Lycée Carnot ECS1B – 2018/2019

Travaux Pratiques d'Informatique

Première partie

Matthias Gorny

Table des matières

Introduction 3

1 Découverte de Scilab 4

2 Premiers programmes en Scilab 8

3 Structures conditionnelles 10

4 Structures répétitives : boucle `for` 11

5 Structures répétitives : boucle `while` 14

6 Fonctions en Scilab 16

7 Représentation graphique avec Scilab 18

8 Matrices et systèmes avec Scilab 22

Introduction

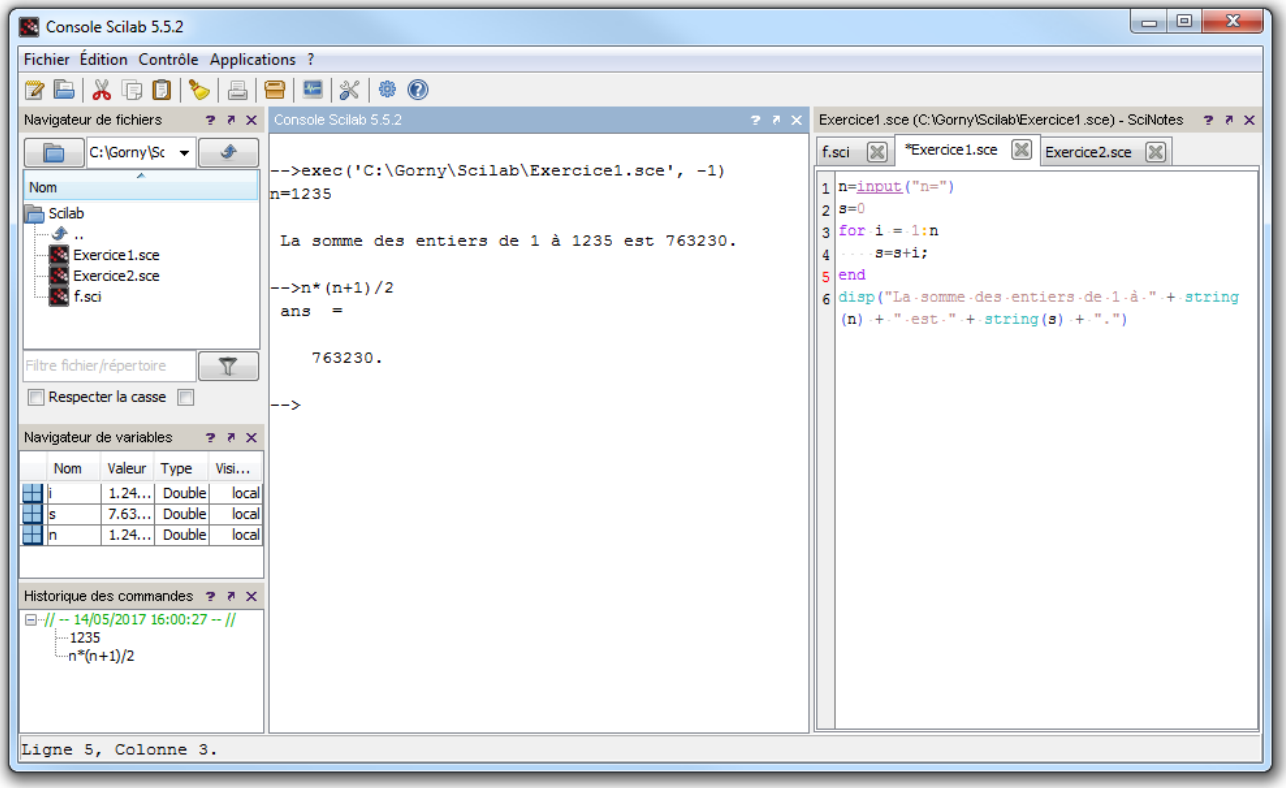
La plupart des feuilles de TP sont découpées en deux parties :

- une partie « entrainement » dont le but est d'aider la compréhension du cours d'informatique et d'algorithmique. Il est donc indispensable d'avoir lu attentivement le cours avant la séance de TP.
- une partie regroupant plusieurs exercices d'application, de difficultés variables.

Quelques instructions :

- Lorsque l'on entame une nouvelle feuille de TP, commencez par créer le dossier `..\ECS1B_TPInfo\TP**\` (en replaçant ** par le numéro du TP) et faites-en le répertoire courant de Scilab. Tous les scripts devront être sauvegardés dans ce dossier.
- Avant de commencer un exercice, tapez la commande *clear* dans la console.
- Pour vous y retrouver plus facilement, veuillez écrire
  - en commentaire au début de chaque programme le numéro du TP et de l'exercice concerné.
  - sur la feuille de TP, le nom que vous avez donné au programme (extension en *.sce*).
- Vous devez tester dans la console Scilab tous les programmes que vous écrivez (sans oublier de les exécuter au préalable).

Pour faciliter ma lecture de vos écrans, veuillez organiser les fenêtres de votre environnement Scilab comme ceci :



Feuille de TP n° 1

Découverte de Scilab

Lors de votre inscription au lycée Carnot, un compte informatique à votre nom a été créé. Votre login est composé de la première lettre de votre prénom, suivi de votre nom (en minuscule et sans espace). Votre mot de passe est votre date de naissance sous la forme : *jjmmaaaa*.

Une fois connecté sur votre compte, créez un dossier intitulé `ECS1B_TPInfo`. Dans ce dossier, créez un nouveau dossier intitulé `TP1`.

I Prise en main de Scilab

• Ouvrir Scilab et constater que que la fenêtre qui apparaît se compose de quatre sous-fenêtres : la *console Scilab*, le *navigateur de fichier*, le *navigateur de variables* et l'*historique des commandes*.

• Repérer les quatre bandes bleues (ou noires selon le système d'exploitation utilisé) sur lesquelles se trouvent les noms des sous-fenêtres. Cliquer sur la petite croix du navigateur de variables. Puis aller sur *Applications* et cliquer sur *Navigateur de variables* pour le faire réapparaitre .

• Fermer la console. Que se passe-t-il ?

• Réorganiser l'environnement de travail en plaçant le navigateur de variables et l'historique en dessous du navigateur de fichier (il suffit de maintenir le clic gauche de la souris enfoncé sur la bande bleue ou noire pour déplacer la fenêtre à l'endroit voulu).

• Dans le navigateur de fichier, sélectionner le dossier `..\ECS1B_TPInfo\TP1\` qui devient alors le répertoire courant de Scilab, c'est-à-dire le répertoire dans lequel se trouvent les fichiers avec lesquels on travaille

• Cliquer sur l'icône pour ouvrir l'**Aide en ligne** dans une nouvelle fenêtre (on peut également taper la commande `help` dans la console).

II Calculs sur les nombres réels

• Pour exécuter une commande, on place le curseur après le symbole `- ->` dans la console Scilab, on rentre la commande et on appuie sur la touche Entrée pour que cette commande soit interprétée. Exécuter les commandes suivantes :

```
-->2
-->5
```

```
-->2+5
-->2-5
```

```
-->2*5
-->2/5
```

```
-->2^5
-->2**5
```

• Placer le curseur sur `- ->` sans écrire de commande. Utiliser plusieurs fois les touches ↑ et ↓. Qu'est ce qu'elles font ?

• Jeter un œil sur l'historique des commandes et constater qu'il résume les commandes effectuées dans la console. Effectuer un clic droit sur la console et cliquer sur *Effacer l'historique*.

• Exécuter les commandes suivantes :

```
-->2.5
-->2,5
-->2.5*4.6, 32/128
```

A quoi servent le point et la virgule ?

En n'utilisant qu'une seule ligne de code, calculer $85 - 71 - 36$, 17×19, $4/3$, 2^8 et $3 + 14159 \cdot 10^{-5}$.

• Comme en mathématiques, les opérations ont un certain ordre de priorité. Il faut donc utiliser des parenthèses. Exécuter les commandes suivantes :

```
-->345/7/3, (345/7)/3, 345/(7/3),
-->216/28, 216/7*4, 216/4*7, (216/7)*4, (216/4)*7, 216/(4*7)
-->7^15, 7^3*5, 7^5*3, (7^5)*3, 7^(3*5)
-->6^2^4, (6^2)^4, 6^(2^4)
```

Calculer $\frac{265}{9*13}$, $7^{3.6*2.51}$, $4^{5/3}$, 9^{8^7} et $-7+\frac{4}{7-\frac{4+7}{4\times 7}}$.

