste prem avec les elements non nuls de tab
  prem:=[];
  pour j de 2 jusque n faire
    si (tab[j]!=0) alors
      prem:=append(prem,j);
fsi
11
fpour
  retourne(prem);
}:;
```

// Success
Warning, step is not numeric p
// Parsing crible
// Warning : x, declared as global variable(s) compiling crible

Done

12 `C100:=crible(100);`

$$[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]$$

13 `C1000:=crible(1000);`

$$[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]$$

14 `C10000:=crible(10000);`
Evaluation time : 6.22

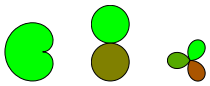
Done

15 `C100000:=crible(100000);`
Evaluation time : 676.56

Done
16 `l1:=length(C100000);C100000[l1-1];`

$$9592, 99991$$

17



11.2 [*] Test de primalité (suite de l'exercice n° 11.1)

Étant donné un entier $n \geqslant 2$, en utilisant la fonction `Erato.sci` de l'exercice 11.1 ou la commande `liste_premiers`, fabriquer un algorithme qui répond si n est ou non un nombre premier.

Mots-clefs : commande `liste_premiers`.

C'est assez évident. Voici un script solution basé sur la remarque : si n est le dernier terme de la liste des nombres premiers $\leqslant n$, alors n est premier et réciproquement.

```
n=input('n=');
A=liste_premiers(n);
if n==A($) then
    afficher(+string(n)+' est un nombre premier.');
else
    afficher(+string(n)+' n''est pas un nombre premier.');
end
```

Ce script étant chargé dans la console, en voici deux exécutions :

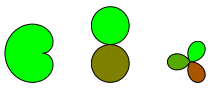
```
-->exec('Chemin du script', -1)
n=12345679
 12345679 n'est pas un nombre premier.
-->
```

puis

```
-->exec('Chemin du script', -1)
n=57885161
 57885161 est un nombre premier.
-->
```

Cela confirme que le nombre $p = 2^{57\,885\,161} - 1$ de l'exercice 11.1 est bien un nombre de Mersenne.

Tester si un nombre est premier ou non s'appelle tester sa primalité. Il y a d'autres tests de primalité, voir [18], le but étant de faire le moins de calculs possible.



11.3 [**] Nombre premier suivant (suite de l'exercice n° 11.1)

1 - Fabriquer un algorithme qui associe à tout entier $n \geqslant 2$ le plus petit nombre premier P_n qui lui est strictement supérieur.

2 - Quel est le premier nombre premier qui suit 10 000 000 ?

Mots-clefs : commande `liste_premiers`, boucle `tant que`, `find`, extraction d'un sous-vecteur, effacement de composantes dans un vecteur.

Solution n° 1 :

1 - On peut s'appuyer sur la commande `liste_premiers` ou, à la rigueur, sur la fonction `Erato.sci` de l'exercice 11.1. La commande `liste_premiers(k)` retourne la liste des nombres premiers compris entre 2 et `k` (au sens large). Il suffit donc de donner à `k` les valeurs `n`, `n+1`, *etc* jusqu'à ce que la longueur de la liste des nombres premiers augmente de 1. Alors, on sait que le dernier nombre de la liste est le nombre premier suivant.

```
n=input('n=');
tic();
stacksize('max');
A=liste_premiers(n);
r=taille(A);
p=A($);//  p : plus grand nombre premier <=n.
q=n+1;
B=liste_premiers(q)
s=taille(B);
while r==s
    q=q+1;
    B=liste_premiers(q);
    s=taille(B);
end;
temps=toc();
afficher('Le plus grand entier inferieur ou egal a '+string(n));
afficher('est '+string(p));
afficher('Son rang dans la liste des nombres premiers est '+string(r));
afficher('Le plus petit nombre premier >n est '+string(q));
afficher('Le calcul a dure '+string(temps));
afficher('secondes.');
```

Ce script exécute la commande `liste_premiers` (q-n+1) fois, nombre qui conditionne fortement le temps de calcul, voir la question n° 2.

2 - Exécutons ce script pour n=10 000 000 :

