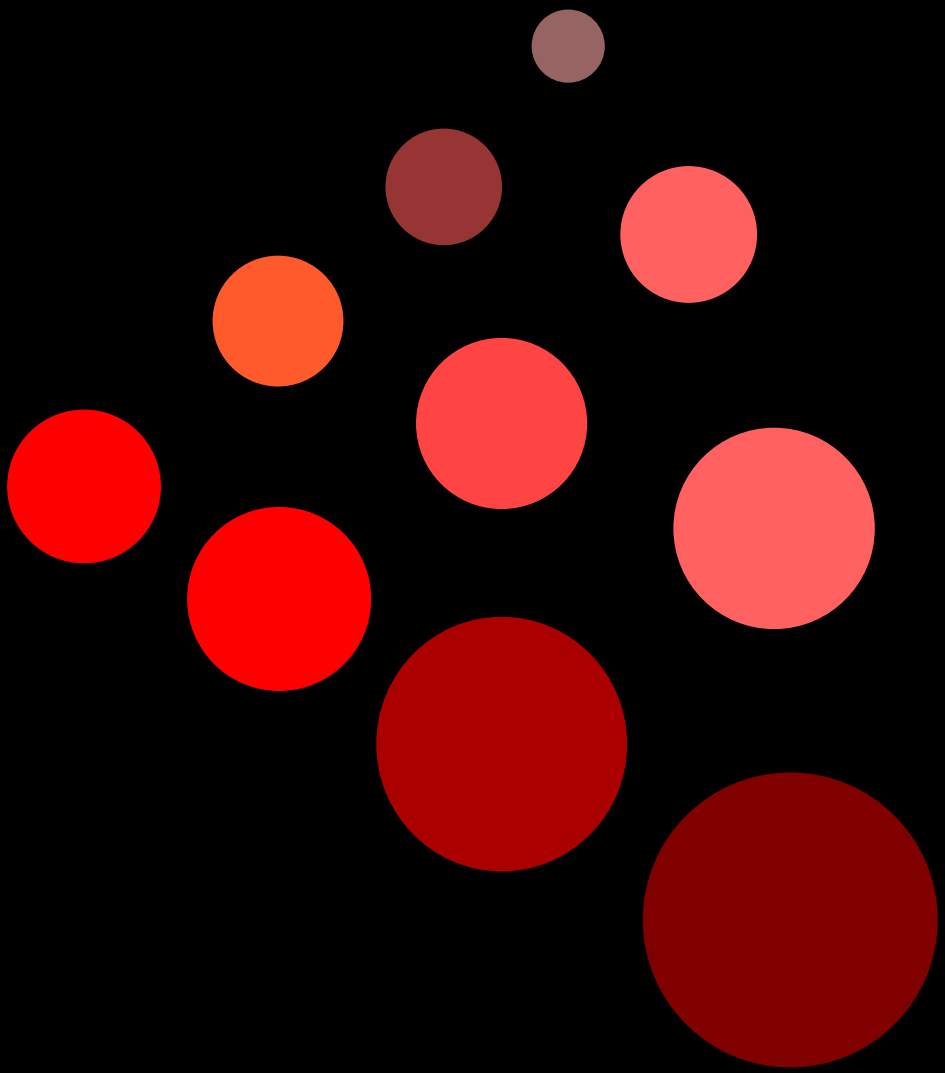
TP d'informatique
en
ECS1
avec

SCILAB

(E.D.E.N.)



Jean-Marc Berrou

Avant-propos

Voici les sujets de TP d'informatique qui ont été proposés à l'E.D.E.N. lors de l'année scolaire 2014-2015 en ECS1. Il s'agit d'un travail *ex nihilo*, un « premier jet » certainement perfectible. J'ai corrigé certaines fautes, mais il doit en rester. Les choix pédagogiques sont certainement discutables, d'ailleurs, je le referai(s) différemment aujourd'hui. Je n'ai toutefois pas le temps de reprendre avec ce que les élèves m'ont appris de ce qui était clair pour eux ou non. J'en profite pour les remercier pour leurs nombreux retours.