• Exécuter les commandes suivantes :

```
-->%e, %e^2, %e^(-1)
-->%pi, %pi/2, %pi/3, %pi/4
```

• Exécuter les commandes suivantes :

```
-->sqrt(2), sqrt(3), sqrt(4), sqrt(3)^2, sqrt(7.9^2)
-->exp(1), %e, log(2), log(exp(-12.5)), exp(log(14.3)), log(0)
-->floor(%pi), floor(-5.67542)
-->abs((-1)^53)
-->cos(%pi/2), cos(%pi/3), sin(%pi/4), sin(%pi/6)
-->(cos(4.76))^2+(sin(4.76))^2
-->tan(%pi), tan(%pi/3)
```

Calculer $\tan\left((\pi+e)^{\sqrt[7]{2}}\right)$ et $\cos\left(\frac{\pi}{3}\left\lfloor \ln\left(\left|1-e^{\sqrt{19}}\right|\right)\right\rfloor\right)$.

• Scilab permet également la manipulation des nombres complexes. Exécuter les commandes suivantes :

```
-->%i, %i^2
-->(3+2*%i)*(4-5*%i), (6*%i-8)/(7-%i)
-->exp(%i*%pi)+1, exp(%i*%pi/2), exp(%i*%pi/3), exp(%i*%pi/4)
-->sqrt(-5)
-->abs(8-7*%i), sqrt(8^2+(-7)^2)
```

Calculer $\frac{(1+i)^{17}-1}{(1+i)^{17}+1}$ et $\left(\frac{3-13i\sqrt{2}}{i-9}\right)^{19}$.

• Exécuter les commandes suivantes :

```
-->exp(709)
-->exp(710)
-->%inf
-->help %inf
```

```
-->1/%inf
-->1/0
-->%eps
-->help %eps
```

• Exécuter `help format` dans la console. A quoi sert la commande `format` ?

III Création de variables par affectation

• Exécuter les commandes suivantes :

```
-->exp(%pi)
-->ans
-->log(2)
-->ans
```

Que contient la variable `ans` ?

• Exécuter les commandes suivantes :

```
-->x
-->x=5
-->x
-->2*x-11
```

```
-->x
-->x=2*x-11
-->x
-->y=3, z=x+y, x, y, z
```

• Jeter un œil sur le navigateur de variables et constater qu'il résume toutes les variables affectées jusqu'ici (`ans, x, y, z`). Exécuter les commandes suivantes :

```
-->clear z
-->ans, x, y, z
-->clear
-->ans, x, y, z
```

Que fait la commande `clear` ? Que contient désormais le navigateur de variables ?

• Exécuter les commandes suivantes :

```
-->a=4, b=5, c=b^a
-->a=4; b=5; c=b^a
```

A quoi sert le point virgule ? Quelle est sa différence par rapport à la virgule ?

• Pour chacune des instructions suivantes, anticiper ce qui sera affiché dans la console. Puis exécuter pour vérifier :

```
-->x=1/2; x=2*x+1; x=2*x+1; x=2*x+1
-->y=7; y=1/y; y=1/y ; y=1/y; y=1/y
-->z=2; z=1-z^2; z=1-z^2 ; z=1-z^2
-->t=4; t=t*%i; t=t*%i; t=t*%i
```

• Exécuter les commandes suivantes :

```
-->var=2; Var=3; var, Var,
-->var1=5, var_1=5, var!=5, var$=5, var#=5, var?=5
-->var%=5, var.1=5, 1var=5, var-1=5,
-->anticonstitutionnellement=5
```

• **Exercice :** Si a et b sont deux variables, quelle suite d'instructions permet à Scilab d'échanger le contenu de a et b ?

• Affecter à la variable %j la valeur du complexe $j = e^{2i\pi/3}$. Que valent j^3 et $1 + j + j^2$?

IV Variables booléennes et opérateurs de comparaison

• Exécuter les commandes suivantes :

```
-->clear
-->%t, %f
-->typeof(%f)
-->~%t, ~%f
-->%t|%t, %t|%f, %f|%t, %f|%f
-->%t&%t, %t&%f, %f&%t, %f&%f
```

• **Exercice :** Vérifier les lois de Morgan avec Scilab.

• Exécuter les commandes suivantes :

```
-->1==2
-->1=2
-->x=%e^2; log(x)<>2
-->y=%pi/4; z=(y>=0)&(y<1); y
```

• **Exercice :** On affecte un nombre réel à une variable x. Écrire une instruction booléenne en Scilab, faisant intervenir x, qui est vraie si et seulement si x appartient à l'intervalle $[-1, 2]$. Même question avec $]-\infty, 1] \cup]5, +\infty[$.

• Exécuter la commande sqrt(3)∧2==3. Le résultat est surprenant, n'est-ce pas ? Pour comprendre d'où provient l'erreur, étudions l'écart relatif entre sqrt(3)∧2 et 3 et comparons la avec la précision ε de Scilab (stockée dans la variable %eps).

```
-->abs(sqrt(3)^2-3)<%eps
-->abs(sqrt(3)^2-3)<3*%eps
```

V Chaines de caractères

• Taper clear dans la console et exécuter.

• Exécuter les commandes suivantes :

```
-->Bonjour
-->"Bonjour"
-->'Bonjour'
-->x="Bonjour"; x
-->y="Je m'appelle Bond, James Bond."
-->y="Je m''appelle Bond, James Bond."
-->x+y
-->z=x+". "+y
```

• Exécuter les commandes suivantes :

```
-->a=5; b="5"; c=7; d="7";
-->a+d
-->a+c, b+d
-->typeof(a), typeof(b)
```

• Exécuter les commandes suivantes :

```
-->e=string(a); e+d
-->z + " " + "Mon nom de code est 00" + string(c)+ "."
-->"Une valeur approchée de pi est " + string(%pi) + '.'
```

A quoi sert la commande string ?

VI Vecteurs/listes

• Exécuter les commandes suivantes :

```
-->clear
-->v=[1,3,5,7,9,11,13,15]
-->-1:8
-->2:0.1:3
-->linspace(1,5,100)
-->ones(1:4)
-->zeros(1:5)
```

• Exécuter plusieurs fois la commande `rand(1:5)`.

• On peut modifier/supprimer/ajouter des coordonnées à un vecteur. Exécuter les commandes suivantes :

```
-->length(v), v(5), v($), v(10)
-->v(2)=-6; v(3)=[]; v
-->v(2:5), v(5:2)
-->v(4:5)=[0,-1]
-->v(1:2)=[]
-->v=[v,%pi]
```

• On peut réaliser des opérations usuelles sur les vecteurs ou leur appliquer des fonctions. Exécuter les commandes suivantes :

```
-->v=[1,3,5,7,9]; w=[-5,%pi,1,-3,2]; x=4;
-->v+w, v-w, v*x, v/x
-->v+[5,-1,2]
-->v*w
-->v.*w, v./w, v.^w
-->log(v), exp(v), sqrt(v)
-->sin(w), cos(w), tan(w), abs(w), floor(w)
```

• Pour l'instant, nous n'avons rencontré que des vecteurs lignes. Mais on peut également manipuler des vecteurs colonnes (et plus généralement des matrices). Exécuter les commandes suivantes :

```
-->u=[-1,0,3,5]; u'
-->u+u'
```

Nous y reviendrons...

• Enfin, on peut appliquer des opérateurs de comparaison avec des vecteurs de même taille et utiliser des vecteurs de booléens. Exécuter les commandes suivantes :

```
-->v=[1,2,3,4]; w=[-2,0,sqrt(10),%pi]; x=[1,-%e,7,5];
-->(v>w)&(v<=x)
-->u=[%t,%t,%f,%t]; or(u), and(u)
```

VII Polynômes

• Exécuter les commandes suivantes :

```
-->clear
-->P=poly([1,-3,0,-5,2],'X','c')
-->typeof(P)
-->Q=poly([1,-3,2],'X')
-->Y=poly(0,"Y"),
-->Z=1+2*Y+Y^2
```

Expliquer le lien entre le polynôme `Q` et le vecteur `[1,-3,2]`.

• Implémenter le polynôme $R = 5X^6 + 3X^3 + 2X - 1$ en Scilab ?

• Exécuter les commandes suivantes :

```
-->P+3*Q, P*Q, Q^2
-->horner(Z,3), horner(Z,[3,-1])
```

Que fait la commande `horner` ?

• Exécuter les commandes suivantes :

```
-->X=poly(0,'X');
-->degree(X^4-X^2+2)
-->P=(1+2*X+X^2)^2; coeff(P)
-->roots(P)
-->Q=poly([%i,2,%i,-1],'Y'); roots(Q)
```

Que font les commandes `coeff` et `root` ?

Feuille de TP n° 2

Premiers programmes en Scilab

Ce TP accompagne le chapitre 2 (Informatique et Algorithmique) : **Programmation en Scilab**.

I L'éditeur de texte SciNotes

• Cliquer sur l'icône  . L'éditeur de texte SciNotes s'ouvre dans une nouvelle fenêtre.

• A l'aide de la bande bleue (ou noire), insérer SciNotes dans l'environnement Scilab à droite de la console comme sur la figure de la page 4.

