TP 1

Révisions : Matrices et programmation

Compétences attendues.

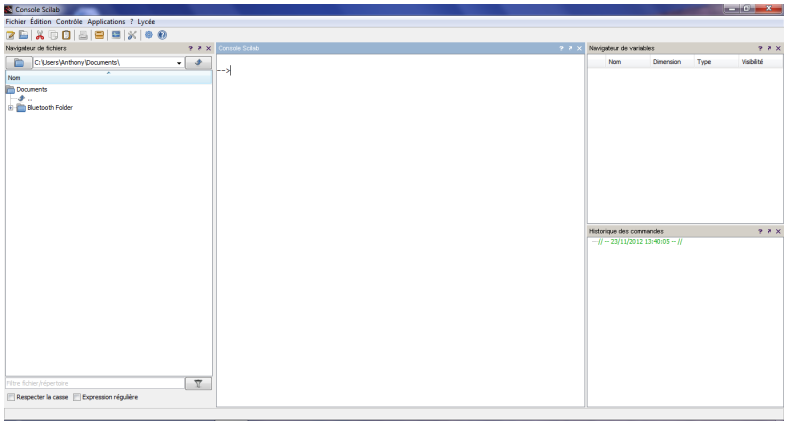
- ✓ Création d'une matrice, opérations sur les matrices.
- ✓ Programmer en `Scilab` : fonctions d'entrée et de sortie, boucles `if`, `for`, `while`.
- ✓ Calcul des termes d'une suite.
- ✓ Calcul d'une valeur approchée de la limite d'une suite ou de la somme d'une série.
- ✓ Évaluer la vitesse de convergence d'une suite ou d'une série en déterminant un rang d'arrêt.

Anthony Mansuy
Professeur de Mathématiques en deuxième année de CPGE filière ECE au Lycée Clemenceau (Reims)
Page personnelle : `http://anthony-mansuy.fr`
E-mail : `mansuy.anthony@hotmail.fr`

En préambule

Le logiciel `Scilab` est un système de calcul numérique. Créé en 2003 et développé par une entreprise française, il est libre et gratuit, à télécharger ici. Je vous demande de l'installer sur vos ordinateurs personnels.

Au lancement de `Scilab`, quatre fenêtres s'ouvrent :



1. `Navigateur de fichiers` : Il permet d'accéder aux fichiers `Scilab` déjà enregistrés,
2. `Console Scilab` : C'est « la feuille de travail » qui permet de taper des commandes et d'analyser les résultats,
3. `Navigateur de variables` : Il permet de parcourir et de modifier les variables stockées en mémoire,
4. `Historiques des commandes` : Toutes les commandes tapées dans la console lors de sessions `Scilab` précédentes sont gardées en mémoire.

En cas de besoin, vous disposez de l'aide `Scilab` qui indique la syntaxe précise de chaque instruction, accompagnée de nombreux exemples. Elle est accessible en cliquant sur la rubrique « ? » depuis la barre de menu ou en tapant directement 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`, `sqrt` (racine carrée), `floor` (partie entière) et `abs` (valeur absolue).
- **Constantes :** `%e` et `%pi`.

Exemple. Taper les instructions suivantes :

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

Remarques.

- Le dernier résultat obtenu est contenu dans la variable `ans`.
- On notera qu'une instruction suivie d'un point-virgule `;` est exécutée mais n'est pas affichée.
- Il n'est pas possible de modifier les instructions déjà entrées dans la console. Par contre, on peut rappeler d'anciennes commandes en utilisant les flèches du clavier ↑ et ↓.

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. Taper les instructions suivantes :

```
--> 2==3, 1.4 < sqrt(2), %pi^2<10, 1+sqrt(3)^2<>4
--> (1<2)&(2^2==4)
--> (log(%e)>=0)|(2^2==4)
```

Remarques.

1. Attention : pour tester l'égalité, il faut bien taper `==` et non `=` qui est une affectation !
2. Le « ou » est inclusif : par exemple, l'instruction `x==y | y==z` renvoie « vrai » si on a soit $x = y$, soit $y = z$, soit les 2.
3. Pour tester un encadrement, par exemple $x < y < z$, `Scilab` n'accepte pas qu'on utilise `x<y<z`. Il faudra penser à séparer en deux tests : `x<y & y<z`.

1.3 Les chaînes de caractères

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

Exemple. Taper l'instruction suivante :

```
--> "Mon message"
```

1.4 Les variables

Une **variable** est un nom dans lequel on peut stocker un objet `Scilab`, que ce soit une fonction, une expression ou une valeur particulière. Pour affecter une valeur à une variable, on utilise le symbole `=`.

Exemple. Taper les instructions suivantes :

```
--> t=log(2)
--> exp(t)^2
```

Lorsqu'une variable est créée, elle apparait dans la fenêtre `Navigateur de variables`. Il s'agira de rester vigilant car ces affectations sont **globales** et seront utilisées par `Scilab` dans toute la feuille de travail. 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`.

