TP4/5 : Écriture de sommes finies en **Scilab**

Pré-requis : l'objectif des premières séances de TP est de faire le point sur des fonctionnalités importantes de **Scilab** qui ont été vues en première année. Je vous invite à consulter les chapitres de cours correspondants sur ma page : support informatique.
On pourra en particulier se reporter au « CH 7 : Les structures itératives ».

- Dans votre dossier `Info_2a`, créer le dossier `TP_4`.

I. Avant propos

On considère dans ce TP les suites $(u_n)_{n\in\mathbb{N}}$ et (v_n) suivantes :

$$\forall n \in \mathbb{N},\ u_n = \frac{n^2}{3^n} \qquad \text{et} \qquad \begin{cases} v_0 = 1 \\ \forall n \in \mathbb{N},\ v_{n+1} = \dfrac{2\,v_n}{\mathrm{e}^{v_n} + \mathrm{e}^{-v_n}} \end{cases}$$

Objectif du TP : il s'agit d'explorer les différentes méthodes permettant le calcul des sommes partielles d'ordre n : $S_n = \sum_{k=0}^{n} u_k$ et $T_n = \sum_{k=0}^{n} v_k$.

II. Calcul des sommes partielles d'ordre n

II.1. Calcul de S_n

Il s'agit ici d'illustrer le cas où la suite (u_n) est donnée sous forme explicite.

II.1.a) Méthode itérative

- Écrire une fonction `calculSn` qui :
  - × prend en paramètre un entier `n`,
  - × renvoie une variable `S`,
  - × à l'aide d'une structure itérative, calcule la valeur de S_n et stocke le résultat dans `S`.

```
function S = calculSn(n)
    S = 0
    for i = 0:n
        S = S + i^2 / (3^i)
    end
endfunction
```

- Que vaut S_0 ? S_5 ? S_{10} ? S_{100} ? S_{1000} ?

On trouve : $S_0 = 0$, $S_5 \simeq 1.4115$, $S_{10} \simeq 1.4989$, $S_{100} \simeq 1.5$ et $S_{1000} \simeq 1.5$.
(noter que l'affichage du résultat se fait avec un nombre fixé de chiffres après la virgule)

II.1.b) Utilisation des fonctionnalités Scilab

- Écrire une fonction `premSuiteU` qui :
  - × prend en paramètre un entier `n`,
  - × renvoie une variable `U`, vecteur contenant initialement `n` + 1 zéros,
  - × à l'aide d'une structure itérative, calcule les `n` + 1 premières valeurs de la suite (u_n) et stocke le résultat dans `U`.

```
function U = premSuiteU(n)
    U = zeros(1,n+1)
    for i = 1:n+1
        U(i) = i^2 / (3^i)
    end
endfunction
```

- Que réalise l'appel `sum(1:5)` ? Détailler le rôle de la fonction `sum`.

La fonction `sum` prend en paramètre une matrice et renvoie la somme de tous ses cœfficients. Ici, on considère le vecteur $[1, 2, 3, 4, 5]$ dont la somme vaut 15.

- En déduire un appel permettant de calculer S_{10}.

```
sum(premSuiteU(10))
```

- Que réalise l'appel `1 ./ (1:5)` ? Et `(4:8) ./ (1:5)` ? Détailler le rôle de l'opérateur `./`

L'opérateur `./` est l'opérateur de division terme à terme.

- Dans le premier cas, on obtient une valeur approchée de $[\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}]$.
- Dans le deuxième cas, les deux matrices étant de même taille, on obtient une matrice de même taille où les cœfficients sont obtenus par division des cœfficients en même position. Plus précisément, on obtient une valeur approchée de $[\frac{4}{1}, \frac{4}{2}, \frac{4}{3}, \frac{4}{4}, \frac{4}{5}]$.

- De même, que réalise l'appel `[1,2; 1,1]^2` ? Et l'appel `[1,2; 1,1].^2` ? Et `(1:5)^2` ?

- L'opérateur `.^` est l'opérateur de puissance terme à terme. L'opérateur `^` est l'opérateur de puissance mathématique classique.
- L'appel `(1:5)^2` est donc impropre : on ne peut mutiplier la matrice $[1, 2, 3, 4, 5]$ par elle-même du fait de ses dimensions. Cependant, **Scilab** récupère cette erreur en élevant au carré tous les cœfficients de $[1, 2, 3, 4, 5]$ ce qui correspond en fait à l'appel `(1:5).^2`.

- Comment peut-on, par simple manipulation matricielle, créer le vecteur `U` des 101 premiers éléments de la suite (u_n) ? On commencera par stocker le vecteur `0:100` dans une variable `N`.

```
N = 0:100; U = (N .^2) ./ (3 .^N)
```
(l'appel `U = (N^2) ./ (3 ^N)` *convient aussi)*

▶ Comment obtient-on alors S_{100} à l'aide de `U` ? Et comment récupérer S_5 ?