• Lorsqu'on clique sur la sous-fenêtre SciNotes, plusieurs icônes apparaissent en haut à gauche de l'environnement de travail. Cliquer sur l'icône  afin d'ouvrir un nouveau script SciNotes.

• Dans le script que l'on vient d'ouvrir, écrire

```
u0=7; q=-1/2; n=25;
u=u0*q^n
```

En cliquant sur le menu *Ficher* (en haut à gauche) puis *Enregistrer sous*, sauvegarder le script sous le nom `script1.sce` dans le dossier `..\ECS1B_TPInfo\TP2\`.

• On va maintenant exécuter ce script dans la console. Cliquer sur *Exécuter* (en haut à gauche) puis *..Exécuter sans écho*. Un raccourci est Ctrl + Maj + E . Que se passe-t-il dans la console ? Exécuter la commande `u` dans la console.

• Cliquer sur *Exécuter* puis *..Exécuter avec écho*. Quelle est la différence avec l'exécution sans écho ?

• Observer que les variables `u0`, `q`, `n` et `u` ont été affectées.

• Modifier la première ligne du script comme cela : `u0=-5; q=2; n=20;` puis exécuter le script sans écho.

• Remarquer la présence d'une petite astérisque dans le titre de l'onglet correspondant au script. Elle signifie que le script n'a pas été enregistré et donc que les modifications n'ont pas été prises en compte. Il faut toujours enregistrer un script (avec le raccourci Ctrl + S par exemple) quand on l'a modifié avant de l'exécuter.

• Lorsqu'on aura créé de nombreux scripts et qu'on voudra les consulter dans plusieurs mois, il ne sera pas évident de se rappeler ce leurs utilités et subtilités. Voilà pourquoi il faut prendre dès maintenant l'habitude d'ajouter des commentaires aux scripts. Le symbole // marque le début d'un commentaire : ce qui suit ce symbole ne sera pas exécuté. Modifier le script comme cela :

```
u0=-5; q=2; n=20;
//Ce programme calcule le n-ième terme de la suite géométrique
//de raison q et de terme initiale u0
u=u0*q^n
```

Enregistrer le script et exécuter-le sans écho puis avec écho.

• Cliquer sur l'icône  afin d'ouvrir un nouveau script. Constater qu'un nouvel onglet apparait à côté de l'onglet précédent. Cela permet de passer d'un script à l'autre très rapidement.

II Entrainement : interaction avec l'utilisateur

Le script que l'on vient de créer n'est pas très pratique : si on veut modifier les valeurs de `u0`, `q` et `n`, il faut systématiquement modifier le script, l'enregistrer et l'exécuter à nouveau. Par ailleurs le programme implémenté dans le script renvoie le résultat stocké dans une variable `u` mais il serait plus clair qu'une phrase soit affichée pour indiquer ce que représente le résultat (ici le $n^{\text{ième}}$ terme de la suite géométrique de raison `q` et de terme initial `u0`).

Nous allons voir comment écrire un programme qui demande à l'utilisateur de préciser le contenu des variables d'entrée et qui affiche une phrase contenant éventuellement les variables de sortie.

• Dans la console, exécuter les commandes suivantes une par une :

```
-->n=input("Entrer un nombre entier : ")
-->nom=input("Comment vous appelez-vous ? ","string")
```

Que fait la commande `input` ? A quoi sert l'option `string` ?

• Créer un nouveau script appelé `suitegeom.sce` avec le contenu suivant :

```
u0=input('Entrer le terme initial de la suite géométrique : ')
q=input('Entrer la raison de la suite géométrique : ')
n=input('Entrer un rang : ')
//Ce programme calcule le n-ième terme de la suite géométrique
//de raison q et de terme initiale u0
u=u0*q^n
```

Enregistrer le script et exécuter-le sans écho (tester-le avec différentes valeurs).

• Dans la console, exécuter les commandes suivantes une par une :

```
-->n=7; s=n*(n+1)/2; disp("La somme des entiers de 1 à ", n, " est ", s)
-->disp(s," est ", n,"La somme des entiers de 1 à ")
```

Que fait la commande `disp` ? Qu'en est-il de l'ordre des arguments ? Exécuter la commande suivante :

```
-->disp('La somme des entiers de 1 à ' + string(n) + ' est ' + string(s) + '.')
```

• Ajouter la commande suivante à la fin du script `suitegeom.sce` :

```
disp('La valeur du '+string(n)+'-ième terme de la suite géométrique de raison '
        +string(q)+' et de terme initial '+string(u0)+' est '+string(u)+'.')
```

Enregistrer le script et exécuter-le sans écho (tester-le avec différentes valeurs). Le résultat est plutôt satisfaisant n'est-ce pas ?

III Exercices

Exercice 1. (★) Écrire un programme qui demande à l'utilisateur deux entiers p et n (avec $p \leqslant n$) et qui affiche trois phrases indiquant les valeurs de $\sum_{k=p}^{n} k$, $\sum_{k=p}^{n} k^2$ et $\sum_{k=p}^{n} k^3$.
On utilisera les formules, vues en cours, donnant une expression de ces sommes en fonction de n et p.

Exercice 2. (★) Écrire un programme qui demande à l'utilisateur deux entiers p et n (avec $p \leqslant n$) et deux réels x et q et qui affiche une phrase indiquant la valeur de $\sum_{k=p}^{n} u_k$, où $(u_k)_{k\in\mathbb{N}}$ est la suite géométrique de raison q et de terme initial x.

Exercice 3. (★) Écrire un programme qui demande à l'utilisateur d'entrer trois réels a, b et c, qui calcule le discriminant du polynôme aX^2+bX+c et qui affiche une phrase du type « *Le polynôme aX^2+bX+c admet pour discriminant...* »

Exercice 4. (★) Écrire un programme qui demande à l'utilisateur d'entrer deux vecteurs de taille 2 représentant les coordonnées de points A et B du plan muni d'un repère orthonormé (un vecteur par point), et qui affiche

- une phrase indiquant les coordonnées du milieu I de $[AB]$,
- une phrase indiquant la longueur du segment $[AB]$.

Exercice 5. (★) Écrire un programme qui demande à l'utilisateur une durée T exprimé en secondes et qui le transforme en heures/minutes/secondes.
On utilisera la commande `floor`.

Feuille de TP n° 3

Structures conditionnelles

Ce TP accompagne le chapitre 2 (Informatique et Algorithmique) : **Programmation en Scilab**.

Exercice 1. (★) Si `n` est une variable contenant un nombre réel, que renvoie la commande `n==4*floor(n/4)` ? Construire alors un programme qui demande à l'utilisateur de rentrer une année et qui affiche un message précisant si l'année est bissextile ou non.

Exercice 2. (★) Ecrire un programme qui demande à l'utilisateur un chiffre entre 0 et 9 et qui renvoie ce chiffre écrit en lettres. Il pourra renvoyer un message d'erreur si la réponse de l'utilisateur n'est pas un entier entre 0 et 9.

Exercice 3. (★) Écrire un programme qui demande à l'utilisateur de rentrer trois réels et qui renvoie un vecteur composés de ces trois réels rangés dans l'ordre croissant.

Exercice 4. (★) Écrire un programme qui demande à l'utilisateur un réel x et qui calcule (une valeur approchée de) $f(x) = \sqrt[4]{1 - \dfrac{5}{3+x}}$ lorsque cette expression à un sens et qui affiche un message d'erreur sinon.

Exercice 5. (★)

1) Écrire un programme qui demande à l'utilisateur d'entrer deux réels b, c et qui résout l'équation $bx + c = 0$ d'inconnue $x \in \mathbb{R}$ (le programme devra proposer une sortie adéquate y compris dans le cas où $b = 0$).
2) Écrire un programme qui demande à l'utilisateur d'entrer trois réels a, b, c et qui résout l'équation $ax^2 + bx + c = 0$ d'inconnue $x \in \mathbb{R}$.
3) Écrire une variante du programme précédent qui résout l'équation dans $\mathbb{C}$.

Exercice 6. (★) Écrire un programme qui demande à l'utilisateur trois réels a, b et c et qui donne le domaine de définition de la fonction $x \longmapsto \ln(ax^2 + bx + c)$.

Exercice 7. (★★) Écrire un programme qui demande à l'utilisateur d'entrer quatre réels a, $b \neq 0$, x, y et qui renvoie la valeur du $n^{\text{ième}}$ terme de la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 = x$, $u_1 = y$ et pour tout $n \in \mathbb{N}$, $u_{n+2} = au_{n+1} + bu_n$.
On utilisera les formules vues en cours donnant, pour tout $n \in \mathbb{N}$, une expression de u_n en fonction de n selon le discriminant de l'équation caractéristique associée à la suite.

Exercice 8. (★★) Écrire un programme qui simule la saisie du code confidentiel d'une carte bancaire : l'utilisateur a trois essais pour taper le bon code et ensuite la carte est bloquée (on peut prendre par exemple 2017 comme code confidentiel).

