TP 1

# Révisions. Matrices et programmation.



## Compétences attendues.

✓ Création d'une matrice, opérations sur les matrices.

✓ Programmer en `Scilab` : fonctions d'entrée et de sortie, boucles `if`, `for`, `while`.


Mathieu Mansuy - Professeur de Mathématiques en ECS2 au Lycée Louis Pergaud (Besançon)
`mansuy.mathieu@hotmail.fr`

# En préambule

Le logiciel de programmation `Scilab` est à télécharger ici. Je vous demande de l'installer sur vos ordinateurs personnels.

Pour pouvoir sauvegarder vos TP durant l'année, créer un dossier `Scilab` sur votre lecteur personnel `perso(\\scribe\prenom.nom)(U:)`, puis un sous-dossier TP1 (on ajoutera un nouveau sous-dossier TP à chaque nouveau TP). Dans le menu Fichier de la console, cliquer sur « Changer le répertoire courant... » et choisir le répertoire `U:\Scilab\TP1\`.

En cas de besoin, vous disposez de l'aide `Scilab` qui indique la syntaxe précise de chaque instruction, accompagnée de nombreux exemples. Il suffit pour cela de taper dans la console :

```
--> help
```

Si on désire de l'aide sur une fonction particulière de `Scilab`, on fait suivre `help` du nom de cette fonction. On obtient par exemple l'aide de la fonction `plot2d` en tapant :

```
--> help plot2d
```

# 1 L'environnement de travail Scilab

## 1.1 Calcul numérique simple

`Scilab` peut s'utiliser comme une calculatrice : on saisit une instruction directement dans la console (après `-->`) puis on l'exécute en entrant.

**Définition.**

- **Opérations :** `+`, `-`, `*`, `/` et `^`.
- **Fonctions usuelles :** `log` (logarithme népérien ln), `exp`, `cos`, `sin`, `tan`, `sqrt` (racine carrée), `floor` (partie entière) et `abs`.
- **Constantes :** `%e`, `%pi` et `%i`.

**Exemple.** Taper les commandes suivantes :

```
--> (2+3)*4-5
--> exp(log(3))
--> 2+3*4-5 ; 2+3*(4-5), (-1)^(-2)/(-1)^(-3)
--> %pi, %e, %i
--> ans
--> sin(%pi)
--> ans
```

**Remarques.**

- Le résultat final est contenu dans la variable `ans`.
- On notera qu'une instruction suivie d'un point-virgule `;` est exécutée mais n'est pas affichée.

## 1.2 Les booléens

Une expression booléenne peut prendre soit la valeur « vrai » (%T) ou « faux » (%F).

**Définition.**

- **Comparaisons - tests :** `==` (test d'égalité), `<`, `>`, `<=`, `>=`, `<>` (différent de).
- **Connecteurs logiques :** & (et), `and` (et), | (ou), `or` (ou).

**Exemples.**

```
--> 2==3, 1.4 < sqrt(2), %pi^2<10, 1+sqrt(3)^2<>4
-->(1<2)&(2^2==4)
-->(sin(%pi)>=1)|(2^2==4)
```

### 1.3 Les variables

Pour affecter une valeur à une variable, on utilise le signe `=`.

**Exemple.** Taper les instructions suivantes.

```
--> t=%pi/4
--> sin(t)^2
```

On peut libérer la place mémoire occupée par une variable en la détruisant à l'aide de la fonction `clear`.

**Exemple.** Taper les instructions suivantes.

```
--> t
--> clear t
--> t
```

Pour effacer tous les calculs de la console ainsi que toutes les variables existantes, on peut utiliser `clear`.

### 1.4 Les chaînes de caractères

Pour définir une chaîne de caractères, on utilisera des guillemets anglo-saxons `"`.

**Exemple.** `--> "Mon  message"`.

## 2 Les matrices

### 2.1 Création de matrices

Pour rentrer une matrice cœfficient par cœfficient, on encadre par des crochets, on sépare les éléments d'une même ligne par des virgules et on change de ligne grâce aux points-virgules.

