

Les tableaux

A. Les vecteurs

◆ **Exercice 1.** [o]

Construire un vecteur contenant les vingt premiers carrés.

```
for i = 1:20 do v(i) = i^2; end, v,
```

◆ **Exercice 2.** [o] (Suite de Fibonacci)

La suite de Fibonacci est la suite $(u_n)_{n \geq 1}$ définie par la donnée de $u_1 = 1$, $u_2 = 2$ et la relation de récurrence $\forall n \geq 1$, $u_{n+2} = u_{n+1} + u_n$. Écrire une fonction **fibonacci**, de la variable **n**, qui calcule et affiche tous les termes de la suite de Fibonacci jusqu'à u_n .

```
function u = fibonacci(n)
    u(1) = 1; u(2) = 2;
    for i = 3:n do u(i) = u(i-1) + u(i-2), end
endfunction
```

◆ **Exercice 3.** [o]

En utilisant (sans la modifier) la fonction **premier** qui teste si un nombre entier est premier, écrire une fonction **premiers**, de la variable **n**, qui affiche dans un vecteur tous les nombres premiers jusqu'à **n**.

```
function v = premiers(n)
    v = [];
    for i = 1:n do
        if premier(i) then v = [v, i], end
    end
endfunction
```

◆ **Exercice 4.** [o]

Écrire une fonction **moyvect** qui calcule la moyenne des coordonnées d'un vecteur.

```
function m = moyvect(u)
    s = 0;
    for i = 1:length(u) do s = s + u(i); end
    m = s/length(u),
endfunction
```

◆ **Exercice 5.** [o]

Écrire une fonction **reversion** qui inverse l'ordre des coordonnées d'un vecteur.

```
function v = reversion(u)
    v = [];
    for i = 1:length(u) do v = [u(i), v], end
endfunction
```

◆ **Exercice 6.** [★]

1. Écrire une fonction `redondance` qui teste si un vecteur possède deux coordonnées égales.
2. Écrire un script `CinqNombresDistincts.sce` qui choisit simultanément 5 nombres entiers distincts entre 1 et 16.

```
1. function b = redondance(u)
    n = length(u);
    b = %f;
    for i = 1 : n do
        for j = i + 1 : n do
            if u(j) == u(i) then b = %t; end
        end
    end
endfunction

2. u = floor(16 * rand(1, 5)) + 1;
while redondance(u) do
    u = floor(16 * rand(1, 5)) + 1;
end
u,
```

◆ **Exercice 7.** [★]

Écrire une fonction `compartab` qui teste si deux vecteurs sont égaux.

```
function b=compartab(u,v)
    b=%t
    if length(u)<>length(v) then
        b=%f
    else
        for i=1:length(u) do
            if u(i)<>v(i) then b=%f, end
        end
    end
endfunction
```

◆ **Exercice 8.** [★] (Tri par bulle)

On veut trier par ordre croissant les coordonnées d'un vecteur donné.

L'une des méthodes les plus simples est le tri par bulle. Le principe consiste à parcourir le vecteur en comparant deux à deux les éléments voisins et en les permutant dans le cas où ils ne sont pas dans le bon ordre. On voit alors les plus grands éléments « remonter » vers la droite du vecteur, comme une bulle de champagne remonte à la surface d'un verre.

En effet, après un balayage complet, le plus grand élément est complètement à droite du vecteur. Il suffit alors d'itérer l'opération sur les autres coordonnées autant de fois qu'il le faut jusqu'à ce qu'un balayage supplémentaire du tableau ne provoque aucune permutation.