Feuille de TP n° 4

Structures répétitives : boucle `for`

Ce TP accompagne le chapitre 2 (Informatique et Algorithmique) : **Programmation en Scilab**.

I Entrainement

En général, la construction d'une boucle `for` au sein d'un programme se base sur quatre étapes :

- On identifie les variables d'entrées (qui proviennent du début du programme ou qui sont fournies par l'utilisateur via la commande `input`).
- On définit des variables qui vont être modifiées lors de la boucle `for`. C'est l'étape d'initialisation.
- On écrit la boucle en tant que telle (`for`$\cdots$`end`). Elle utilise les variables d'entrées et, à chaque étape de la boucle, les variables définies à l'étape d'initialisation sont mises à jour.
- On renvoie les variables de sorties (affichées éventuellement dans une phrase via la commande `disp`) qui sont en général des fonctions des quantités calculées pendant la boucle.

- Étudier en détail l'exemple du chapitre 2 (calcul de la somme $\sum_{i=1}^{n} i^p$, n et p étant fournis par l'utilisateur).

- Écrire un script contentant le programme suivant :

```
n=input('Entrer un entier strictement positif : ')
q=input('Entrer un réel : ')
u=1; s=1;
for  i=1:n
     u=u*q; s=s+u;
end
disp('La somme des ' + string(q) + '^k pour k allant de 0 à ' + string(n)
          + ' est ' + string(s) + '.')
```

Analyser ce programme en détail : identifier les quatre étapes recensées en début de paragraphe et ajouter des commentaires au programme (on pourra s'aider d'un tableau comme dans le chapitre 2).
Enregistrer-le sous le nom `SommeGeom.sce` et exécuter-le sans écho dans la console (tester-le avec plusieurs valeurs de `n` et `q`) et comparer avec la formule du cours.

- Écrire un script contentant le programme suivant :

```
for  j=1:10
     t=0;
     for  i=1:10
          t=t+j;
          disp(string(i)+'x'+string(j)+'='+string(t));
     end
     disp('########')
end
```

Après avoir identifié ce que fait ce programme, enregistrer-le avec nom approprié et exécuter-le sans écho dans la console. Ajouter des commentaires au code et modifier le programme pour qu'il affiche une phrase précisant ce qu'il fait. Enregistrer et exécuter-le sans écho à nouveau.

II Quelques programmes à savoir faire absolument

... c'est écrit dans le programme officiel d'ECS1 !

1) Calcul de $n!$

Écrire un programme qui demande à l'utilisateur un entier naturel n et qui calcule $n!$ en utilisant une structure répétitive.

2) Calcul de $\binom{n}{p}$

Écrire un programme qui demande à l'utilisateur deux entiers naturels n et p (avec $p \leqslant n$) et qui calcule $\binom{n}{p}$ en utilisant une structure répétitive.

III Exercices

Exercice 1. (★) Modifier le programme précédent pour qu'il traite le cas où $p > n$ et pour qu'il renvoie un message d'erreur si n ou p ne sont pas des entiers naturels.

Exercice 2. (★) Écrire un programme qui demande un entier naturel n et qui calcule $\displaystyle\sum_{1\leqslant i<j\leqslant n} \frac{1}{i+j}$.

Exercice 3. (★) Pour tout $n \in \mathbb{N}$, posons $S_n = \displaystyle\sum_{1\leqslant i\leqslant k\leqslant n} (k^2 - i + 1)$

1) Calculer S_n en fonction de $n \in \mathbb{N}$.
2) Écrire un programme qui demande à l'utilisateur un entier naturel n, calcule S_n de deux façons différentes (avec la formule de la question précédente et avec une boucle `for`) et affiche les deux valeurs calculées.

Exercice 4. (★) Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par $u_0 \in \mathbb{R}$ et, pour tout $n \in \mathbb{N}$, $u_{n+1} = \cos\left(e^{u_n}\right) + \dfrac{u_n}{2}$. Écrire un programme qui demande à l'utilisateur un entier naturel n et un réel u_0 et qui calcule u_n.

Exercice 5. (★★) Soient $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$ deux suites définies par $u_0 = -2$, $v_0 = 1$ et

$$\forall n \in \mathbb{N}, \qquad u_{n+1} = 5u_n + 4v_n \qquad \text{et} \qquad v_{n+1} = 4u_n + 5v_n.$$

Ecrire un programme qui prend en entrée un entier naturel n et qui calcule u_n et v_n. Comparer avec le résultat théorique de l'exercice 3 de la feuille d'exercice n° 5.

Exercice 6. (★★) Pour tous $x \in \mathbb{R}$ et $n \in \mathbb{N}$, notons $S_n(x) = \sum_{k=0}^{n} \frac{x^k}{k!}$.

1) Écrire un programme qui demande à l'utilisateur un entier naturel n et un réel x et qui calcule $S_n(x)$ (sans utiliser le symbole $\wedge$).
2) Comparer $S_n(x)$ et $\exp(x)$ pour plusieurs valeurs de $x \in \mathbb{R}$ et $n \in \mathbb{N}$. Commenter.

Exercice 7 – conjecture de Syracuse. (★★) La suite de Syracuse est la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 \in \mathbb{N}$ et, pour tout $n \in \mathbb{N}$,

$$u_{n+1} = \begin{cases} \dfrac{u_n}{2} & \text{si} \quad u_n \text{ est pair,} \\ 3u_n + 1 & \text{sinon.} \end{cases}$$

1) Caractériser la suite de Syracuse quand $u_0 = 1$.
2) Écrire un programme qui demande à l'utilisateur d'entrer deux entiers naturels n et u_0 et qui affiche les $n+1$ premiers termes $u_0, \dots, u_n$ de la suite de Syracuse (on pourra par exemple retourner un vecteur contenant ces $n+1$ valeurs).
3) Tester le programme pour différentes valeurs. Que peut-on conjecturer [1] ?

Exercice 8. (★★) Pour tout $n \in \mathbb{N}^*$, posons $S_n = \sum_{k=1}^{n} \frac{(-1)^{k+1}}{k}$. Nous montrerons dans le chapitre *Dérivées successives et formules de Taylor* que $(S_n)_{n\in\mathbb{N}^*}$ converge vers $\ln(2)$ et que, pour tout $n \in \mathbb{N}^*$, $|S_n - \ln(2)| \leqslant \frac{1}{n+1}$.

1) Déterminer un rang n tel que, $|S_n - \ln(2)| \leqslant 0,01$.
2) Écrire un programme qui détermine une valeur approchée de $\ln(2)$ à 10^{-2} près.

Exercice 9 – Algorithme de Hörner. (★★★) Soient $x \in \mathbb{K}$ et $P = \sum_{k=0}^{n} a_k X^k \in \mathbb{K}[X]$. On souhaite demander à un ordinateur de calculer $P(x) = \sum_{k=0}^{p} a_k x^k$.

1) Implémenter en `Scilab` un algorithme naturel permettant de calculer $P(x)$.
2) Cette méthode naïve est assez lente : elle nécessite $1+2+\cdots+n = \frac{n(n+1)}{2}$ multiplications et n additions. L'algorithme de Hörner consiste en l'écriture de $P(x)$ sous la forme

   $$P(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + \cdots + x(a_{n-1} + xa_n)))).$$

   Implémenter en `Scilab` l'algorithme de Hörner. Combien nécessite-il d'additions et de multiplications ? Est-il plus rapide ?

Exercice 10. (★★★) Le nombre 4150 possède une propriété remarquable : il est égal à la somme des puissances cinquièmes de ses chiffres. En effet

$$4150 = 4^5 + 1^5 + 5^5 + 0^5.$$

1) En raisonnant par l'absurde, montrer que si un nombre vérifiant cette propriété a au plus 6 chiffres.
2) En déduire que seul un nombre fini d'entiers vérifie cette propriété.
3) Écrire un programme Scilab qui renvoie la liste des nombres satisfaisant cette propriété.

1. La conjecture de Syracuse n'est pas démontrée à ce jour... mais elle a été vérifiée par ordinateur au moins jusque l'entier $N < 1,25 \times 2^{62}$.

Feuille de TP n° 5

Structures répétitives : boucle `while`

Ce TP accompagne le chapitre 2 (Informatique et Algorithmique) : **Programmation en Scilab**.

I Entrainement

Une boucle `while` permet de répéter une séquence d'instructions tant qu'une certaine condition, appelée test d'arrêt, est vraie. Elle s'arrête dès que cette condition est fausse.

- Sans l'écrire dans un script, dire ce que fait le programme suivant :

```
n=input('Entrer un entier naturel : ')
k=0; s=1;
while  k<n
       k=k+1; s=s+1/(k+1);
end
disp(s)
```

Pour nous aider à le comprendre, complétons le tableau suivant : on fournit `n=4`, alors

valeurs en début de boucle		`k<n`	valeurs en fin de boucle	
`k`	`s`		`k`	`s`
0	1	vrai		

Remplacer la boucle `while` par une boucle `for` (effectuer cette modification sur cette feuille à côté du script initial).