**Exemple.** `A=[1,2,-1,2;3,7,8,4;6,-4,2,5]` donne la matrice $A = \begin{pmatrix} 1 & 2 & -1 & 2 \\ 3 & 7 & 8 & 4 \\ 6 & -4 & 2 & 5 \end{pmatrix}$.

Pour obtenir le format ou la taille d'une matrice, on utilise la fonction `size`.

**Exemple.** Taper les instructions suivantes.

```
--> size(A)
--> size(A,1)
--> size(A,2)
```

**Définition.**

- `zeros(n,p)` : matrice nulle à $n$ lignes et $p$ colonnes.
- `ones(n,p)` : matrice ne contenant que des 1 à $n$ lignes et $p$ colonnes.
- `eye(n,p)` : matrice identité à $n$ lignes et $p$ colonnes (complétée par des zéros si $n \neq p$).

**Exemple.** Taper les instructions suivantes.

```
--> zeros(3,5)
--> ones(3,2)
--> eye(3,3), eye(3,5)
```

## 2.2 Création de vecteurs

**Définition.**

- `[a:r:b]` donne le vecteur-ligne $\begin{pmatrix} a & a+r & a+2r & a+3r & \dots & a+nr \end{pmatrix}$, le dernier cœfficient étant le plus grand réel $a+nr$ inférieur ou égal à $b$ si $r > 0$ (le plus petit réel $a+nr$ supérieur ou égal à $b$ si $r < 0$).
  `[a:b]` donne le même vecteur que `[a:1:b]`.
- `linspace(a,b,n)` donne le vecteur-ligne de premier cœfficient $a$ et de dernier cœfficient $b$, contenant $n$ valeurs régulièrement espacées entre $a$ et $b$.
  `linspace(a,b)` donne le même vecteur que `linspace(a,b,100)`.

**Exemple.** Taper les instructions suivantes.

```
--> I=3:9, J=1:3:13, K=13:-3:1
--> t=linspace(0,1,5)
```

## 2.3 Extraction et insertion

**Définition.**

- `A(i,j)` donne le cœfficient de la ligne $i$ et la colonne $j$ de $A$.
- `A(k)` donne le $k$-ème cœfficient de $A$ si on numérote ses cœfficients de haut en bas puis de gauche à droite.

**Exemple.** Taper les instructions suivantes.

```
--> A
--> A(3,2), A(7), A(10)
--> A(3,2) = 5; A
```

**Remarque.** `Scilab` parcourt donc la matrice de haut en bas, puis de gauche à droite.

**Définition.**

- `A(i,:)` donne la ligne $i$ de $A$.
- `A(:,j)` donne la colonne $j$ de $A$.
- `A(n1:n2,p1:p2)` est la sous-matrice de $A$ constituée des lignes $n1$ à $n_2$ et des colonnes $p1$ à $p2$.

**Exemple.** Taper les instructions suivantes.

```
--> A(:,2)
--> A(3,:)
--> A(3,:)= [0,0,0,0]; A
--> A(1:2,1:3)
```

## 2.4 Opérations sur les matrices

**Définition.**

- `A=[B,C]` donne une matrice qui contient les colonnes de $B$ puis celles de $C$, à condition que $B$ et $C$ aient le même nombre de lignes.
- `A=[B;C]` donne une matrice qui contient les lignes de $B$ puis celles de $C$, à condition que $B$ et $C$ aient le même nombre de colonnes.

**Exemple.** Taper les instructions suivantes.

```
--> u=[3,2,1];v=[4,5,6];w=[9,8,7,6];
--> [v,u,w], [v;u]
--> [w;v;u]
```

**Définition.**

- **Transposée d'une matrice :** `A'` donne la transposée de la matrice $A$.
- **Inverse d'une matrice :** `inv(A)` donne la matrice inverse $A^{-1}$.
- **Somme de deux matrices :** `A+B` donne la somme $A + B$.
- **Produit de deux matrices :** `A*B` donne le produit $AB$.
- **Puissance d'une matrice :** `A^k` donne la matrice $A^k$.