```
-->exec('Chemin du script', -1)
n= 10000000
 Le plus grand entier inf\'{e}rieur ou \'{e}gal \`{a} 10000000
 est 9999991
 Son rang dans la liste des nombres premiers est 664579
 Le plus petit nombre premier >n est 10000019
 Le calcul a dur\'{e} 25.781
 secondes.
```

```
-->
```

Le premier nombre premier après 10 000 000 est donc 10000019. C'est le 664 580ème nombre premier.

La commande `liste_premiers` a fonctionné 20 fois. Quand on calcule le premier nombre premier après 20 000 000, elle n'intervient que 4 fois, ce qui fait que le temps de calcul est plus court. On s'attend à ce que dans les cas usuels, la commande `liste_premiers` soit peu utilisée (car les nombres premiers sont nombreux). C'est un avantage de ce script.

```
-->exec('Chemin du script', -1)
 Le plus grand entier inferieur ou egal a 20000000
 est 19999999
 Son rang dans la liste des nombres premiers est 1270607
 Le plus petit nombre premier >n est 20000003
 Le calcul a dure 10.778
 secondes.
-->
```

On aurait pu répondre à la question n° 1 à l'aide de la fonction `nps.sci` suivante qui, pour l'entrée `n`, retourne le plus petit nombre premier strictement plus grand que `n`.

```
function q=nps(n)
    A=liste_premiers(n);
    r=taille(A);
    q=n+1;
    B=liste_premiers(q)
    s=taille(B);
    while r==s
        q=q+1;
        B=liste_premiers(q);
        s=taille(B);
    end;
endfunction
```

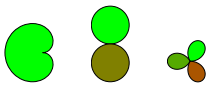
Utilisons `nps.sci` pour calculer le premier nombre entier après 50 000 000 :

```
-->exec('chemin de nps.sci', -1)
-->nps(50000000)
 ans  =
    50000017.
-->
```

Le calcul est long.

Solution n° 2 : Le problème est de savoir où chercher le premier nombre premier après n. D'après un théorème de P. Tchebyschev (1850, [18]), pour tout entier $n \geqslant 2$, il existe au moins un nombre premier p tel que $n < p < 2n$. Il suffit donc d'exécuter la commande `liste_premiers(2*n-1)` et d'extraire le nombre recherché dans la liste obtenue, ce qui est facile. Malheureusement, si par exemple $n = 50000000$, il faudra calculer la liste des nombres premiers $\leqslant 99999999$, trop grand nombre qui provoque un message d'erreur pour dépassement de capacité. La deuxième solution paraît donc meilleure mais elle ne l'est pas. Voici le script correspondant :

```
clear;
stacksize('max');
n=input('n=');//n entier superieur ou egal a 2.
L=liste_premiers(2*n-1);
A=find(L>n);
L=L(A);
p=min(L);
afficher('Le plus petit nombre premier >n est '+string(p));
```



11.4 [**] Factorisation en nombres premiers (suite de l'exercice n° 11.1)

Fabriquer un algorithme qui à tout entier $n \geqslant 2$ associe la suite de ses facteurs premiers et la suite de leurs puissances respectives.

Mots-clefs : commandes `liste_premiers`, `find`, extraction d'un sous-vecteur, effacement de composantes dans un vecteur.

L'algorithme ci-dessous est très simple : on passe en revue la liste des nombres premiers $\leqslant n$ obtenue à l'aide de la fonction `Erato.sci` (à charger dans la console) ou à l'aide de la commande `liste_premiers`, puis on regarde jusqu'à quelle puissance chacun d'eux divise `n` (si un nombre premier ne divise pas `n`, il le divise avec la puissance 0). On efface ces facteurs parasites avant d'obtenir le résultat définitif, qui est présenté sur la forme d'un tableau à 2 lignes, la première étant la ligne des facteurs, la seconde la ligne de leurs puissances.