Je rends texte disponible en espérant qu'il puisse être de quelque utilité, soit pour l'apprentissage de Scilab, soit pour l'apprentissage de ConTeXt. C'est aussi en hommage à tous celles et ceux qui partagent leurs connaissances sur la toile que je partage cette contribution. Le contenu de ce document leur doit beaucoup et, si je n'ai pas toujours noté mes sources au fur et à mesure, je tiens à remercier tous ces mécènes.

Le texte a été mis en forme en TeX, plus exactement avec ConTeXt, surcouche moins connue en France que LaTeXmais qui offre plus de possibilités de personnalisation des documents. Si vous êtes intéressés par les sources, je vous les communiquerai sur simple demande en espérant un retour qui puisse m'aider à améliorer le contenu des TP et/ou le code ConTeXt

Je vous souhaite une bonne lecture.

Jean-Marc Berrou

Table des matières

TP6 | RÉSOLUTION APPROCHÉE D'ÉQUATIONS

TP7 | VARIABLES ALÉATOIRES

TP8 | INTÉGRATION

TP9 | INTÉGRATION

TP 10 | DERNIER TP, BILAN, EXERCICES

TP1 | Découverte de SCILAB

1.1 But du cours et découverte du logiciel

Le but de ces TP d'informatique est de se familiariser avec le logiciel SCILAB , sa syntaxe, et la création d'algorithmes simples dans ce langage en lien avec le programme de mathématiques. Lors des écrits de mathématiques, une ou plusieurs questions d'algorithmique seront posées et il faudra y répondre *sur papier* en langage SCILAB . Ces questions ne sont en général pas les plus difficiles pour qui y est préparé et **la meilleure préparation est de pratiquer régulièrement**.

Vous pouvez installer le logiciel SCILAB sur votre ordinateur personnel car c'est un logiciel libre, distribué gratuitement sur le site. Vous pouvez l'utiliser comme une 'super calculatrice' pour vérifier vos calculs, faire des simulations....

Il est temps maintenant de lancer SCILAB (et attendre les instructions du professeur)