**Exemples.** Taper les instructions suivantes.

```
--> a=[1,2;3,4]; b=[5,6;7,8];
--> a+b, a*b, inv(a), b', a^4
```

On peut aussi faire ces opérations coefficients par coefficients. Il faudra alors ajouter un point · qui précèdera l'opération que l'on souhaite faire.

**Définition.**

- **Produit coefficient par coefficient :** `A.*B` donne le produit terme à terme de $A$ par $B$.
- **Puissance coefficient par coefficient :** `A.^k` donne une matrice dont les cœfficients sont les cœfficients de $A$ à la puissance $k$.
- **Quotient coefficient par coefficient :** `A./B` donne le quotient terme à terme de $A$ par $B$.
- Toutes les fonctions usuelles peuvent s'appliquer terme à terme à une matrice.

**Exemple.** Taper les instructions suivantes.

```
--> a.*b, a./b, a.^2
--> log(a)
```

**Définition.**

- `sum(A)` calcule la somme de tous les éléments de $A$.
- `prod(A)` calcule le produit de tous les éléments de $A$.
- `mean(A)` calcule la moyenne de tous les éléments de $A$.
- `cumsum(A)` donne la matrice des sommes cumulatives de $A$ de haut en bas puis de gauche à droite.
- `max(A)` donne le maximum des cœfficients de $A$.
- `min(A)` donne le minimum des cœfficients de $A$.
- `size(A)` donne un vecteur $[n, p]$ où $n$ est le nombre de lignes et $p$ le nombre de colonnes de la matrice $A$.
- `rank(A)` donne le rang de la matrice $A$.
- `spec(A)` donne le spectre de $A$ (c'est-à-dire ses valeurs propres).

**Exemples.** Rentrer la matrice $B = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$, et taper les instructions suivantes.

```
--> sum(B), mean(B), max(B), rank(B)
--> cumsum(B), cumprod(B)
```

Que fait la fonction `cumprod` ?

**Exemple.** Que font les instructions suivantes ?

```
--> prod(1:10)
--> cumprod(1:10)
```

### 2.5 Recherche d'un élément dans une matrice

**Définition.**

Si $B$ est une matrice de booléens alors :

- `find(B)` donne les indices des éléments vrais (de haut en bas puis de gauche à droite).
- `[n,p]=find(B)` repère les mêmes indices mais renseigne dans $n$ la ligne et dans $p$ la colonne.

**Exemples.** Entrer la matrice $C = \begin{pmatrix} 1 & 2 & -1 \\ 3 & 7 & 8 \\ 6 & -4 & 2 \end{pmatrix}$ et taper les instructions suivantes.

```
--> find(C==2)
--> [n,p]=find(C==2)
```

## 3 Programmer en Scilab

### 3.1 Les scripts

Lorsque le nombre d'instructions est plus important, on utilise l'éditeur de texte `SciNotes` :

- on l'ouvre depuis la console grâce au menu `Applications/SciNotes`, ou directement par l'instruction `--> scinotes` ;
- On tape les instructions dans ce fichier (appelé le *script*) ;
- On le sauvegarde dans le répertoire courant sous la forme d'un fichier `ex0.sce` où 0 est le numéro de l'exercice (voire `ex0.0.sce` si l'on veut préciser en plus le numéro de la question) ;
- On l'exécute ensuite dans la console par la commande `exec`, ou directement à partir de `SciNotes` en cliquant sur ▷ ou en appuyant sur `F5`.

**Exemple.** Ouvrir l'éditeur de texte et taper les lignes suivantes.

```
1 // Representation graphique d'une fonction sur l'intervalle [-1,1]
2 n=100;
3 x=linspace(-1,1,n);
4 y=exp(-x^2/2)/sqrt(2*%pi);
5 clf();
6 plot2d(x,y)
```

Enregistrer votre fichier sous le nom `essai.sce`. On peut alors exécuter ce script dans la console avec l'instruction

```
--> exec("essai.sce")
```

**Remarques.**

- Pour éviter l'affichage des différentes instructions, on termine la commande `exec` par un point virgule (par exemple `--> exec("essai.sce");`).
- On peut écrire des commentaires en les précédant de //. Tout ce qui suit // ne sera pas exécuté.