2 Les matrices

2.1 Création de matrices

Pour définir une matrice cœfficient par cœfficient, on encadre par des crochets, on sépare les éléments d'une même ligne par des virgules ou des espaces 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 $A = \begin{pmatrix} 1 & 2 & -1 & 2 \\ 3 & 7 & 8 & 4 \\ 6 & -4 & 2 & 5 \end{pmatrix}$.

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 $n \neq p$).
- `rand(n,p)` : matrice à n lignes et p colonnes dont les coefficients sont choisis au hasard dans $]0,1[$.
- `diag([a1,a2,...,an])` : matrice diagonale d'ordre n dont les coefficients diagonaux valent $a_1, a_2, \ldots, a_n$.

Exemple. Taper les instructions suivantes :

```
--> zeros(3,5), ones(3,2)
--> eye(3,5), rand(4,3)
--> diag([1,2,3])
```

2.2 Création de vecteurs

Définition.

- `[a:r:b]` (les crochets sont facultatifs) donne le vecteur-ligne $\begin{pmatrix} a & a+r & a+2r & a+3r & \ldots & a+nr \end{pmatrix}$, 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
```

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 $n2$ 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)
```

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]
```

2.4 Opérations sur les matrices

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 $\cdot$ 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.
- **Minimum coefficient par coefficient :** `min(A,B)` donne les minimums terme à terme de A et B.
- **Maximum coefficient par coefficient :** `max(A,B)` donne les maximums terme à terme de A et 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), sqrt(B)
--> min(A,B), max(A,B)
```

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 A.
- `length(A)` donne le nombre total d'éléments de A.
- `rank(A)` donne le rang de la matrice A.
- `spec(A)` donne le spectre de A (c'est-à-dire ses valeurs propres).
- `[P,D]=spec(A)` donne des matrices P inversible et D diagonale telles que $A = PDP^{-1}$.

Exemples. Définir la matrice $A = \begin{pmatrix} 1 & 2 & 3 & 5 \\ 5 & 6 & 7 & 9 \end{pmatrix}$ puis taper les instructions suivantes.

```
--> size(A), size(A,1), size(A,2), length(A)
--> sum(A), mean(A), max(A), min(A)
--> cumsum(A), cumprod(A)
```

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

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

```
--> A==2
--> A<0
```

Expliquer les résultats obtenus avec `Scilab`.

Définition.

Si B est une matrice de booléens alors :

- `find(B)` donne les indices des éléments vrais dans B (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.
- `sum(B)` donne le nombre d'éléments vrais dans B.

Exemples. Taper les instructions suivantes :

```
--> find(A==2)
--> [n,p]=find(A==2)
--> sum(A>=3)
```

3 Programmer en Scilab

3.1 Les scripts

Taper directement dans la console a deux inconvénients : l'enregistrement n'est pas possible, et si plusieurs lignes d'instructions ont été tapées, les modifications ne sont pas aisées. 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` ;
- 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 en cliquant sur ▷ à partir de `SciNotes`.

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

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

Enregistrer votre fichier sous le nom `essai.sce` puis l'exécuter.

Conseils de mise en forme des scripts.
L'éditeur reconnait le langage `Scilab` : il met en couleur les mots-clef, un décalage de début de ligne appelé indentation se fait automatiquement lorsqu'on commence une boucle ou un test, il ferme automatiquement les parenthèses ouvertes... Pour que les scripts soient rapidement compréhensibles, on respectera les quelques règles suivantes :

- 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 //. Tout ce qui suit // ne sera pas exécuté.
- 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("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.

Pour effectuer un bloc d'instructions sous certaines conditions, on utilise une boucle `if`. Voici la syntaxe :

`if` $condition_1$ `then`
  $instructions_1$
`elseif` $condition_2$ `then`
  $instructions_2$
...
`elseif` $condition_{k-1}$ `then`
  $instructions_{k-1}$
`else`
  $instructions_k$
`end`

Si la $condition_1$ est vraie, le premier bloc d'instructions ($instructions_1$) est effectué. Si la $condition_1$ est fausse, on fait un deuxième test. Si la $condition_2$ est vraie, le deuxième bloc d'instructions ($instructions_2$) est effectué. Ce procédé est répété jusqu'à la $condition_{k-1}$. Si la $condition_{k-1}$ est vraie, le bloc d'instructions ($instructions_{k-1}$) est effectué. Si la $condition_{k-1}$ est fausse, le dernier bloc d'instructions ($instructions_k$) est effectué.

Exemple. Taper dans `SciNotes` le script suivant 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.

Pour répéter un bloc d'instructions un nombre déterminé de fois, on utilise une boucle `for`. La syntaxe est la suivante :

`for` $variable$ `=` $début$`:`pas`:`fin
  $instructions$
`end`

La $variable$ parcourt dans l'ordre les entiers de $début$ à fin en suivant le pas régulier et, à chaque étape, les $instructions$ sont effectuées.