En pratique, on n'utilisera une boucle `while` que lorsqu'elle est véritablement nécessaire : quand on ne connaît pas à l'avance le nombre d'itérations à effectuer. C'est le cas dans l'exemple suivant.

- Sans l'écrire dans un script, dire ce que fait le programme suivant :

```
p=input('Entrer un entier naturel p : ')
A=input('Entrer un réel A : ')
n=0; s=0;
while  s<A
       n=n+1; s=s+n^p;
end
disp(n)
```

Pour nous aider à le comprendre, complétons le tableau suivant : on fournit `p=5` et `A=1000`, alors

valeurs en début de boucle		`n<A`	valeurs en fin de boucle	
`n`	`s`		`n`	`s`
0	0	vrai		

• Étudier en détail l'exemple du chapitre 2 (calcul du plus petit rang $n \in \mathbb{N}^*$ pour lequel $\sum_{k=1}^{n} \frac{1}{k} \geqslant A$, A étant fourni par l'utilisateur).

II Exercices

Exercice 1. (★) Pour $n \in \mathbb{N}^*$, posons $S_n = \sum_{k=1}^{n} \frac{1}{k^2}$. On admet que la suite $(S_n)_{n\in\mathbb{N}^*}$ converge vers $\frac{\pi^2}{6}$.

1) Écrire un programme qui calcule une approximation de $\frac{\pi^2}{6}$ à l'aide de la valeur de S_{1000}.
2) Écrire un programme qui demande à l'utilisateur un réel $\varepsilon > 0$ et qui détermine le premier rang n tel que $\left|S_n - \frac{\pi^2}{6}\right| \leqslant \varepsilon$.

Exercice 2. (★) Soit $(u_n)_{n\in\mathbb{N}}$ la suite telle que $u_0 \in \mathbb{R}_+^*$ et, pour tout $n \in \mathbb{N}$, $u_{n+1} = 1 + \frac{2}{u_n}$.

Nous avons vu dans le chapitre 6 que cette suite converge vers 2. Écrire un programme qui demande à l'utilisateur deux réels strictement positifs u_0 et ε et qui détermine le premier rang n pour lequel $|u_n - 2| \leqslant \varepsilon$.

Exercice 3. (★★) Considérons $(u_n)_{n\in\mathbb{N}}$ la suite telle que $u_0 = 1$ et, pour tout $n \in \mathbb{N}$, $u_{n+1} = 1 - e^{u_n}$.

1) Étudions la convergence de la suite $(u_n)_{n\in\mathbb{N}}$.
   a) Montrer que, pour tout $n \in \mathbb{N}$, $u_n \in [-2, 1]$.
   b) Déterminer les points fixes de $f \circ f$ où $f : x \in \mathbb{R} \mapsto 1 - e^x$.
   c) Montrer que les suites $(u_{2n})_{n\in\mathbb{N}}$ et $(u_{2n+1})_{n\in\mathbb{N}}$ sont monotones.
   d) En déduire que $(u_n)_{n\in\mathbb{N}}$ converge vers 0.
2) Écrire un programme qui calcule le premier rang n pour lequel $|u_n| \leqslant 0,01$.

Exercice 4 – conjecture de Syracuse (suite). (★) La suite de Syracuse est la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 \in \mathbb{N}$ et, pour tout $n \in \mathbb{N}$,

$$u_{n+1} = \begin{cases} \dfrac{u_n}{2} & \text{si} \quad u_n \text{ est pair,} \\ 3u_n + 1 & \text{sinon.} \end{cases}$$

La conjecture de Syracuse affirme que, *pour tout* $x \in \mathbb{R}$, *il existe* $n_x \in \mathbb{N}$ *tel que* $u_{n_x} = 1$.

Écrire un programme qui prend en entrée un entier u_0 et calcule le plus petit rang n tel que $u_n = 1$.

Exercice 5. (★★) Écrire un programme qui choisit au hasard un entier entre 1 et 100, qui demande à l'utilisateur de deviner le nombre choisi et qui répond *plus* ou *moins* selon la valeur proposée par l'utilisateur. Le programme continue d'interroger l'utilisateur jusqu'à ce qu'il devine la bonne réponse.

Indication : La commande `rand()` *renvoie un nombre aléatoire tiré au hasard dans l'intervalle* $]0, 1[$. *Si* $n \in \mathbb{N}^*$, *alors la commande* `1+floor(n*rand())` *renvoie donc un entier tiré au hasard entre* 1 *et* n. *On y reviendra plus en détail lors des TP de Probabilités...*

Feuille de TP n° 6

Fonctions en Scilab

Ce TP accompagne le chapitre 3 (Informatique et Algorithmique) : **Fonctions en Scilab**.

I Entrainement

Nous avons vu que Scilab dispose de nombreuses commandes et fonctions prédéfinies. Nous allons à présent voir comment créer nos propres fonctions.

• On peut coder des fonctions de la variable réelle à valeurs réelles. Par exemple la fonction $f : x \longmapsto \dfrac{\sin(x)}{1+x^2}$, s'implémente comme cela en Scilab :

```
function y=fct1(x)
    y=sin(x)/(1+x^2);
endfunction
```

Recopier ce code dans un script Scilab. En écrivant `y=fct1(x)`, nous avons choisi de :

- stocker dans la variable `x` l'argument d'entrée de la fonction.
- stocker dans la variable `y` l'argument de sortie de la fonction
- appeler `fct1` la fonction.

Il faut absolument sauvegarder ce script sous le nom `fct1.sci` (`fct1` car c'est le nom que l'on a donné à la fonction et `.sci` car c'est l'extension pour les fonctions Scilab).

Exécuter le script sans écho dans la console. Désormais `fct1` s'ajoute à la liste des fonctions Scilab. Tester les commandes suivantes :

```
-->clear
-->fct1(0), fct1(%pi), fct1(1)
-->x
-->y
-->y=fct1(-1); y
```

• Constater que les variables `x` et `y` ne sont pas affectées. C'est normal ce sont des variables muettes.

• On a vu dans le TP1 que les fonctions usuelles (`exp, log, sqrt, ans, floor, sin, cos, tan`) peuvent prendre en entrée des vecteurs et renvoient un vecteur de même taille dont chaque coordonnée est évaluée par la fonction. Est-ce que cela fonctionne toujours avec la fonction que nous avons définie ?

Exécuter la commande `fct1([1,2,3])` dans la console.

Elle renvoie un message d'erreur. C'est normal car notre fonction n'est pas compatible avec les opérations algébriques sur les vecteurs. Faites les modifications nécessaires (c'est-à-dire remplacer `*` par `.*`, `/` par `./` et $\wedge$ par `.^`) et tester-la (après l'avoir enregistrée et exécutée à nouveau) avec la commande `fct1([1,2,3])`.

 Cette méthode ne fonctionne pas si la fonction contient des structures conditionnelles, comme on le verra dans les exercices.

• On peut aussi créer des fonctions qui prennent plusieurs arguments d'entrée et/ou plusieurs arguments de sortie. Recopier la fonction ci-contre qui prend en arguments deux entiers naturels et qui renvoie le quotient et le reste de la division euclidienne de a par b.

```
function [q,r]=DivEucl(a,b)
    q=0;
    while  (q+1)*b<a
           q=q+1;
    end
    r=a-q*b;
endfunction
```

Tester les commandes suivantes :

```
-->DivEucl(60,7)
-->v=DivEucl(60,7); v
-->[q,r]=DivEucl(60,7)
```

II Exercices

Exercice 1. (★) Reprendre le TP n° 4 et écrire :

1) une fonction `facto.sci` qui prend en entrée un entier naturel n et qui calcule $n!$.
2) une fonction `binom.sci` qui prend en entrée deux entiers naturels n et p (pas forcément tels que $p \leqslant n$) et qui calcule $\binom{n}{p}$.

Exercice 2. (★) Écrire une fonction en Scilab qui prend en entrée un entier n et quatre réels a, b, x, y et qui renvoie la valeur du $n^{\text{ième}}$ terme de la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 = x$, $u_1 = y$ et pour tout $n \in \mathbb{N}$, $u_{n+2} = au_{n+1} + bu_n$. *On utilisera la formule de récurrence et une boucle* `for`.

Tester cette fonction avec $(n, a, b, x, y) = (10, 6, 7, -2, -1)$ et comparer avec la valeur théorique donnée par une formule du cours.

Exercice 3. (★★) Implémenter en Scilab [1] :

1) une fonction `somme.sci` qui prend en entrée un vecteur et qui calcule la somme de ses coordonnées.
2) une fonction `minmax.sci` qui prend en entrée un vecteur composé de nombres réels et qui renvoie en sortie la plus grande et la plus petite de ses coordonnées.