**Conseils de mise en forme des scripts.**

- Écrire une seule instruction par ligne.
- Commenter son script (rôle de chaque variable, d'une boucle, ...) pour s'y retrouver plus rapidement lorsqu'on reprendra le programme. Les lignes de commentaires commencent par le symbole //.
- Sauter des lignes entre les différentes étapes du script et les commenter.
- Indenter (i.e. mettre un espace au début de la ligne) les boucles, structures conditionnelles, ... surtout lorsqu'elles s'imbriquent les unes dans les autres, afin de rendre visible le début et la fin d'une boucle, et ainsi de bien faire ressortir la structure du programme.

**Pratique.** En sélectionnant une partie d'un programme et en faisant `ctrl D`, toute la zone sélectionnée se retrouve précédée d'un `//`. Elle ne sera donc pas exécutée. C'est très pratique pour repérer une erreur dans un programme : on exécute le programme partie par partie. On constate ainsi ce qui fonctionne ou pas, et on détermine de cette manière où est l'erreur. Pour activer de nouveau une zone mise en commentaire, on la sélectionne et on effectue ctrl maj D.

## 3.2 Les fonctions d'entrée et de sortie

**Définition.**

- **Instructions de saisie :**
  L'instruction `x=input(" ")` donne la main à l'utilisateur pour qu'il rentre la valeur à affecter à la variable x.
  L'instruction `x=input("Mon message")` écrit le message `Mon message` sur l'écran puis donne la main à l'utilisateur pour qu'il rentre la valeur à affecter à la variable x.
- **Instructions d'affichage :**
  L'instruction `disp(calcul)` affiche le résultat du calcul.
  L'instruction `disp(calcul1,calcul2,"Mon message")` écrit le message `Mon message` sur l'écran puis le résultat de calcul2 puis le résultat de calcul1 (attention, affichage en sens inverse).

**Exemple.** Taper les instructions suivantes.

```
--> a=input(" Entrez un nombre positif")
--> disp(a, (1+sqrt(a))/2," la solution est ")
```

## 3.3 Instructions conditionnelles

**Définition.**

- **Sans alternative :**
  ```
  if condition then instruction1; instruction2; ...  ; end
  ```
- **Avec alternative :**
  ```
  if condition then instruction1; instruction2; ...
     else instruction1bis; instruction2bis; ...
  end
  ```

**Exemple.** Taper dans un script les instructions suivantes, puis l'exécuter. Que calcule-t-il ?

```
a=input(" Entrez un reel a : ")
if a>=0 then
      disp(a)
      else disp(-a)
end
```

## 3.4 Boucle for

**Définition.**

- **Syntaxe :**
  `for variable=rangdébut:pas:rangfin ; instruction1; instruction 2; ... ; end`

- **Syntaxe dans un script :**

```
for variable=rangdébut:pas:rangfin
  instruction1
  instruction2
  ...
end
```

**Exemple.** Taper dans un script les instructions suivantes, puis l'exécuter. Que calcule-t-il ?

```
r=1;
for i=1:100
      r=r*i;
end
disp(r);
```

## 3.5 Boucle while

**Définition.**

- **Syntaxe :**
  `while condition then instruction1; instruction 2; ... ; end`

- **Syntaxe dans un script :**

```
while condition
  instruction1
  instruction2
  ...
end
```

**Exemple.** Taper dans un script les instructions suivantes, puis l'exécuter. Que calcule-t-il ?

```
s=0; i=2;
while i<=100 then
      s=s+i;
      i=i+2;
end;
disp(s);
```

## 4 Exercices

**Exercice 4.1 (★)**
Sans rentrer les cœfficients un à un, déclarer les matrices $A = \begin{pmatrix} 5 & 3 & 3 \\ 3 & 5 & 3 \\ 3 & 3 & 5 \end{pmatrix}$ et $B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$.

```
--> A = 3*ones(3,3)+2*eye(3,3)
--> B = eye(4,4) ; B(:,1) = ones(4,1)
```

**Exercice 4.2 (★)**
Que vaut le produit d'un vecteur colonne $\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$ par le vecteur ligne $\begin{pmatrix} 1 & \dots & 1 \end{pmatrix}$ ?