```
// Charger la fonction Erato.sci.
function F=factorisation(n)
Facteurs=Erato(n);
Puissances=[];
for i=1:taille(Facteurs)
    j=1;
    while reste(n,Facteurs(i)^j)==0
        j=j+1
    end
    Puissances=[Puissances,j-1];
end
K=find(Puissances==0);
Facteurs(K)=[];
Puissances(K)=[];
F=[Facteurs;Puissances];
endfunction
```

Exemple : mettre 12345 sous la forme d'un produit de nombres premiers :

On charge dans la console `Erato.sci` puis `factorisation.sci` (téléchargeable sur le site) et on saisit dans la console la commande ad hoc :

```
-->clear
-->exec('Chemin de Erato.sci', -1)
-->exec('Chemin de factorisation.sci', -1)
-->factorisation(12345)
 ans  =
    3.    5.    823.
    1.    1.    1.
-->
```

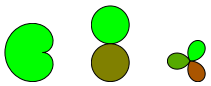
La décomposition de 12345 en un produit (unique) de facteurs premiers est donc : $12345 = 3 \cdot 5 \cdot 823$, ce que l'on peut vérifier à l'aide du script suivant (cette vérification est inutile mais amusante) :

```
m=prod(F(1,:).^F(2,:));
if n==m then
    afficher('Ce resultat est exact')
else
    afficher('Ce resultat est faux')
```

```
end
```

Exécutons le script :

```
-->exec('Chemin du script', -1)
Ce resultat est exact
-->
```



11.5 [***] Peut-on faire l'appoint ?

Quelqu'un doit payer un achat. Le commerçant n'ayant pas de monnaie lui demande de faire l'appoint, c'est à dire de donner exactement la somme due. Est-ce que c'est possible ? Si oui, quelles pièces doit-il donner ?

Mots-clefs : pour la solution n° 2, commandes `find`, `diag`, produit de 2 matrices, extraction de sous-matrices.

Solution n° 1 : Les données sont décrites dans les commentaires de la fonction `aptt.sci` (téléchargeable sur le site) ci-dessous, solution de ce problème. C'est un algorithme trivial dans sa conception : on calcule toutes les sommes que l'on peut payer avec les pièces disponibles. Quand on rencontre la somme requise, on note les pièces utilisées. `aptt.sci` retourne une matrice éventuellement vide dont chaque ligne décrit une solution.

```
function solutions=aptt(p,a,b,c,d,e,f,g,h)
// a,b,c,d,e,f,g,h designent respectivement les
//nombres de pieces de 1 c, 2 c, 5 c, 10 c, 20 c, de 50 c, de 1 euro et
//de 2 euros disponibles pour l'achat (nombres entiers >=0) ; p designe
//le prix de l'achat (en centimes).
solutions=[];
for ja=0:a
  for jb=0:b
    for jc=0:c
      for jd=0:d
        for je=0:e
          for jf=0:f
            for jg=0:g
              for jh=0:h
                s=200*jh+100*jg+50*jf+20*je+10*jd+5*jc+2*jb+ja;
                if s==p then
                  solutions=[solutions;[ja,jb,jc,jd,je,jf,jg,jh]];
                end;
              end;
            end;
          end;
        end;
      end;
    end;
  end;
end;
endfunction
```

Voici une exécution de ce script :

```
-->exec('Chemin du fichier aptt.sci', -1)
-->aptt(12,0,6,2,0,2,3,1,0)
 ans  =
          column 1 to 7
    0.    1.    2.    0.    0.    0.    0.
    0.    6.    0.    0.    0.    0.    0.
          column 8
```

```
    0.
    0.
-->
```

Cela signifie qu'il y a 2 façons de payer un achat de 12 centimes. La première consiste à donner une pièce de 2 centimes et 2 pièces de 5 centimes. Au contraire, avec les mêmes pièces, on ne peut pas payer un achat de 13 centimes :