Par exemple, on a

3 5 2 1 8 9 7 3	vecteur de départ
3 2 1 5 8 7 3 9	5 et 9 remontent
2 1 3 5 7 3 8 9	3 et 8 remontent
1 2 3 5 3 7 8 9	2 et 7 remontent
1 2 3 3 5 7 8 9	5 remonte

Écrire une fonction `tribulle` qui trie un vecteur (entré en argument de cette fonction) selon cet algorithme.

```
function A = tribulle(u)
    A = u;
    marquelafin = "un balayage de plus";
    while marquelafin <> "plus de balayage" do
        marquelafin = "plus de balayage";
    end
```

```

for j = 1:length(u) - 1 do
    if u(j) > u(j + 1) then
        prov = u(j);
        u(j) = u(j + 1);
        u(j + 1) = prov;
        marquelafin = "un balayage de plus";
    end
end
end
if marquelafin<>"plus de balayage" then A = [A;u], end
end
endfunction

```

Il existe beaucoup d'autres algorithmes de tri (par insertion, par sélection croissante, ...). Les livres de Knuth constituent à ce sujet une lecture indispensable pour quiconque veut en apprendre bien davantage.

KNUTH ERVIN DONALD, américain, né le 10 janvier 1938

Informaticien de renommée mondiale et professeur émérite en informatique à l'Université de Stanford, Knuth est surtout connu pour l'ouvrage The Art of Computer Programming (couramment appelé TAOCP), dont il a entrepris la rédaction à la fin des années 60. Les tomes parus à ce jour sont l'une des références dans le domaine de l'informatique (pour ne pas dire qu'ils sont la « bible » des informaticiens). Les premiers tomes ont été composés et imprimés avec les techniques traditionnelles (caractères en plomb) avant l'arrivée des premières photocomposeuses. Pour garder (et même améliorer) l'aspect riche (et précieux) des premiers tomes, Knuth a entrepris, dans le courant des années 70, la conception d'un système complet d'édition de textes scientifiques lui garantissant une qualité constante dans le temps. Il avait d'emblée visé un système définitif (parfait ?) en concevant deux langages : TeX et Metafont (le premier pour le texte et le second pour les caractères) dont la syntaxe et le lexique de base n'évolueraient pas au cours du temps et qui seraient « dégelés » à l'aide de moyens évolutifs et adaptables aux besoins. Ainsi il assurait la qualité, la portabilité et la pérennité de ses textes (ce cours, ainsi que tous les documents que je vous distribue, sont rédigés à l'aide de TeX). Mais Knuth est surtout connu pour ses nombreuses contributions dans plusieurs branches de l'informatique théorique. Il a, en particulier, créé un domaine entier de l'informatique : l'analyse d'algorithmes, c'est-à-dire l'étude mathématique rigoureuse de l'efficacité (en temps ou en mémoire) des algorithmes (en moyenne ou dans le pire des cas). Knuth consacre désormais presque toute son énergie à achever les 7 volumes de TAOCP (la première édition du premier volume remonte à 1968 et seuls trois volumes ont paru).



◆ **Exercice 9.** [★] (Première loi de Mendel)

Ayant des individus homozygotes AA et aa dans la lignée parentale, la première génération fille F_1 n'aura que des individus hétérozygotes Aa de phénotype A . En les croisant entre eux, on attend la proportion « 3 pour 1 » entre le phénotype A (correspondant aux génotypes AA , aA et Aa) et le phénotype a (correspondant au génotype aa).

Cependant, si vous comptez les phénotypes de 100 individus de la F_2 , vous n'aurez que très rarement les bonnes proportions (c'est-à-dire 25 phénotypes a et 75 phénotypes A). On va utiliser SCILAB pour visualiser la distribution du nombre de phénotypes a dans ce type d'expérience.