En déduire une ligne de commande créant la matrice $\begin{pmatrix} 1 & 1 & \dots & 1 \\ 2 & 2 & \dots & 2 \\ \vdots & \vdots & & \vdots \\ 10 & 10 & \dots & 10 \end{pmatrix}$.

On a :

$$\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} \times \begin{pmatrix} 1 & \dots & 1 \end{pmatrix} = \begin{pmatrix} a_1 & a_1 & \dots & a_1 \\ a_2 & a_2 & \dots & a_2 \\ \vdots & \vdots & & \vdots \\ a_{10} & a_{10} & \dots & a_{10} \end{pmatrix}.$$

On en déduit les commandes suivantes (rappelons que `v'` désigne la transposée du vecteur `v`) :

```
--> A = (1:10)'*ones(1,10)
```

**Exercice 4.3 (★)**
On définit la matrice $A = \begin{pmatrix} 1 & 1 & 2 & 0 \\ 1 & -1 & 0 & -2 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & -1 \end{pmatrix}$. Déclarer la matrice $A$. Appliquer l'algorithme du pivot de Gauss à l'aide de `Scilab` (on effectuera chacune des opérations élémentaires sur `Scilab`). En déduire le rang. Vérifier votre résultat à l'aide de la fonction `rank`.

On applique l'algorithme de Gauss étape par étape, puis on vérifie à la fin que le rang de $A$ est bien égal à 2.

```
--> A = [1,1,2,0;1,-1,0,-2;0,1,1,1;1,0,1,-1]
--> B = A
--> B(2,:) = B(2,:)-B(1,:) ; B(4,:) = B(4,:)-B(1,:); B
--> B(2,:) = (-1/2)*B(2,:); B
--> B(3,:) = B(3,:)-B(2,:); B(4,:) = B(4,:)+B(2,:); B
--> rank(A)
```

---

**Exercice 4.4 (★★)**

1. (a) En une seule ligne de commande, créer le vecteur $x = \left(1, \frac{1}{4}, \frac{1}{9}, \frac{1}{16}, \ldots, \frac{1}{100}\right)$ sans saisir un à un les éléments.

   (b) Compléter la commande précédente pour qu'elle renvoie $\sum_{k=1}^{10} \frac{1}{k^2}$.

2. En une seule ligne de commande, calculer la somme $\sum_{n=0}^{10} \frac{1}{2^n}$.

3. Que calculent les commandes suivantes :

   (a) `x = ones(1,10) ; y = cumsum(x)`

   (b) `x = ones(1,10) ; y = sum(cumsum(x))`

   (c) `x = ones(1,10) ; y = sum(cumsum(cumsum(x)))`

1. (a) On fait des opérations pointées (coefficients par coefficients).

   ```
   --> x = (1:10).^(-2)
   ```

   (b) On applique la fonction `sum` :

   ```
   --> x = (1:10)^(-2); sum(x)
   ```

2. Sur le modèle de la question précédente :

   ```
   --> sum((1/2)^(0:10))
   ```

3. (a) `x` est le vecteur de taille 10 contenant que des 1. `y` est un vecteur de taille 10 dont la $i$-ème composante contient :

   $$y_i = \sum_{j=1}^{i} x_i = \sum_{j=1}^{i} 1 = i.$$

   (b) `y` contient le réel :

   $$\sum_{i=1}^{10} i = \frac{10 \times 11}{2} = 55.$$

(c) `y` contient cette fois le réel :

$$\sum_{i=1}^{10}\sum_{j=1}^{i} j = \sum_{i=1}^{10} \frac{i(i+1)}{2} = \frac{1}{2}\sum_{i=1}^{10}(i+i^2) = \frac{10\times 11}{4} + \frac{10\times 11\times(2\times 10+1)}{12}.$$