```
-->exec('Chemin du fichier aptt.sci', -1)
-->solutions=aptt(13,0,3,2,0,2,3,1,0)
 solutions  =
     []
 -->
```

Pour payer 7,49 euros avec la répartition $[a,b,c,d,e,f,g,h] = [3,4,5,3,7,2,8,2]$, il y a trop de solutions. Dans le script ci-dessous, nous avons supprimé l'affichage des solutions (; à la fin de la ligne de commande) et demandé combien il y a de solutions. On en trouve 372 :

```
 -->solutions=aptt(749,3,4,5,3,7,2,8,2);
-->size(solutions,"r")
 ans  =
    372.
-->
```

Solution n° 2 : La fonction `aptm.sci` (téléchargeable sur le site) calcule aussi tous les paiements exacts possibles. L'algorithme repose sur du calcul matriciel.

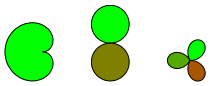
```
function Sol=aptm(p,a,b,c,d,e,f,g,h)
    E=[a,b,c,d,e,f,g,h];
    // a,b,c,d,e,f,g,h designent respectivement les nombres de pieces de
    //1 c, 2 c, 5 c, 10 c, 20 c, de 50 c, de 1 euro et de 2 euros
    //disponibles pour l'achat (nombres entiers >=0) ; p designe le prix
    //en c.
A=[0:E(1)]';
for i=2:8
    B=[];
    r=size(A,'r');
    for j=0:E(i)
        C=[A,j*ones(r,1)];
        B=[B;C];
    end
    A=B;
end
val=diag([1,2,5,10,20,50,100,200]);
sommes=sum(A*val,"c");
lignessol=find(sommes==p*ones(sommes));
Sol=A(lignessol,:)
endfunction
```

Cet algorithme est plus difficile que le premier mais n'est pas très bon. La matrice `A` a

$$(a+1)(b+1)(c+1)(d+1)(e+1)(f+1)(g+1)(h+1)$$

lignes et 8 colonnes. Le nombre de lignes de `A` peut être grand. Par exemple, dans le cas du paiement de 7,49 euros avec la répartition $[a,b,c,d,e,f,g,h] = [3,4,5,3,7,2,8,2]$, il retourne un message de dépassement de

capacité, ce qui n'est pas étonnant car `A` a alors 311 040 lignes. Néanmoins, ça marche si on alloue davantage d'espace pour le calcul à l'aide de la commande `stacksize('max')`.



11.6 [***] Records

Soit S une suite de nombres entiers. On appelle record de S son premier élément ainsi que tout terme de cette suite qui est strictement plus grand que tous les termes qui le précèdent.
Le problème est de produire la liste des records de S à l'aide d'un algorithme.

Mots-clefs : Boucles `tant que` et `pour`, commande `find`, criblage.

La solution présentée ci-dessous est un criblage inspiré par le crible d'ératosthène (voir 11.1). On garde le premier élément qui est par définition un record et on supprime tous les termes de S qui lui sont inférieurs ou égaux parce qu'ils ne peuvent pas être des records. Dans l'algorithme, ce nouvel état de S s'appelle encore S. Si sa longueur est 1, on arrête : il y a un seul record. Si c'est au moins 2, son deuxième élément est un record. Si la longueur de S est exactement 2, on arrête : il y a 2 records. Sinon, on crible S de nouveau à partir du deuxième record, comme on l'a fait précédemment à partir du premier record. On réitère le procédé si c'est nécessaire. On s'arrête quand le dernier record obtenu est le dernier terme de S qui est alors la liste des records.

```
S=input('S=');// S est une liste de nombres entiers.
i=1;
while i<taille(S)
    J=find(S(i+1:taille(S))<=S(i));
    S(i+J)=[];
    i=i+1;
end
afficher('La liste des records de S est ');
afficher(S);
```

Voici une exécution de cet algorithme : trouver la liste des records d'une suite de 1000 nombres entiers engendrée à l'aide de la commande `tirage_entier(1000,-13,67)` qui retourne 1000 nombres entiers tirés au hasard (suivant la loi uniforme) entre -13 et 67 :