Exercice 4. (★★)

1) Implémenter en Scilab les fonctions suivantes :

$$f : x \longmapsto \frac{e^x - e^{-x}}{e^x + e^{-x}}, \qquad g : x \longmapsto \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right), \qquad h : x \longmapsto \begin{cases} 1 - |x| & \text{si} \quad x \in [-1,1] \\ 0 & \text{si} \quad x \notin [-1,1] \end{cases},$$

$$\phi : x \longmapsto \begin{cases} \dfrac{\sin(x)}{x} & \text{si} \quad x \neq 0 \\ 1 & \text{si} \quad x = 0 \end{cases} \qquad \text{et} \qquad \psi : x \longmapsto \begin{cases} \sin(x)\ln(-x) & \text{si} \quad x < 0 \\ 0 & \text{si} \quad x = 0 \\ x^{-3/2}(1 - \cos(2x)) & \text{si} \quad x > 0 \end{cases}.$$

Pour la fonction h, on pourra s'aider de la fonction `max` *pour éviter le recours à une structure conditionnelle.*

2) Testez ces fonctions avec en entrée le vecteur `X=[-1,0,1]` (si ce n'est pas déjà fait, il faut les rendre compatibles avec un traitement vectoriel).
3) Constater un problème avec les fonctions ϕ et ψ. Après avoir bien relu le chapitre 3, régler ce problème.

Exercice 5. (★★) Implémenter en Scilab la fonction f qui est 2-périodique sur $\mathbb{R}$ et qui vérifie

$$\forall x \in [-1,1], \qquad f(x) = x\sqrt{1 - x^2}.$$

Exercice 6. (★★★) Pour tout $n \in \mathbb{N}$, notons S_n la fonction définie par

$$\forall x \in \mathbb{R}, \qquad S_n(x) = \sum_{k=0}^{n} \frac{(-1)^k x^{2k}}{(2k)!}.$$

1) Écrire une fonction en Scilab nommée *sommecos.sci* qui prend en entrée $(x, n) \in \mathbb{R} \times \mathbb{N}$ et qui renvoie la valeur de $S_n(x)$ (sans utiliser le symbole $\wedge$).
2) Comparer les fonctions $\cos$ et S_n sur $[-k\pi, k\pi]$ pour différentes valeurs des entiers k et n.

1. Scilab dispose déjà de fonctions permettant de faire cela (il s'agit de `sum`, `max` et `min`)... mais on fait semblant de ne pas le savoir.

Feuille de TP n° 7

Représentation graphique avec Scilab

Ce TP accompagne le chapitre 4 (Informatique et Algorithmique) : **Représentation graphique en Scilab**.

I Entrainement

1) `plot` VS `plot2d`

• Exécuter les commandes suivantes dans la console :

```
-->X=[0,1,4,11,10,15,5,4,2,1.5,1,0.5,0];
-->Y=[0,2,4,4,7,10,10,8,7.5,7.5,9.5,8.5,8.5];
-->plot(X,Y)
-->plot(Y,X)
```

Par défaut, les tracés successifs se superposent. Il faut penser à utiliser la commander `clf();` pour effacer le contenu de la fenêtre graphique avant de tracer une nouvelle figure.

• Exécuter les commandes suivantes dans la console :

```
-->clf(); plot(X,Y,'r')
-->clf(); plot(X,Y,'g')
-->clf(); plot(X,Y,'x')
-->clf(); plot(X,Y,'o')
-->clf(); plot(X,Y,'--')
```

```
-->clf(); plot(X,Y,':')
-->clf(); plot(X,Y,'c*')
-->clf(); plot(X,Y,'rx')
-->clf(); plot(X,Y,'go:')
```

• Exécuter les commandes suivantes dans la console :

```
-->clf(); plot(X,Y,-X,Y)
-->clf(); plot(X,Y,'k',-X,Y,'k')
-->clf(); plot(X,Y,'r-',-X,Y,'g*')
```

• Lorsque l'on désire tracer plusieurs courbes ayant le même vecteur d'abscisses `X`, on peut n'écrire qu'une seul fois `X` et regrouper les vecteurs ordonnées. Exécuter les commandes suivantes :

```
-->X=[5,9,11,13,14,17,20,21];
-->EM=[23.5,24,23,23.5,22,24,25,24];
-->MLP=[23.5,24,23,22.5,22,23,20,21.8];
-->FF=[19,17,19,20,19,19.5,19,19.5];
-->JLM=[17,18,17,18.5,20,18,19,19.3];
-->clf(); plot(X,[EM;MLP;FF;JLM])
```

Il s'agit de courbes de sondages *(elles correspondent aux intentions de votes pour les quatre candidats de tête à l'élection présidentielle de 2017 relevées à 8 dates d'avril 2017)*. Les couleurs choisies par Scilab ne correspondent pas vraiment aux couleurs usuellement attribuées aux candidats. Hélas avec cette syntaxe on ne peut pas changer les couleurs. Plusieurs options s'offrent à nous :

- L'ancienne syntaxe :

```
-->clf(); plot(X,EM,'c',X,MLP,'k',X,FF,'b',X,JLM,'r')
```

- L'utilisation de la fonction `plot2d` qui fonctionne essentiellement comme `plot` mais avec des variantes dans la syntaxe, notamment pour la gestion des options (les couleurs sont symbolisées par des numéros) :

```
-->clf(); plot2d(X',[EM',MLP',FF',JLM'],style=[4,1,2,5])
```

Les apostrophes après chaque vecteur servent à les transposer (c'est-à-dire les transformer en vecteurs lignes).

On liera attentivement le chapitre 4 pour d'autres détails sur les fonctions `plot` et `plot2d` et sur leurs différences.

• Un intérêt de `plot2d` est que l'on peut modifier la zone graphique et insérer des légendes directement dans la syntaxe de la fonction. Exécuter les commandes suivantes :

```
-->legende="Macron@Le Pen@Fillon@Mélenchon";
-->clf();
-->plot2d(X',[EM',MLP',FF',JLM'],style=[4,1,2,5],rect=[5,16,21,25],leg=legende)
```

• Exécuter les commandes suivantes :

```
-->Z=[0,3,-2,-5,-1,2,4,7]
-->clf(); plot2d(X,Z);
-->clf(); plot2d2(X,Z);
-->clf(); plot2d3(X,Z);
```

2) Représentation graphique de suites

Soit $N \in \mathbb{N}$. Représenter graphiquement les N premiers termes d'une suite $(u_n)_{n\in\mathbb{N}}$ consiste à relier successivement les points $(0, u_0)$, $(1, u_1)$, $(2, u_2)$, ..., (N, u_N).

Exécuter les commandes suivantes :

```
-->N=20; u=1;
-->U=[u]; for k=1:N; u=1+2/u; U=[U,u]; end;
-->plot(0:N,U)
```

Que peut-on conjecturer ?

• Exécuter les commandes suivantes :

```
-->N=100; s=0;
-->S=[]; for k=1:N; s=s+(-1)^(k+1)/k; S=[S,s]; end;
-->plot(1:N,S)
-->plot([1,N],[log(2),log(2)],'r')
```

Que peut-on conjecturer ?

3) Tracé de courbes représentatives de fonctions

Tracer la courbe représentative d'une fonction sur un intervalle, revient à tracer une infinité de points. Or il n'est évidemment pas possible pour un ordinateur de réaliser une infinité d'instructions. Pour tracer une courbe avec Scilab, on va représenter un nombre fini de points et de les relier par une ligne continue.

• Exécuter sans écho dans la console le script `fct1.sci` du TP précédent. Exécuter ensuite les commandes suivantes :

```
-->x=linspace(-15,15,10);
-->clf(); plot(x,fct1(x),'r')
```

• Il faut bien sûr un plus grand nombre de points pour que l'approximation de la courbe par une ligne brisée donne l'illusion de courbe.

```
-->x=linspace(-15,15,20); clf(); plot(x,fct1(x),'r')
-->x=linspace(-15,15,30); clf(); plot(x,fct1(x),'r')
-->x=linspace(-15,15,1000); clf(); plot(x,fct1(x),'r')
```

• Il y a d'autres syntaxes pour tracer la courbe représentative d'une fonction. Exécuter ensuite les commandes suivantes :

```
-->x=linspace(-15,15,1000);
-->clf(); plot(x,fct1,'r')
-->clf(); plot2d(x,fct1(x),5)
-->clf(); fplot2d(x,fct1,5)
```

4) Diagrammes à barres et histogrammes

• Exécuter les commandes suivantes :

```
-->x=1:5; y=[10,5,9,7,1];
-->clf(); bar(x,y,'r')
-->z=[6,3,3,5,2];
-->clf(); bar(x,[y',z'])
-->clf(); bar(x,[y',z'],'stacked')
```

• Exécuter plusieurs fois les commandes suivantes :

```
-->x=2*rand(1:50)-1;
-->clf(); histplot(5,x)
-->clf(); histplot(10,x)
-->clf(); histplot(10,x,normalized=%f)
-->clf(); histplot([-1,-0.5,0.5,0.75,1],x)
```

5) Personnalisation des fenêtres graphiques

• Exécuter les commandes suivantes :

```
-->clf(); x=linspace(-%pi,%pi,1000);
-->plot2d(x,[cos(x)',(1-x.^2/2)'])
```

Réduire la zone du graphique visible à l'intervalle $[-1,1]$ à l'aide la commande `square`. Ajouter ensuite un quadrillage, un titre, une légende et annoter les axes (à l'aide des fonctions `xgrid, title, legend, xlabel, ylabel` respectivement).