---

**Exercice 4.5 (★★)**

1. Écrire une commande affichant tous les multiples de 7 inférieur ou égaux à 1000.
2. Compléter la commande précédente pour qu'elle renvoie le plus grand multiple de 7 inférieur ou égal à 1000.
3. Ecrire une commande renvoyant le plus petit multiple de 7 supérieur strictement à 1000.

1. `-> 1:7:1000`
2. On utilise la fonction `max` : `-> max(1:7:1000)`. La réponse est 994.
3. On peut proposer : `-> max(1:7:1000)+7`. La réponse est 1001.

---

**Exercice 4.6 (★★)**

Le nombre de véhicules passant sur une petite route de campagne en une journée est une variable aléatoire $X$ qui suit une loi de Poisson $\mathscr{P}(5)$.

On peut simuler une année de circulation sur cette route en tapant `A = grand(52,7,'poi',5)`. La $i$-ème ligne de $A$ correspondant à la $i$-ème semaine de circulation. On obtient ainsi une matrice A qui contient $52 \times 7$ nombres tirés suivant la loi $\mathscr{P}(5)$.

1. Déterminer le nombre maximal de véhicules circulant sur la route en une journée durant l'année. Et le nombre maximal de véhicules un mercredi durant l'année ?
2. Qu'effectue la commande `sum(A,'c')` ? À l'aide de cette commande, déterminer les numéros des semaines où la route à connu son maximum de fréquentation.
3. Déterminer le nombre de jours où la route a connu son minimum de fréquentation.
4. En une seule ligne de commande, calculer le nombre moyen de véhicules par jour durant ces 364 jours. Recommencez plusieurs fois en recréant la matrice A. Que constatez-vous ? Était-ce prévisible ?
5. En une seule ligne de commande, évaluer la probabilité qu'une journée voit passer plus de 8 véhicules. Sauriez-vous calculer la valeur exacte de cette probabilité ?

1. On obtient le nombre maximal de véhicules circulant en une journée grâce à la commande `-> max(A)`.

   Pour obtenir le mercredi où il y a eu le plus de circulation, on utilise la commande `-> max(A(:,3))`.
2. La commande `sum(A,'c')` renvoie un vecteur colonne $C$ de taille $52 \times 1$, où la $i$-ème coordonnée contient la somme de la $i$-ème ligne de $A$, c'est à dire :

$$[C]_i = \sum_{j=1}^{7} [A]_{i,j}.$$

On doit chercher les indices des lignes où le coefficient de $C$ est maximal. On utilise pour cela la commande `find` :

```
--> find(C==max(C))
```

**Remarque.** De même, `sum(A,'r')` (`r` pour `row`) renverra un vecteur ligne de taille $1 \times 7$, où la $j$-ème coordonnée contient la somme de la $j$-ème colonne de $A$. Sur le même principe, on dispose aussi des commandes `max(A,'r')`, `min(A,'r')`, `max(A,'c')`, `min(A,'c')`.

3. La commande `-> find(A==min(A))` renvoie un vecteur dont les composantes sont les indices des coefficients où $A$ atteint son minimum (en parcourant la matrice de haut en bas et de gauche à droite). On cherche le nombre d'éléments dans ce vecteur, c'est à dire sa longueur, d'où la commande :

```
--> length(find(A = min(A)))
```

4. On peut utiliser la commande `mean` qui renvoie la moyenne des coefficients de $A$, d'où l'instruction `-> mean(A)`.

   En faisant plusieurs essaies, on constate que cette moyenne est proche de 5, ce qui n'est pas étonnant car l'espérance d'une variable suivant une loi de Poisson $\mathscr{P}(\lambda)$ est égale à $\lambda$, soit ici 5.