```
-->clear
-->exec('Chemin du fichier ', -1)
S=tirage_entier(100,-13,67)
 La liste des records de S est
    16.    38.    40.    47.    64.    66.
-->
```

On constate que cette liste de 1000 nombres entiers ne comporte que 6 records (dans ce cas, le nombre de records est au moins 1 et au plus 81 ; c'est un nombre aléatoire).

Transformons cet algorithme en une fonction *scilab* que nous appellerons `records.sci` dont l'argument est une liste d'entiers `L`, cette fonction retournant le nombre de records de `L` :

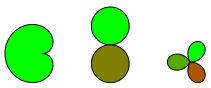
```
function R=nbrec(L)
    i=1;
    while i<taille(L)
        J=find(L(i+1:taille(L))<=L(i));
        L(i+J)=[];
        i=i+1;
    end
    R=taille(L);
```

```
endfunction
```

Application : calculons le nombre de records de 3 tirages au hasard de 100000 nombres entiers entre -13 et 67 :

```
-->clear
-->exec('Chemin de records.sci', -1)
-->L=tirage_entier(100000,-13,67);
-->R=nbrec(L)
 R  =
    3.
-->L=tirage_entier(100000,-13,67);
-->R=nbrec(L)
 R  =
    5.
-->L=tirage_entier(100000,-13,67);
-->R=nbrec(L)
 R  =
    6.
```

La fonction `records.sci` est téléchargeable sur le site.



11.7 [***] Un problème d'arithmétique : les citrons

Une paysanne indienne va au marché en bicyclette pour vendre ses citrons. Un passant distrait la bouscule. Le panier de citrons est renversé. Le fautif entreprend de ramasser les citrons.

Il demande à la paysanne : « combien y en avait-il ? »

Elle lui répond : « en les rangeant par deux, il en reste un ; en les rangeant par trois, il en reste un ; en les rangeant par quatre, il en reste un ; en les rangeant par cinq, il en reste un ; en les rangeant par six, il en reste un ; mais en les rangeant par sept, il n'en reste pas. »

« C'est bon » lui dit le passant, « je les ai tous ramassés ».

Sachant qu'il ne pouvait pas y avoir plus de 500 citrons dans le panier (ce qui fait environ 50 kg), combien y en avait-il exactement ?

Mots-clefs : commandes `string`, `ensemble`, `intersection`, `find`, criblage, critères de divisibilité.

On est sûr que ce problème a au moins une solution. Le résoudre de manière algorithmique est contre-nature car avec un peu d'arithmétique, il y a une solution rapide. Dans ce cas, l'algorithmique permet d'être à peu près parfaitement ignorant.

Solution algorithmique n° 1 : On écrit la liste des nombres entiers de 1 à 500. On la transforme en ensemble dont on prend l'intersection avec l'ensemble des nombres de 1 à 500 de la forme $2*p+1$, $3*p+1$, *etc.*

```
clear;
e1=intersection(ensemble(string(1:2:500)),ensemble(string(1:3:500)));
e2=intersection(ensemble(string(1:4:500)),ensemble(string(1:5:500)));
f1=intersection(e1,e2);
f2=intersection(f1,ensemble(string(1:6:500)));
g=intersection(f2,ensemble(string(7:7:500)));
afficher('L''ensemble des solutions possibles est ');
afficher(g);
```

Voici l'exécution de ce script :

```
-->exec('Chemin du script', -1)
L'ensemble des solutions possibles est
 {301}
-->
```

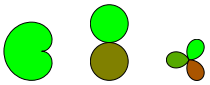
Solution algorithmique n° 2 : C'est la même que précédemment, en manipulant des vecteurs au lieu d'ensembles. On écrit la liste des nombres entiers de 1 à 500 puis on garde ceux qui, diminués de 1, sont divisibles par 2, *etc.* On passe donc ces nombres au crible. On ne manipule que des listes dans cette solution.