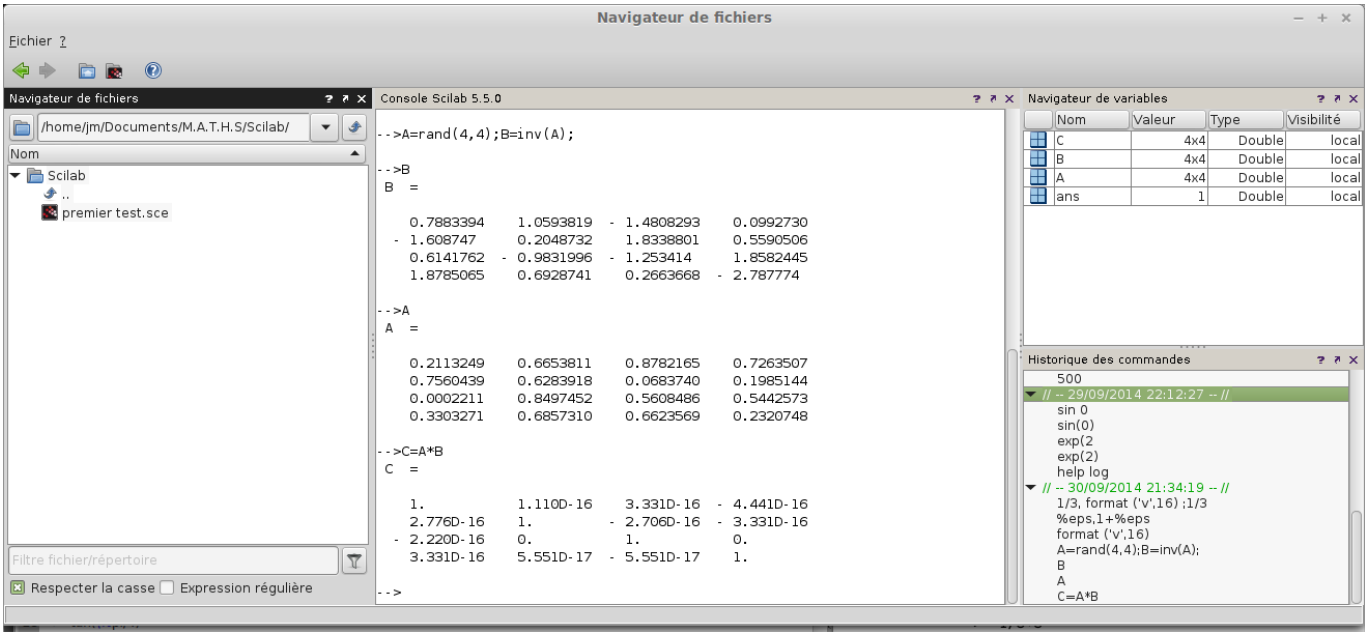


Figure 1.1 : La fenêtre principale de SCILAB . Celle-ci est divisée en plusieurs zones

Celle de gauche est un navigateur qui permet de retrouver vos fichiers (cf prochains TP). Au centre, la partie où l'on tape les commandes et où s'affichent les résultats. C'est celle que l'on utilisera le plus. En haut à droite, un récapitulatif de toutes les variables utilisées, avec leur valeur actuelle. En dessous, l'historique des commandes. Si vous avez besoin de retrouver une commande déjà tapée, pas besoin de le refaire, elle a été conservée en mémoire et est disponible dans ce cadre.

1.2 Premières opérations

On peut utiliser Scilab comme une calculatrice, pour faire des opérations simples. C'est un bon moyen de se familiariser avec les principales commandes et peut vous permettre plus tard de vérifier vos exercices (intégrales, matrices...) tout en révisant l'informatique.

NB : En cas d'erreur, vous pouvez retrouver la dernière frappe (pour la modifier) avec la flèche 'haut' ou dans la zone d'historique en bas à droite.

Exercice 1 Scilab comme 'calculatrice'

1. Effectuer les calculs suivants
(certaines commandes ne fonctionnent pas, c'est délibéré...)

```
-->  1+1
-->  1+8*9
-->  2^3
-->  45/7+2
-->  1/3*3
-->  sin0
-->  sin(0)
-->  sin(%pi/2)
-->  cos(%pi/12)
-->  tan(%pi/4)
-->  ans+1
-->  %e
-->  ln(%e)
-->  (%i+1)^2
-->  %e^(%i*%pi/4)
-->  rand()
```

2. Taper `help log`. Sur la page d'aide, vous trouverez les commandes qui permettent de calculer $\ln(3)$ et e^5

3. La touche F1 donne accès à l'aide générale. Prenez un moment pour voir la richesse du logiciel en attendant de passer à la suite.

À retenir

- La commande `help` permet d'obtenir de l'aide sur n'importe commande. Vous pouvez aussi utiliser la touche F1.
- Les principales constantes mathématiques s'obtiennent avec % par exemple, `%pi, %e, %i`
- la variable `ans` (comme *answer*) contient le résultat du dernier calcul.

Quelques fonctions usuelles de SCILAB *:*

- Racine carrée : `sqrt()`
- Module : `abs()`
- Partie entière : `floor()`
- Logarithme népérien : `log()`
- Logarithme en base 10 : `log10()`
- Exponentielle : `exp()`
- Cosinus : `cos()`
- Sinus : `sin()`
- Tangente : `tan()`
- Arccos : `acos()`
- Arcsin : `asin()`
- Arctan : `atan()`

La commande `rand()` renvoie un nombre aléatoire entre 0 et 1

Exercice 2 utiliser la commande `rand` pour générer un nombre entier aléatoire compris entre 10 et 20

Exercice 3 Affectations, utilisation de variables

Tapez les instructions suivantes dans la console en anticipant mentalement l'affichage. Regardez aussi de temps en temps les fenêtres de droites pour voir ce qui s'y passe.

```
-->  a=2
-->  a
-->  a=5 ;
-->  a
-->  a=a+3
-->  a=a+3;
-->  a
-->  b=a
-->  c=a-3*b
-->  a, b, c
-->  a=3; b=2; c=a*b;
-->  c
-->  a+a+a+..
-->  1
-->  1+..
-->  2
-->  who
```

Vous pourrez ensuite cliquer dans la fenêtre en bas à droite sur `a=3 ; b=2 ; c=a*b ;` plusieurs fois. Cela permet de rappeler la commande sans tout retaper. On peut également utiliser la flèche `haut' du clavier.