5. Nous n'aurons qu'une approximation de cette probabilité, mais on peut compter le nombre de jours où 8 voitures sont passées et diviser ce nombre par 364.

```
--> length(find(A>=8))/364
```

---

## Exercice 4.7 (★★)

1. Écrire un programme permettant de calculer $\displaystyle\sum_{k=1}^{n} \frac{(-1)^{k-1}}{k}$ pour une valeur de $n$ entrée par l'utilisateur. On proposera pour cela deux méthodes, l'une utilisant la fonction `sum`, l'autre à l'aide d'une boucle `for`.
2. On peut montrer que la série $\displaystyle\sum \frac{(-1)^{n-1}}{n}$ converge et que sa somme vaut $\ln(2)$. Écrire un programme permettant de déterminer le plus petit entier naturel $n$ pour lequel $|S_n - \ln(2)| \leq 10^{-3}$, où $(S_n)$ désigne la suite des sommes partielles de cette série.

1. **Première méthode.** On utilise une boucle `for` :

```
n = input("n ?")
S = 0
for k=1:n
      S = S + ((-1)^(k-1))/k
end
disp(S)
```

**Deuxième méthode.** À l'aide d'opérations pointées.

```
n = input("n ?")
u = (-1).^(0:(n-1))
```

```
v = (1:n)^(-1)
w = u.*v
disp(sum(w))
```

2. On utilise une boucle `while`.

```
S = 0
k = 0
while abs(S-log(2))>0,001
      k = k + 1
      S = S + ((-1)^(k-1))/k
end
disp(k)
```

---

**Exercice 4.8 (★★)**
Soient $(u_n)_{n\in\mathbb{N}^*}$ et $(v_n)_{n\in\mathbb{N}^*}$ les suites définies par : $\forall n \in \mathbb{N}^*$, $u_n = \sum_{k=1}^{n} \frac{1}{k^2}$ et $v_n = u_n + \frac{1}{n}$.

1. Montrer que les suites $(u_n)_{n\in\mathbb{N}^*}$ et $(v_n)_{n\in\mathbb{N}^*}$ sont adjacentes.
2. Écrire un programme qui donne une approximation de $\sum_{k=1}^{+\infty} \frac{1}{k^2}$ à $\varepsilon$ près pour $\varepsilon > 0$ donné.

1. On vérifie que :
   - $(u_n)$ est croissante (immédiat) ;
   - $(v_n)$ est décroissante : pour tout $n \in \mathbb{N}^*$, on a :
   $$u_{n+1} - u_n = \frac{1}{(n+1)^2} + \frac{1}{n+1} - \frac{1}{n} = \frac{1}{(n+1)^2} - \frac{1}{(n+1)n} \leq 0.$$
   - $(v_n - u_n)$ tend vers 0 (immédiat).

   Donc les suites $(u_n)$ et $(v_n)$ sont adjacentes. Par le théorème des suites adjacentes, on sait donc qu'elles convergent vers une même limite qu'on notera $\ell$ (dont on sait grâce à Euler qu'elle vaut $\frac{\pi^2}{6}$). De plus on a pour tout $n \in \mathbb{N}^*$ :
   $$u_n \leq \ell \leq v_n.$$
2. On cherche le premier $n \in \mathbb{N}$ tel que $v_n - u_n \leq \varepsilon$. On utilise donc une boucle `while`.

```
eps = input("epsilon ?")
n = 1
u = 1
v = 2
while v-u > eps
      n = n+1
      u = u+1/(n^2)
      v = u+1/n
```

```
end
disp(n)
```

## Exercice 4.9 (★★)

Ecrire un programme demandant une matrice $A$ à l'utilisateur, et renvoyant la trace de $A$. Votre programme devra afficher un message d'erreur si la matrice entrée par l'utilisateur n'est pas carrée.

```
A = input("une matrice ?")
[n,p] = size(A)
if n<>p then
      disp("A n'est pas carrée")
      else T = 0
            for k = 1:n
                  T = T + A(k,k)
            end
            disp(T)
end
```

## Exercice 4.10 (★★★)

1. (a) Établir que, lorsque $k$ est inférieur ou égal à $n$, on a :

$$\binom{n}{k} = \prod_{i=1}^{k} \left( \frac{n+1}{i} - 1 \right).$$

   (b) En déduire une ligne de commandes, établie à partir des vecteurs `1:k` et `ones(1,k)`, permettant de calculer $\binom{n}{k}$ lorsque $k$ et $n$ sont entrés par l'utilisateur.