• Exécuter les commandes suivantes :

```
-->x=2*rand(1:50)-1;
-->scf();
-->subplot(2,3,1)
-->histplot(5,x)
-->subplot(2,3,3)
-->histplot(10,x)
-->subplot(2,3,5)
-->histplot(20,x)
```

Que fait `subplot` ? Que fait la commande `scf()` ?

II Exercices

Exercice 1 – transition avec un prochain TP. (★) Représenter le diagramme à barres traduisant le tableau ci-dessous :

k	2	3	4	5	6	7	8	9	10	11	12
p_k	$\frac{1}{36}$	$\frac{1}{18}$	$\frac{1}{12}$	$\frac{1}{9}$	$\frac{5}{36}$	$\frac{1}{6}$	$\frac{5}{36}$	$\frac{1}{9}$	$\frac{1}{12}$	$\frac{1}{18}$	$\frac{1}{36}$

Nous avons vu dans le chapitre *Variables aléatoires réelles finies* que $(p_k)_{2\leqslant k\leqslant 12}$ est la loi d'une variable aléatoire représentant la somme des chiffres lorsqu'on lance deux dés.

Exercice 2. (★) Pour $n\in\mathbb{N}^*$, posons $S_n=\displaystyle\sum_{k=1}^{n}\frac{1}{k^2}$.

1) Écrire un programme qui représente graphiquement les 20 premiers termes de la suite $(S_n)_{n\geqslant 1}$.

2) Modifier le programme afin qu'il superpose au tracé la fonction constante égale à $\dfrac{\pi^2}{6}$. Commenter.

Exercice 3. (★) Tracer la courbe représentative des cinq fonctions de l'exercice 3 du TP 6, sur des intervalles bien choisis et sur cinq graphiques différents.

Exercice 4. (★★) Pour tout $a > 0$, posons $\gamma_a : x \longmapsto \begin{cases} 0 & \text{si} \quad x < 0 \\ x^{a-1}e^{-x} & \text{si} \quad x > 0 \end{cases}$.

1) Implémenter une fonction en Scilab qui prend en entrée x et a et qui calcule $\gamma_a(x)$ (et qui soit compatible avec un traitement vectoriel pour la première coordonnée).
2) Écrire un programme qui représente, sur une même fenêtre graphique, les fonctions g_a sur l'intervalle $[-1, 5]$ pour $a \in \left\{\frac{1}{100}, \frac{1}{2}, 1, \frac{3}{2}, 2, \frac{5}{2}, 3, 100\right\}$.
   On répartira les tracés sur deux lignes et on utilisera une boucle `for`.

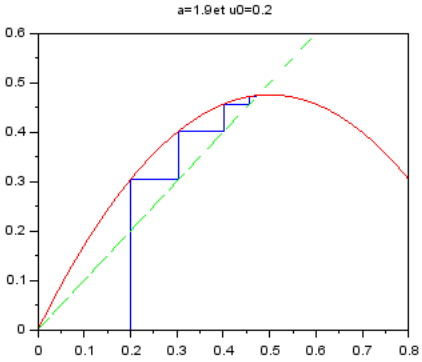
Exercice 5. (★★) Dans le TP6, nous avons implémenté en Scilab les fonctions $S_n : x \longmapsto \sum_{k=0}^{n} \frac{(-1)^k x^{2k}}{(2k)!}$, $n \in \mathbb{N}$, sous le nom `sommecos.sci`, et nous les avons comparées avec la fonction cosinus. Sur une même fenêtre graphique découpée en huit zones, représenter

- sur la première ligne : les fonctions $\cos$ et S_n pour $n \in \{5, 6, 10, 20\}$, sur l'intervalle $[-2\pi, 2\pi]$.
- sur la deuxième ligne : les fonctions $\cos$ et S_n pour $n \in \{20, 30, 40, 50\}$, sur l'intervalle $[-10\pi, 10\pi]$.

Exercice 6. (★★★) Pour tout $a \in \mathbb{R}_+^*$, considérons la fonction $f_a : x \longmapsto ax(1-x)$ sur $[0, 1]$.

1) Implémenter une fonction en Scilab qui prend en entrée x et a et qui calcule $f_a(x)$ (et qui soit compatible avec un traitement vectoriel pour la première coordonnée).
2) Le graphique ci-contre représente la trajectoire (en bleu) d'une suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 \in [0, 1]$ et, pour tout $n \in \mathbb{N}$, $u_{n+1} = f_a(u_n)$. La fonction f_a est représentée en rouge et la droite $y = x$ en pointillés verts.

   La courbe bleue est la ligne brisée qui joint les points $(0, u_0)$, (u_0, u_1), (u_1, u_1), (u_1, u_2), ...



   a) Écrire une fonction qui prend en entrée $a \in \mathbb{R}_+^*$, $u_0 \in [0, 1]$ et $n \in \mathbb{N}^*$ et qui reproduit la figure ci-contre, la trajectoire bleu s'arrêtant au point (u_n, u_{n+1}).
   b) Tracer huit graphiques (répartis sur quatre lignes dans une même fenêtre graphiques) pour $n = 100$ et $(a, u_0) \in \{(0.9, 0.6), (1.7, 0.9), (2.3, 0.2), (2.8, 0.9), (3, 0.1), (3.4, 0.7), (3.5, 0.8), (3.9, 0.1)\}$.

Feuille de TP n° 8

Matrices et systèmes avec Scilab

Ce TP accompagne le chapitre 5 (Informatique et Algorithmique) : **Matrices avec Scilab**.

I Entrainement

1) Construction et modification d'une matrice

• Exécuter les commandes suivantes dans la console :

```
-->u=[-1,0,1,2,3]
-->v=[1;3;-2;5;0;2]
-->A=[7,8;9,10]
-->B=[1/7,2/5,-1/8,1;0,2,1/2,1/3;3,0,1/9,2]
-->L1=[2,-1,0,3,-4]; L2=[1,-8,9,-4,7]; L3=[0,0,7,-1,2]; M=[u;L1;L2;L3]
-->C1=[1;-1]; C2=[-2;3]; C3=[0;-4]; C4=[2;0]; N=[C1,C2,C3,C4]
-->[N,A;u,11]
```

• Construire les matrices suivantes avec Scilab : $A = \begin{pmatrix} 7 & 7 & -5 \\ -1 & 0 & -2 \end{pmatrix}$, $B = \begin{pmatrix} 6 & -4 \\ 1 & 3 \end{pmatrix}$,

$$X = \begin{pmatrix} 1 \\ -1 \\ 0 \\ 2 \\ -3 \\ -10 \end{pmatrix}, \quad C = \begin{pmatrix} 2 & -1 & 0 & 0 & 1 \\ 3 & 4 & -3 & 7 & 0 \\ 1 & -1 & 2 & 1 & -3 \\ 4 & -1 & 0 & 1 & 2 \end{pmatrix} \quad \text{et} \quad M = \begin{pmatrix} 2 & -1 & 0 & 0 & 1 & 1 \\ 3 & 4 & -3 & 7 & 0 & -1 \\ 1 & -1 & 2 & 1 & -3 & 0 \\ 4 & -1 & 0 & 1 & 2 & 2 \\ 6 & -4 & 7 & 7 & -5 & -3 \\ 1 & 3 & -1 & 0 & -2 & -10 \end{pmatrix}$$

On construira la matrice M par blocs à partir des matrices A, B, C et X (pas nécessairement avec une seule commande).