Attention

- Le symbole = dans Scilab n'a pas le sens usuel du = mathématique. Il représente en fait la flèche ← d'affectation. Scilab effectuera toujours le calcul à droite du =, avec les valeurs en cours avant de rentrer le résultat dans la variable à gauche du =
- Noter la différence entre , et ; qui peut être importante si l'on fait tourner un programme qui effectue des milliers de calculs...
- On peut taper plusieurs calculs sur une même ligne en les séparant par , ou ;
- Si on souhaite répartir un calcul sur deux lignes, on peut le « découper » avec . .

On voit de plus en plus souvent dans les livres récents de mathématiques l'utilisation de := pour définir un nouvel objet de mathématique, alors que = est utilisé pour les identités et les équations.

Exercice 4 Entrez votre année de naissance dans la variable `an` et écrivez un calcul avec cette variable.
Complément : si vous avez fini, sauriez-sous trouver une commande qui soit encore valable dans quelques années ? (vous pourrez taper `help date` pour des idées supplémentaires)

Conseil

Dans les programmes, préférez des noms de variables explicites, comme `an`, `age`, `moyenne`, plutôt que des lettres `n`, `N`, `x`.
Bien que plus long à taper, cela facilite la lecture.

Exercice 5 Supposons que `a` et `b` sont deux variables déjà définies dans Scilab. Donner une suite d'instructions qui échange les valeurs des variables `a` et `b`, puis fait afficher les nouvelles valeurs de `a` et `b`.
Tester votre programme (prendre par exemple `a=5` et `b=3`).

1.3 Logique et booléens

Exercice 6 : **Logique et Booléens**
Entrez les commandes suivantes :

```
-->  a=2
-->  a==2
-->  a==5
-->  a<5
-->  a>5
-->  c=a>5
-->  c
-->  b=%T
-->  b|c
-->  b&c
```

```
-->  ~(2==3)
```

- SCILAB permet d'effectuer des tests logiques, dont la réponse est *vrai* (, true) ou *faux* (, false). Cela sera utile dans les algorithmes pour introduire des conditions avec `if` ou `while`...
- On peut combiner les conditions avec les connecteurs *et* (noté `&`) et *ou* (noté `|` obtenu avec AltGr+6) ainsi que la négation `~` (AltGr+2)
- Les valeurs *vrai* (, true) ou *faux* (peuvent être affectées à des variables avec et

---

Exercice 7 Prévoir les réponses de SCILAB aux commandes suivantes :

```
-->  (1>2)|(8<9)|(2<>5)
-->  (abs(3+4*%i)>4)&((2==2)|(3==4))
-->  ~((%T)|(%F))
-->  ~((%T)&(%F))
-->  ~((%T)|((%F)&(abs(2-3*%i)>log(%e 4))))
```

Le == est un **test**. Il renvoie vrai ou faux. Il correspond par exemple au = des équations en mathématiques.

Les principaux tests avec SCILAB

- égalité `==` ou différence `<>`
- inégalité stricte `<` et `>`
- inégalité large `<=` et `>=`

$\neq$ se code en losange car `><` est plutôt un signe de colère en émoticones...

NB : Nous avons donc bien deux notations pour le = habituellement utilisé en mathématiques :

- le `=` pour l'affectation et
- le `==` pour tester l'égalité.

1.4 Chaînes de caractères

Exercice 8 Observer l'effet des commandes suivantes :

```
--> a='bonjour'
--> a+a
--> a=a+' '
--> a
--> a=a+a
--> 'bonjour', 'bonjour'
--> A='Affecter à une variable une chaîne de caractères, ..
--> c''est simple ..
--> comme ''bonjour'' '
--> a+A
--> length(a+A)
--> part(a,3)
--> part(a,3:7)
--> clear
--> a
```

À retenir

- on définit une chaîne de caractères avec des guillemets '(3) ou des apostrophes ''(4). Ces caractères ont donc un statut spécial et pour les afficher il faut les doubler.
- `+` ici réalise une *concaténation*. Les chaînes de caractères sont accolées (concaténées)
- On peut extraire une partie d'une chaîne de caractère avec la commande `part`
- De même que pour les vecteurs, la commande `length()` renvoie la longueur de la chaîne

1.5 La vraie nature de SCILAB

SCILAB est en fait bien plus qu'une calculatrice. Il est utilisé en ingénierie ou en finance pour des simulations très avancées. C'est une sorte d'expert en gros calculs.