Exemple. Taper dans `SciNotes` le script suivant puis l'exécuter. Que calcule-t-il ?

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

3.5 Boucle while

Définition.

Pour répéter un bloc d'instructions tant qu'une condition est vérifiée, on utilise une boucle `while`. La syntaxe est la suivante :

```
while condition
  instructions
end
```

Tant que la *condition* est vérifiée, les *instructions* sont effectuées. Dès que la *condition* n'est plus vérifiée, les *instructions* sont ignorées.

Exemple. Taper dans `SciNotes` le script suivant 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 1

1. Sans rentrer les cœfficients un à un, déclarer les matrices suivantes :

$$A = \begin{pmatrix} 5 & 3 & 3 \\ 3 & 5 & 3 \\ 3 & 3 & 5 \end{pmatrix} \qquad \text{et} \qquad B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}.$$

2. (a) Que vaut le produit d'un vecteur colonne $\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$ par le vecteur ligne $\begin{pmatrix} 1 & \dots & 1 \end{pmatrix}$?

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

Exercice 2

On considère les matrices suivantes :

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 1 & -2 & 3 \\ 2 & 1 & 5 \end{pmatrix} \qquad \text{et} \qquad B = \begin{pmatrix} 4 & -2 & -2 \\ 1 & 1 & -2 \\ 2 & -4 & 2 \end{pmatrix}$$

Définir chacune de ces matrices sur `Scilab`, puis appliquer l'algorithme du pivot de Gauss pour démontrer si elles sont inversibles ou non (on effectuera chacune des opérations élémentaires sur `Scilab`). Vérifier votre résultat à l'aide de la fonction `inv`.

Exercice 3

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

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

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)))`

Exercice 4

Pour toute matrice $A = (a_{i,j}) \in \mathscr{M}_n(\mathbb{R})$, on pose : $Tr(A) = \sum_{i=1}^{n} a_{i,i}$. On dit que $Tr(A)$ est la trace de A.

Écrire 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.

Exercice 5

L'objectif de cet exercice est de proposer différentes méthodes pour calculer $\binom{n}{k}$ avec `Scilab`.

1. Écrire un programme qui, lorsque k et n sont entrés par l'utilisateur, calcule et affiche $\binom{n}{k}$ en utilisant la formule explicite des coefficients binomiaux.

2. (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 un programme faisant intervenir les vecteurs `1:k` et `ones(1,k)`, permettant de calculer $\binom{n}{k}$ lorsque k et n sont entrés par l'utilisateur.

3. (a) Déterminer ce que calcule la commande suivante :

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

   (b) En déduire un programme permettant de calculer $\binom{n}{k}$ lorsque k et n sont entrés par l'utilisateur.

4. (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, compléter la matrice C pour que, pour tout i et j dans $[\![0, n-1]\!]$, l'élément de la ligne $i+1$ et de la colonne $j+1$ soit égal à $\binom{i}{j}$.

   (c) En déduire un programme permettant de calculer $\binom{n}{k}$ lorsque k et n sont entrés par l'utilisateur.

Exercice 6

Écrire un programme qui demande trois réels a, b et c et qui affiche l'ensemble des racines réels du polynôme du second degré $P(x) = ax^2 + bx + c$. Votre programme devra afficher le message "P n'admet pas de racine" si le discriminant est strictement négatif.

Exercice 7

1. Écrire un programme permettant de calculer $\sum_{k=1}^{n} \frac{(-1)^{k-1}}{k}$ 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 $\sum \frac{(-1)^{n-1}}{n}$ 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.

Exercice 8

Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par $u_0 = 20$ et, pour tout entier naturel n, $u_{n+1} = \ln(u_n^2)$.

1. Écrire un programme qui, étant donné un entier naturel n, calcule et affiche la valeur de u_n.
2. Écrire un programme qui permet de déterminer le plus petit entier n tel que $u_n < 0$.

Exercice 9

On considère la suite $(u_n)_{n\in\mathbb{N}^*}$ définie par : $\forall n \in \mathbb{N}^*$, $u_n = \displaystyle\sum_{k=1}^{n} \frac{1}{\sqrt{k}}$.

1. Montrer que, pour tout entier $k \geq 1$, $\sqrt{k+1} - \sqrt{k} \leq \dfrac{1}{2\sqrt{k}}$.
2. En déduire que : $\forall n \in \mathbb{N}^*$, $u_n \geq 2(\sqrt{n+1} - 1)$. Quelle est la nature de la suite $(u_n)_{n\in\mathbb{N}^*}$?
3. Écrire un programme qui permet de déterminer le plus petit entier n tel que $u_n \geq 10^3$.

Exercice 10

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

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 $\displaystyle\sum_{k=1}^{+\infty} \frac{1}{k^2}$ à ε près pour $\varepsilon > 0$ donné.