1. Écrire un script `Mendel.sce` qui génère aléatoirement un génotype AA , aA , Aa ou aa .
2. Compléter le script précédent pour qu'il génère 100 génotypes (AA , aA , Aa ou aa) et compte le nombre de phénotypes a et le nombre de phénotypes A obtenus.
3. Terminer ce script en lui faisant compter 1000 fois le nombre de phénotypes a dans une population de 100 individus. On stockera les 1000 résultats dans un vecteur `vectphenoa` et on calculera la valeur moyenne des coordonnées de ce vecteur (on pourra utiliser `moyvect`).
4. Ajouter à la fin de votre script la commande `histplot(20, vectphenoa)` et compiler...

```

for j = 1 : 1000 do
    nombrephenoa = 0; nombrephenoA = 0;
    for i = 1 : 100 do
        gene1 = rand(); gene2 = rand();
        if gene1 < 0.5 & gene2 < 0.5 then
            geno = "aa";
        elseif gene1 >= 0.5 & gene2 < 0.5 then
            geno = "Aa";
        elseif gene1 < 0.5 & gene2 >= 0.5 then
            geno = "Aa";
        else
            geno = "AA";
        end
        if geno == "aa"
            nombrephenoa = nombrephenoa + 1;
        else
            nombrephenoA = nombrephenoA + 1;
        end
    end
end

```

```

elseif gene1 >= 0.5 & gene2 >= 0.5 then
    geno = "AA";
end
if geno == "aa" then
    nombrephenoa = nombrephenoa + 1;
else
    nombrephenoA = nombrephenoA + 1;
end
end
end
vectphenoa(j) = nombrephenoa;
end
moyvect(vectphenoa), histplot(20, vectphenoa)

```

B. Les matrices

◆ *Exercice 10.* [o]

1. Écrire une fonction `matnulle` qui permette de construire la matrice nulle de taille $n \times m$.
2. Écrire une fonction `matid` qui permette de construire la matrice identité I_n .

```

1. function 0 = matnulle(n,m)
    for i = 1:n do
        for j = 1:m do
            0(i,j) = 0,
        end
    end
end
endfunction

```

En fait, cette fonction existe déjà en SCILAB et s'appelle `zeros`.

```

2. function I = matid(n)
    for i = 1:n do
        for j = 1:n do
            I(i,j) = 0,
        end
        I(i,i) = 1,
    end
end
endfunction

```

En fait, cette fonction existe déjà en SCILAB et s'appelle `eye` (elle permet même de faire des matrices rectangulaires).

◆ *Exercice 11.* [o]

1. Écrire une fonction `matprod` qui effectue le produit de deux matrices lorsque c'est possible et vous signale l'erreur dans le cas contraire.
2. Écrire une fonction `mattrace` qui calcule la trace d'une matrice carrée, c'est-à-dire la somme des coefficients diagonaux de cette matrice.

```

1. function C = matprod(A,B)
    tA = size(A); tB = size(B);
    if tA(2) <> tB(1) then
        C = "Le calcul est impossible"
    else
        for i = 1:tA(1) do
            for j = 1:tB(2) do
                C(i,j) = 0;
                for k = 1:tA(2) do
                    C(i,j) = C(i,j) + A(i,k) * B(k,j);
                end
            end
        end
    end
end
endfunction

```

Bien que la commande $A * B$ permette d'accéder directement à ce calcul, le programme de BCPST demande de savoir programmer le produit matriciel.

```
2. function tr=mattrace(A)
    tA=size(A);
    if tA(1)<>tA(2) then
        tr="Le calcul est impossible"
    else
        tr=0;
        for i=1:tA(1) do
            tr=tr+A(i,i);
        end
    end
endfunction
```

Là encore, la commande existe déjà (elle s'appelle `trace`) mais on vous demande de savoir la programmer.

◆ **Exercice 12.** [o]

Écrire une fonction `danstab` qui teste si un élément `x` appartient à une matrice `A`.

```
function dedans = danstab(A, x)
    dedans = %f;
    taille = size(A)
    for i = 1 : taille(1) do
        for j = 1 : taille(2) do
            if A(i,j) == x then
                dedans = %t;
            end
        end
    end
endfunction
```

◆ **Exercice 13.** [o]

Écrire une fonction `permuteligne` qui permute deux lignes d'une matrice.

```
function B = permuteligne(A, k, l)
    B = A; B(k,:) = A(l,:); B(l,:) = A(k,:);
endfunction
```