- La valeur de S_{100} est donnée par l'appel `sum(U)`.
- S_5 est la somme des 6 premiers éléments de (u_n) (ne pas oublier 0). On récupère le vecteur contenant ces 6 éléments par l'appel `U(1:6)` et donc S_5 par l'appel `sum(U(1:6))`.

II.2. Calcul de T_n

Il s'agit ici d'illustrer le cas où la suite (u_n) est donnée sous forme récurrente.

II.2.a) Méthode itérative

▶ Écrire une fonction `calculTn` qui :
- × prend en paramètre un entier `n`,
- × renvoie une variable `T`,
- × à l'aide d'une structure itérative, calcule la valeur de $S_{\mathtt{n}}$ et stocke le résultat dans `T`.

On pourra utiliser une variable auxiliaire `v` afin de calculer les différents termes de (v_n).

```
function T = calculTn(n)
    T = 1
    v = 1
    for i = 1:n
        v = 2 * v / (exp(v) + exp(-v))
        S = S + v
    end
endfunction
```

▶ Que vaut T_0 ? T_{100} ? T_{10000} ?

On trouve : $T_0 = 1$, $T_{100} \simeq 18.18$, $T_{10000} \simeq 197.58$.

II.2.b) Utilisation des fonctionnalités Scilab

▶ Écrire une fonction `premSuiteV` qui :
- × prend en paramètre un entier `n`,
- × renvoie une variable `V`, vecteur contenant initialement `n` + 1 zéros,
- × à l'aide d'une structure itérative, calcule les `n` + 1 premières valeurs de la suite (v_n) et stocke le résultat dans `V`.

```
function V = premSuiteV(n)
    V = zeros(1,n+1)
    V(1) = 1
    for i = 1:n
        V(i+1) = 2 * V(i) / (exp(V(i)) + exp(-V(i)))
    end
endfunction
```

▶ En déduire un appel permettant de calculer T_{10}.

```
sum(premSuiteV(10))
```

III. Calcul des n premières sommes partielles

III.1. Calcul des n premières sommes partielles de $\sum u_n$

▶ Soit $n \in \mathbb{N}$. Quel lien y a-t-il entre S_n et S_{n+1} ?

$$S_{n+1} = S_n + \frac{(n+1)^2}{3^{n+1}}$$

▶ En tirant profit de l'égalité précédente, écrire une fonction `premSn` qui :

× prend en paramètre une variable `n`,

× renvoie une variable `tabS`, vecteur contenant initialement `n` + 1 zéros,

× à l'aide d'une structure itérative, stocke la valeur de S_i dans la $i^{\text{ème}}$ case de `tabS`.

On ne devra pas effectuer d'appel à `calculSn` mais on pourra s'inspirer de son code.

```
function tabS = premSn(n)
    tabS = zeros(1, n+1)
    tabS(1) = 0
    for i = 1:n
        tabS(i+1) = tabS(i) + (i+1)^2 / (3^(i+1))
    end
endfunction
```

▶ Comme précédemment, on aurait aussi pu tirer parti des fonctionnalités de **Scilab**.
Que réalise l'appel `cumsum(1:5)` ? Détailler le rôle de la fonction `cumsum`.

- La fonction `cumsum` (*Cumulative Sum*) prend en paramètre un vecteur `u` et renvoie un vecteur `v` de même taille dont le $i^{\text{ème}}$ cœfficient est la somme des i premiers éléments de `u`.
- Ainsi, `cumsum(1:5)` renvoie le vecteur $[1, 3, 6, 10, 15]$.

▶ Quel appel, tirant parti des fonctionnalités **Scilab**, permet d'obtenir les 101 premiers éléments de la suite (S_n) ? On utilisera la fonction `cumsum`.

- Si on a codé la fonction `premSuiteU`, on peut réaliser l'appel : `cumsum(premSuiteU(100))`.
- Sinon on peut, commme précédemment, créer le vecteur qui contient les 101 premiers éléments de la suite (u_n) : `N = 0:100; U = (N .^2) ./ (3 .^N); cumsum(U)`.

III.2. Calcul des n premières sommes partielles de $\sum v_n$

▶ Soit $n \in \mathbb{N}$. Quel lien y a-t-il entre T_n et T_{n+1} ?

$T_{n+1} = T_n + v_{n+1}$

▶ En tirant profit de l'égalité précédente, écrire une fonction `premTn` qui :

× prend en paramètre une variable `n`,

× renvoie une variable `tabT`, vecteur contenant initialement `n` + 1 zéros,

× à l'aide d'une structure itérative, stocke la valeur de T_i dans la $i^{\text{ème}}$ case de `tabT`.