```
clear
V1=1:500;
V2=V1(find((V1-1)-2*floor((V1-1)/2)==0));
V3=V2(find((V2-1)-3*floor((V2-1)/3)==0));
V4=V3(find((V3-1)-4*floor((V3-1)/4)==0));
V5=V4(find((V4-1)-5*floor((V4-1)/5)==0));
V6=V5(find((V5-1)-6*floor((V5-1)/6)==0));
V=V6(find(V6-7*floor(V6/7)==0));
afficher('La liste des solutions possibles est ');
afficher(V);
```

Voici l'exécution de ce script :

```
-->exec('Chemin du script', -1)
La liste des solutions possibles est
    301.
-->
```

Solution arithmétique : Un peu d'arithmétique nous dit que le (ou les nombres) recherché est de la forme $60 * p + 1$ et est divisible par 7. C'est donc celui (ou ceux) des nombres 61, 121, 181, 241, 301, 361, 421, 481, 541 qui est divibible par 7, ce qui conduit à une solution unique : 301.



Bibliographie

[1] *Aide en ligne Scilab*
http://help.scilab.org/docs/5.4.1/fr_FR/

[2] Michaël Baudin. *Introduction à « Scilab »*, Jérôme Briot traducteur. Mis en ligne en mars 2013.
http://scilab.developpez.com/tutoriels/introduction-scilab

[3] Alexandre Casamayou-Boucau, Pascal Chauvin, Guillaume Connan *Programmation en Python pour les mathématiques*, 2ème édition, Dunod Éditeur (2016).
https://irem.univ-lille1.fr/activites/spip.php?article264

[4] *Flocons de Koch*, activité pour la classe de Première S
http://www.gradus-ad-mathematicam.fr/Premiere_Algorithmique1.htm

[5] Claude Gomez *Le calcul numérique : pourquoi et comment ?*
http://www.scilab.org/fr/community/education/maths/doc

[6] Renée De Graeve, Bernard Parisse, B. Ycart *Tutoriel « Xcas » : Démarrer en calcul formel*,
http://www-fourier.ujf-grenoble.fr/~parisse/giac/doc/fr/tutoriel/index.html

[7] Reneé De Graeve, et Bernard Parisse *Tutoriel « Xcas » : « Xcas » au lycée*,
http://www-fourier.ujf-grenoble.fr/~parisse/giac/troussesurvie_fr.pdf

[8] Reneé De Graeve, et Bernard Parisse *Correction avec Xcas du stage algorithmique de l'IREM de Grenoble*, 17 janvier 2010, 7. Le crible d'Ératosthène
http://www-fourier.ujf-grenoble.fr/~parisse/irem/Algocas.pdf

[9] Bernard Parisse et Renée De Graeve *Giac/Xcas*, version 1.1.2 (2014)
http://www-fourier.ujf-grenoble.fr/~parisse/giac_fr.html

[10] Thomas Rey *Pour commencer avec « Xcas »*,
http://reymarlioz.free.fr/spip.php?article150

[11] G. Sallet *Initiation à scilab, agrégation de mathématiques*, INRIA, Université de Metz, 2006-2007
http://poncelet.sciences.univ-metz.fr/~sallet/Scilab_intro_agreg.pdf

[12] *Scilab pour l'enseignement des mathématiques*, 2ème édition, 2013
http://www.scilab.org/fr/community/education/maths/doc

[13] Gérard Swinnen *Apprendre à programmer avec Python 3*, Eyrolles Éditeur, 3ème édition (2016) disponible en ligne :
https://fr.m.wikibooks.org/wiki/Apprendre_à_programmer_avec_Python/Licence

[14] Wikipedia. *American Standard for Information Interchange*
https://fr.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange

[15] Wikipedia *Crible d'Ératosthène*
http://fr.wikipedia.org/wiki/Crible_d'Eratosthene

[16] Wikipedia *Le tri à bulles*
https://fr.wikipedia.org/wiki/Tri_a_bulles

[17] Wikipedia *Matrices de Toeplitz*
http://fr.wikipedia.org/wiki/Matrice_de_Toeplitz

[18] Wikipedia *Nombre premier*
http://fr.wikipedia.org/wiki/Nombre_premier