◆ **Exercice 14.** [o]

Écrire une fonction `rempl` qui remplace dans une matrice `A` toutes les occurrences de `x` par `y`.

```
function B = rempl(A, x, y)
    tA = size(A);
    for i = 1 : tA(1) do
        for j = 1 : tA(2) do
            if A(i,j) == x then
                B(i,j) = y;
            else
                B(i,j) = A(i,j);
            end
        end
    end
endfunction
```

◆ Exercice 15. [★]

Écrire une fonction `pascal` qui récite les premières lignes du triangle de Pascal.

```
function T = pascal(n)
    T(1,1) = 1;
    for i = 2:n+1 do
        T(i,1) = 1;
        T(i-1,i) = 0;
        for j = 2:i do
            T(i,j) = T(i-1,j) + T(i-1,j-1);
        end
    end
    T = string(T); T = rempl(T,"0"," "), // Pour faire joli...
endfunction
```

◆ Exercice 16. [★★] (Bataille navale simple)

On veut écrire un script `Batnav.sce` qui simule le jeu de la bataille navale (dans un cas simple où les bateaux sont tous de longueur 1). Ce script doit :

- ▶ attribuer 11 points au joueur (qu'il devra éviter de perdre ensuite);
- ▶ fabriquer une grille 4×4 et choisir au hasard 5 cases de cette grille pour l'emplacement des bateaux (on fabriquera une matrice 4×4 dont tous les coefficients sont nuls sauf ceux qui correspondent aux bateaux qui vaudront 1; on pourra utiliser les exercices précédents, en particulier le script `CindNombresDistincts.sce` de l'exercice ??);
- ▶ afficher au début une grille vierge (on affichera une matrice remplie du caractère ' ');
- ▶ procéder au tir en
 - ▷ demandant au joueur les coordonnées de son tir;
 - ▷ vérifiant si la case correspondante contient un bateau;
 - ▷ affichant la nouvelle grille avec un o dans la case choisie si celle-ci contient un bateau et un x dans le cas contraire;
 - ▷ retirant un point au joueur si son tir a fait plouf;
- ▶ recommencer l'opération précédente tant que tous les bateaux n'ont pas été découverts ou que le score n'est pas tombé à zéro;
- ▶ afficher finalement le score du joueur.

```
// Génération aléatoire de la position des bateaux
bateau = floor(16 * rand(1,5)) + 1;
while redondance(bateau) do bateau = floor(16 * rand(1,5)) + 1; end
// Remplissage de la grille (cette étape n'est en fait pas indispensable...)
for i = 1:4 do
    for j = 1:4 do
        if danstab(bateau, 4 * (i - 1) + j) then
            P(i,j) = 1;
        else
            P(i,j) = 0;
        end
    end
end
// Jeu
disp("Le jeu commence.")
disp("Vous devez détruire 5 bateaux dans une grille 4x4.")
disp("Vous avez 11 points. Vous en perdez un à chaque mauvais tir.")
grille = [" ", " ", " ", " "; " ", " ", " ", " "; " ", " ", " ", " "; " ", " ", " ", " "];
score = 11;
pascoule = 5;
while score > 0 & pascoule > 0 do
    rep = input(' Quelles sont les coordonnées de votre tir? (sous forme [i, j])');
    i = rep(1); j = rep(2);
    if P(i,j) == 1 then
        disp("Coulé!"),
        grille(i,j) = "o"; disp(grille),
    end
end
```

```
    pascoule = pascoule - 1;
    disp("Vous avez toujours "+string(score)+" points")
else
    disp("Plouf!"),
    grille(i,j) = "x"; disp(grille)
    score = score - 1;
    disp("Vous perdez 1 point et votre score est désormais "+string(score))
end
else
    if score == 0 then
        disp("Vous avez perdu!")
    else
        disp("Bravo, vous avez gagné et votre score est "+string(score))
    end
end
end
```