On ne devra pas effectuer d'appel à `calculTn` mais on pourra s'inspirer de son code. On pourra notamment utiliser une variable `v` calculant les valeurs successives des termes de la suite (v_n).

```
function tabT = premTn(n)
    tabT = zeros(1, n+1)
    v = 1
    tabT(1) = 1
    for i = 1:n
        v = 2 * v / (exp(v) + exp(-v))
        tabT(i+1) = tabT(i) + v
    end
endfunction
```

À retenir : on a étudié deux méthodes en **Scilab** pour obtenir les éléments de (S_n).

1) La suite des sommes partielles étant une suite (grande nouvelle), on peut se servir des procédés vus en TP2.

2) On peut aussi préférer créer le vecteur des premiers éléments de la suite (u_n) et utiliser les instructions `sum` ou `cumsum` suivant ce que l'on cherche à obtenir. Encore une fois, il faut se reporter au TP2.

IV. Tracé des sommes partielles de $\sum u_n$

▶ Compléter le programme suivant afin qu'il permette d'effectuer le tracé des 51 premiers éléments de (S_n). On exécutera ce programme.

```
N = 0:50
U = (N .^2) ./ (3 .^N)
tabS = cumsum(U)
plot(N, tabS, 'rx')
```

▶ Quelle conjecture peut-on émettre sur la nature de la série $\sum u_n$?

D'après la représentation graphique, on peut émettre l'hypothèse que $\sum u_n$ est une série convergente, de somme $\frac{3}{2}$.

V. Suite des sommes partielles aux concours

V.1. ECRICOME 2015

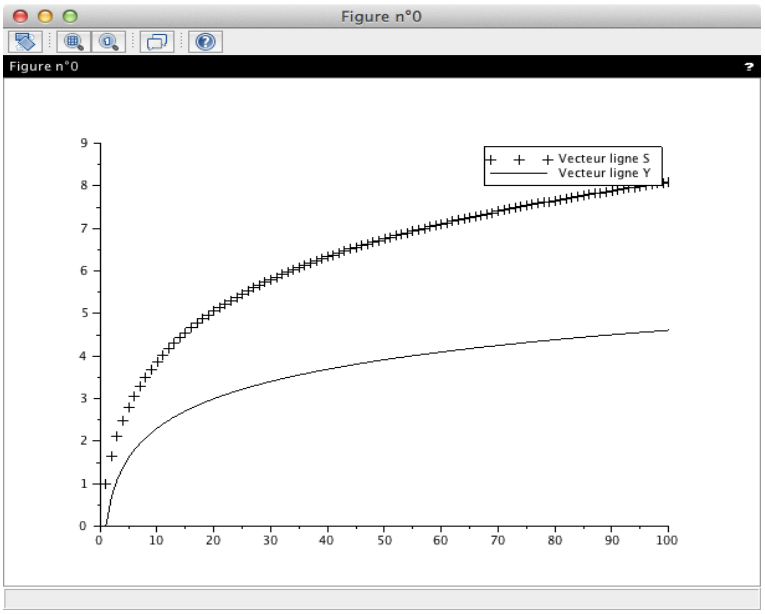
En TP2, on a déjà introduit l'épreuve `ECRICOME 2015` qui commençait par l'étude d'une suite récurrente $(u_n)_{n\in\mathbb{N}^*}$, définie par une relation de récurrence de type $u_{n+1} = F(u_n)$.
La première question, consistait à compléter le programme permettant le calcul des 100 premiers éléments de (u_n). On rappelle ce programme ci-dessous.

```
U = zeros(1,100)
U(1) = 1
for n = 1 : 99
  U(n+1) = 1 - exp(-U(n))
end
plot(U,"+")
```

► *On modifie le programme précédent en remplaçant la dernière ligne par :*

```
X = 1 : 100
S = cumsum(U)
Y = log(X)
plot2d(X,S)
plot2d(X,Y)
```

Le programme ci-dessus permet d'obtenir la représentation graphique suivante :



► *Que représente le vecteur-ligne* `S` *?*
Quelle conjecture pouvez-vous émettre sur la nature de la série de terme général u_n ?

- Le vecteur `U` contient les 100 premiers éléments de la suite (u_n). L'opérateur `cumsum` permet de calculer la somme cumulée de ce vecteur. Ainsi, `S` contient les 100 premières sommes partielles de la série $\sum u_n$.
- D'après le tracé obtenu, on peut émettre l'hypothèse que la série $\sum u_n$ est divergente. Plus précisément, que $S_n \underset{n\to+\infty}{\longrightarrow} +\infty$.

▶ *À l'aide de la question **2.h)*** (consistait à démontrer : $\forall n \in \mathbb{N}^*,\ u_n \geqslant \frac{1}{n}$) *établir la nature de la série de terme général* u_n.