• Exécuter les commandes suivantes dans la console :

```
-->size(A), size(B), size(X), size(C), size(M)
-->length(A), length(B), length(X), length(C), length(M)
-->M(4,2), M(2,4), M(5,3), M(3,5)
-->M(2,:), M(5,:), M(:,1), M(:,4)
-->M([3,2],[1,5,6])
```

• Exécuter les commandes suivantes dans la console :

```
-->linspace(1,9,20)
-->1:2:20
-->zeros(3,6)
-->ones(5,2)
```

```
-->eye(3,3), eye(5,8), eye(8,5)
-->diag([1,-2,3,-4,5])
-->rand(4,7)
```

• Nous allons maintenant modifier des coefficients de la matrice M. Exécuter les commandes suivantes dans la console :

```
-->M(1,1)=21, M(5,6)=-13
-->M(4,:)=[], size(M)
-->M(:,1)=[0;0;1;0;0]
-->M(:,[2,3])=[], size(M)
```

```
-->M=[[0;1;0;1;0],M], size(M)
-->M=[M;[1,2,3,4,5]]
-->M([2,3],[1,4,5])=rand(2,3)
```

- Construire en Scilab :
  - la matrice élémentaire $E_{5,12} \in \mathcal{M}_{8,20}(\mathbb{R})$.
  - la matrice de $\mathcal{M}_{11,7}(\mathbb{R})$ dont tous les coefficients sont égaux à 1 sauf ceux se trouvant sur les lignes $2, 5, 6, 10$ et ceux sur les colonnes $3, 7$ qui valent 0.
  - la matrice de $\mathcal{M}_{11,7}(\mathbb{R})$ dont tous les coefficients sont égaux à 1 sauf ceux se trouvant à la fois sur les lignes $2, 5, 6, 10$ et sur les colonnes $3, 7$ qui valent 0.
  - une matrice diagonale d'ordre 7 dont les coefficients sont tirés aléatoirement entre 0 et 1.

2) Opérations sur les matrices

- Exécuter les commandes suivantes dans la console :

```
-->A=[1,0,1;0,1,0], B=[1,2,3;-4,-5,-6]
-->C=[2,4,0,1;1,5,-6,8;0,-2,0,4]
-->A', B', C'
-->A+B, A-B
-->3*A, A/4
-->A.*B, A./B, B.^A, A.^3
```

```
-->A*B
-->B^A
-->A^3
-->A*C, B*C
-->C*A
-->M=[1,2;-3,5]; M*M, M^2
```

Insistons bien sur la différence entre :

- le produit et la puissance terme à terme sur des matrices de même taille (avec `.*` et $.\wedge$).
- et le produit matriciel (avec des matrices de tailles compatibles avec le produit) et la puissance de matrices carrées (avec `*` et $\wedge$).

- Exécuter les commandes suivantes dans la console :

```
-->A=[1,2,-1;3,6,-4;4,3,0;0,1,2;-6,8,1]
-->B=[1,2,3;-1,5,-4;7,-2,2;1,-1,2;0,2,0]
-->A<B, find(A<B)
-->A==B, find(A==B)
```

- Exécuter les commandes suivantes dans la console :

```
-->A=[1,2,3;-4,-5,-6];
-->min(A), max(A)
-->sum(A), prod(A)
```

3) Inversion de matrices et résolution de $AX = B$

- Exécuter les commandes suivantes dans la console :

```
-->A=[-3,-2,-5,-3;-2,0,-1,-2;-2,-3,-5,-3;-1,-4,-4,-2]
-->B=inv(A)
-->A*B, B*A
```

Que fait la commande `inv(A)` ?

- Exécuter les commandes suivantes dans la console :

```
-->C=[2,0,-1,-1,-3;4,2,-4,3,2;-2,1,-1,-3,5]
-->inv(C)
-->rank(C)
```

```
-->K=kernel(C)
-->C*K
-->rank(A)
-->kernel(A)
```

Que font les commandes `rank(C)` et `kernel(C)` ?

- Si `M` est une matrice implémentée en Scilab, que renvoie la commande

`rank(M)+size(kernel(M),'c')-size(M,'c')` ?

Pourquoi ? Tester-la avec les matrices `A` et `B`.

• Exécuter les commandes suivantes dans la console :

```
-->A=[-3,-2,-5,-3;-2,0,-1,-2;-2,-3,-5,-3;-1,-4,-4,-2], B=[-1;4;-2;-3]
-->X=inv(A)*B
-->[X0,K]=linsolve(A,-B)
```

Que fait la commande `linsolve(A,-B)` ? Cette commande n'a pas grand intérêt dans le cas où `A` est inversible. Par contre, si ce n'est pas le cas :

```
-->A=[2,0,-1,-1,-3;4,2,-4,3,2;-2,1,-1,-3,5], B=[-9;-1;7]
-->[X0,K]=linsolve(A,-B)
-->A=[-2,4,-1;-5,-3,4;8,10,-9], B=[2;3;-1]
-->[X0,K]=linsolve(A,-B)
```

II Exercices

Exercice 1. (★) Résoudre les systèmes linéaires suivants à l'aide de Scilab :

1) $\begin{cases} x + \frac{y}{2} - \frac{z}{2} = \frac{5}{2} \\ \frac{x}{3} + \frac{5y}{3} + \frac{3z}{2} = \frac{7}{3} \\ \frac{x}{3} - \frac{4y}{3} - \frac{11z}{6} = -\frac{1}{2} \end{cases}$

2) $\begin{cases} 3y - 4z = -7 \\ 5x + 2z = 4 \\ -2x - y = 3 \end{cases}$

3) $\begin{cases} 2x + + 5z = 2 \\ x + y + z = 3 \\ + 4y - 6z = 8 \\ -x - 3y + 2z = -7 \end{cases}$

4) $\begin{cases} x - 2y + 4z = 1 \\ 3x - y - z - 2t = 0 \\ -5x + 6z + 4t = 1 \\ 2x - 4y + 8z = -1 \end{cases}$

5) $\begin{cases} \frac{x}{5} + y - z = \frac{1}{3} \\ \frac{x}{4} - \frac{y}{2} - \frac{z}{3} = -\frac{1}{2} \end{cases}$

6) $\begin{cases} 2x + 6y + z - 3t + 7u = -3 \\ 3x + 5y + z - 2u = 2 \\ 2x + 8y + 5z + 6t - 7u = 5 \\ -4x - 2y + 3z + 6t - 3u = 1 \\ -5x - y + 3z + 3t + 6u = -4 \end{cases}$

7) $\begin{cases} ix + 2y - z = -5 + 2i \\ 3x - 4iy = 9 \\ ix + 7y + 3iz = 3 + i \end{cases}$

8) $\begin{cases} -ix - y + z = 2 - i \\ + (1+i)y + 2z = 1 - i \\ x - iy + (1-i)z = 2 \\ x + (1-2i)y - iz = i \end{cases}$

Comparer avec les résultats théoriques que l'on a déterminés dans la feuille d'exercices n° 16.

Pour préparer la section suivante, veuillez sauvegarder les commandes qui implémentent en Scilab les matrices et vecteurs associés à ces systèmes (par exemple en faisant des copier-coller dans un fichier `.txt`*).*

Exercice 2. (★) Implémenter en Scilab une fonction qui prend en entrée deux entiers naturels n et p strictement positifs et qui renvoie la matrice de taille $n \times p$ dont les coefficients sont les entiers de 1 à np (rangés par ordre croissant ligne par ligne).

Exercice 3. (★★) Soit $A = \begin{pmatrix} 3 & 1 & -2 \\ 0 & -1 & 0 \\ -4 & -5 & -3 \end{pmatrix}$. Écrire un programme en Scilab qui demande un entier naturel n à l'utilisateur et qui calcule A^n (sans utiliser la commande $\wedge$). Peut-on calculer A^n si $n \in \mathbb{Z}\backslash\mathbb{N}$? Si oui, modifier le programme en conséquence.

III Algorithme du pivot de Gauss

Voici une fonction en Scilab qui prend en argument une matrice A et renvoie une matrice échelonnée obtenue via l'algorithme de Gauss appliqué à la matrice A :

```
function A=PivotGauss(A)
  [n,p]=size(A);
  for  j=1:min(n-1,p)
       //On recherche le premier pivot non nul pour la colonne j :
       pivot=0; i0=j-1;
       while  (pivot==0)&(i0<n)
              i0=i0+1;
              pivot=A(i0,j);
       end
       //Ce premier pivot non nul (s'il existe) est en position i0.
       if  pivot<>0  then
           //On échange les lignes i0 et j :
           [                                ]
           //On annule les coefficients de la colonne j
           //dans les lignes suivant la ligne j :
           for  k=(j+1):n
                [                                ]
           end
       end
  end
endfunction;
```

1) Compléter cette fonction Scilab et implémenter-la dans un script nommé `PivotGauss.sci`. Tester-la avec les matrices associées aux systèmes de l'exercice 1.
2) Modifier la fonction afin que :
   - elle prenne en argument deux matrices A et B (ayant la même nombre de lignes),
   - elle effectue les mêmes opérations élémentaires sur les lignes de B que sur les lignes de A,
   - elle renvoie une matrice échelonnée A_0 et une matrice B_0 de telle sorte que, si B est un vecteur colonne, alors le système $AX = B$ est équivalent au système $A_0X = B_0$.

   Nommer `PivotGauss2.sci` cette fonction. Tester-la fonction avec les matrices et vecteurs associés aux systèmes de l'exercice 1.