   (c) Retrouver ce résultat avec la commande suivante (qu'on justifiera) :

```
cumprod((n :-1 :1)./(1 :n))
```

2. On désire créer une matrice d'ordre $n$ ($n \geq 1$) dont l'élément de la ligne $i+1$ et de la colonne $j+1$ est égal à $\binom{i}{j}$, avec $i$ et $j$ dans $[\![0, n-1]\!]$.

   (a) Créer une matrice $C$ d'ordre $n$ dont les éléments de la première colonne sont égaux à 1 (les autres éléments étant nuls).

   (b) En utilisant la formule de Pascal, écrire une ligne de commandes permettant de remplir les $n$ première lignes de la matrice $C$ (triangle de Pascal).

1. (a) On a :

$$\begin{aligned}\binom{n}{k} &= \frac{n!}{k!(n-k)!} = \frac{n(n-1)\dots(n-k+1)}{1\times 2\times\dots\times(k-1)k} \\ &= \frac{n+1-1}{1}\frac{n+1-2}{2}\dots\frac{n+1-(k-1)}{k-1}\frac{n+1-k}{k} = \prod_{i=1}^{k}\left(\frac{n+1}{i}-1\right).\end{aligned}$$

   (b) On peut utiliser le programme suivant :

```
k = input("k ?")
n = input("n ?")
u = (n+1)*ones(1,k)
v = (1:k)^(-1)
w = u.*v-1
if k>n then
      coeffbin = 0
      else coeffbin = prod(w)
end
disp(coeffbin)
```

(c) `[n:-1:1]` est le vecteur `[n,n-1,...,2,1]`. Et donc `(n :-1 :1)./(1 :n)` est le vecteur `[n/1,(n-1)/2,...,2/(n-1),1/n]`. Par conséquent, `cumprod((n :-1 :1)./(1 :n))` est le vecteur dont la $k$-ème composante est :

$$\frac{n}{1} \times \cdots \times \frac{n-k+1}{k} = \frac{n(n-1)\dots(n-k+1)}{k!} = \binom{n}{k}.$$

On peut en déduire le programme suivant :

```
k = input("k ?")
n = input("n ?")
if k==0 then
      coeffbin = 1
      else A = cumprod((n :-1 :1)./(1 :n)) ;
            coeffbin = A(k)
end
disp(coeffbin)
```

Notons que si l'on veut éviter le recours à une disjonction de cas, il existe une solution encore plus astucieuse :

```
k = input("k ?")
n = input("n ?")
A = [1,cumprod((n :-1 :1)./(1 :n))]
disp(A(k+1))
```

2. (a) On peut procéder ainsi :

```
n = input("n ?")
C = zeros(n,n)
C(:,1) = ones(n,1)
```

(b) On rappelle que $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$. Reste à remplir la matrice ligne par ligne à l'aide de cette relation :

```
n = input("n ?")
```

```
C = zeros(n,n)
C(:,1) = ones(n,1)
for k=2:n
      for j = 2:k
            C(k,j) = C(k-1,j-1)+C(k-1,j)
      end
end
disp(C)
```

---

Voici la réponse de `Scilab` lorsqu'on entre `n=6` :

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 \\ 1 & 3 & 3 & 1 & 0 & 0 \\ 1 & 4 & 6 & 4 & 1 & 0 \\ 1 & 5 & 10 & 10 & 5 & 1 \end{pmatrix}.$$