On sait :

× $\forall n \in \mathbb{N},\ 0 \leqslant \frac{1}{n} \leqslant u_n$.

× La série $\sum_{n\geqslant 1} \frac{1}{n}$ est une série de Riemann d'exposant 1 ($1 \not> 1$).

Ainsi, la série $\sum_{n\geqslant 1} \frac{1}{n}$ est divergente.

Donc, par le théorème de comparaison des séries à termes positifs, la série $\sum u_n$ diverge.

V.2. EML 2015

On considère l'application $f : \mathbb{R} \to \mathbb{R},\ x \mapsto f(x) = x^3\, e^x$
et la suite réelle $(u_n)_{n\in\mathbb{N}}$ *définie par :* $u_0 = 1$ *et* $\forall n \in \mathbb{N}, u_{n+1} = f(u_n)$.

Il était demandé de démontrer que la série $\sum_{n\geqslant 1} \frac{1}{f(n)}$ converge (de somme S) et que :

$$\forall n \in \mathbb{N}^*,\ \left| S - \sum_{k=1}^{n} \frac{1}{f(k)} \right| \leqslant \frac{1}{(\mathrm{e}-1)\,\mathrm{e}^n}$$

▶ *En déduire une fonction **Scilab** qui calcule une valeur approchée de* S *à* 10^{-4} *près.* (*cf* TP3)

- Si on sait : $\frac{1}{(\mathrm{e}-1)\,\mathrm{e}^n} \leqslant 10^{-4}$, on obtient par transitivité : $|S - S_n| \leqslant 10^{-4}$.

  L'idée est donc de trouver le premier entier tel que : $\frac{1}{(\mathrm{e}-1)\,\mathrm{e}^n} \leqslant 10^{-4}$.

  On peut démontrer que cet entier vaut : $\lceil 4\ln(10) - \ln(\mathrm{e}-1)\rceil$ $(= 9)$ et ainsi, utiliser une boucle `for` pour calculer S_9 qui fournit l'approximation souhaitée.

```
function S = calcApprochS()
    n = ceil(4 * log(10) - log(exp(1)-1))
    S = 0
    for k = 1:n
        S = S + 1 / (k^3 * exp(k))
    end
endfunction
```

- Le deuxième choix est de calculer les valeurs successives de (S_n) tant que $\frac{1}{(\mathrm{e}-1)\,\mathrm{e}^n} \leqslant 10^{-4}$ **n'est pas** vérifiée *i.e.* tant que $\frac{1}{(\mathrm{e}-1)\,\mathrm{e}^n} > 10^{-4}$.

```
function S = calcApprochS()
    n = 1
    S = 1 / exp(1)
    while 1 / ((exp(1)-1) * exp(n)) > 10 ^(-4)
        n = n + 1
        S = S + 1 / (n^3 * exp(n))
    end
endfunction
```

V.3. ECRICOME 2018

Pour tout entier naturel n non nul, on pose : $u_n = \sum_{k=1}^{n} \frac{1}{k} - \ln(n)$.

▸ *Écrire une fonction d'en-tête :* `function y = u(n)` *qui prend en argument un entier naturel n non nul et qui renvoie la valeur de u_n.*

```
1  function y = u(n)
2      S = 0
3      for k = 1:n
4          S = S + 1/k
5      end
6      y = S - log(n)
7  endfunction
```

Détaillons les différents éléments de ce code :

× en ligne 2, on crée la variable `S` dont le but est de contenir, en fin de programme $\sum_{k=1}^{n} \frac{1}{k}$.
Cette variable `S` est donc initialisée à 0.

× de la ligne 3 à la ligne 5, on met à jour la variable `S` à l'aide d'une boucle.
Pour ce faire, on ajoute au $k^{\text{ème}}$ tour de boucle la quantité $\frac{1}{k}$.
Ainsi, `S` contient bien $\sum_{k=1}^{n} \frac{1}{k}$ en sortie de boucle.

× en ligne 6, on affecte à la variable `y` la valeur $u_n = \sum_{k=1}^{n} \frac{1}{k} - \ln(n)$.

Commentaire

Pour le calcul de la somme $\sum_{k=1}^{n} \frac{1}{k}$, on peut aussi tirer profit des fonctionnalités **Scilab** :

```
S = sum(1 ./ 1:n)
```

Pour bien comprendre cette instruction, rappelons que :
× l'instruction `1:n` permet de créer la matrice ligne $(1 \quad 2 \quad \dots \quad n)$.
× l'opérateur `./` permet d'effectuer la division terme à terme.
Ainsi, l'instruction `1 ./ 1:n` permet de créer la matrice ligne $(\frac{1}{1} \quad \frac{1}{2} \quad \dots \quad \frac{1}{n})$.
× la fonction `sum` permet de sommer tous les coefficients d'une matrice.
On obtient donc bien la somme à calculer par cette méthode.