Pour cela, les outils de base dans SCILAB ne sont pas les nombres mais les vecteurs (ou plutôt *matrices*). SCILAB peut effectuer une même opération sur toutes les composantes d'un vecteur.

Pour définir un vecteur composante par composante (*cf les exemples ci après*) ou avec la commande suivante très pratique :

`nombre1 :pas :nombre2`

pour laquelle quelques exemples valent mieux qu'un long discours.

Exercice 9 Vecteurs
Entrez les commandes suivantes

```
--> 1:1:5
--> 1:5
--> 1:2:5
--> 1:3:5
--> a=1:0.1:10
--> a^2
--> exp(a)
--> sin(a)
--> b=1:5
--> a+b
--> b+b
--> a=3:7
--> a+b
--> a+exp(b)
--> help sum
--> sum(a)
--> length(a)
--> a'
--> a=[1,2,3]
--> a=[1;2;3]
--> a=a'
```

Premiers pas avec les vecteurs

- `nombre1 :pas :nombre2` crée le vecteur dont les coordonnées sont tous les nombres de `nombre1` inclus à `nombre2` en allant de `pas` en `pas`
- les vecteurs peuvent être très grands et permettre d'effectuer plusieurs opérations en une ligne de commande
- on peut effectuer la somme de toutes les composantes d'un vecteur avec `length()`, calculer la somme de toutes les composantes avec `sum()`, transposer un vecteur ligne en vecteur colonne avec le `'`

Exercice 10

1. Calculez avec Scilab la somme des 100 premiers carrés de nombres entiers
2. Calculez avec Scilab la somme des 1000 premiers cubes de nombres entiers
3. Calculez $\sum_{k=1}^{100} \sin(k)$

Le deuxième calcul nous rappelle que Scilab calcule des valeurs *approchées*, c'est l'une des raisons de sa rapidité d'exécution. Un logiciel de calcul formel comme Xcas ou Sage pourrait donner une valeur exacte.

1.6 Un peu de dessin

Si jamais rien ne s'affiche, essayer : `usecanvas(%T)` dans la console puis réessayer

Scilab dispose également de commandes graphiques qui rendent mieux grâce aux nombres que des listes interminables.

Vous aurez certainement besoin de la commande
`clf()` (comme *clear fenêtre* ^^) qui efface la fenêtre graphique

Exercice 11

1. créez un vecteur `x=-10:10`
2. en tapant `[x,x^2]`, vous obtenez un tableau de valeurs de la fonction carré
3. tapez maintenant `plot(x,x^2)` et vous obtenez la courbe de la fonction carré. Vous auriez pu taper `plot(x^2)`
4. comparez
   `x=-10:10;plot(sin(x))` et
   `x=-10:0.1:10;plot(sin(x))`
5. en cas de problème d'affichage, pensez à fermer la fenêtre ou créer un nouveau graphique.

Exercice 12

Trouvez une commande qui affiche un cercle de centre 0 et de rayon 1.

Si vous avez tout terminé, vous pouvez taper `help plot` pour voir les différentes options de tracé.

TP2 | Structures conditionnelles

Pour les concours, il vous sera demandé de comprendre le fonctionnement d'algorithmes, d'être capables d'en écrire ou d'en modifier. Ceci implique deux compétences distinctes :

- ⋆ comprendre les ressorts généraux de l'algorithmique d'une part
- ⋆ connaître la syntaxe de Scilab d'autre part.

Pour ce deuxième TP, nous allons voir comment écrire des algorithmes avec Scilab

2.1 Quelques rappels d'algorithmique

Un algorithme est une suite finie d'instructions dont l'exécution dans l'ordre aboutit à la réalisation d'une tâche prédéfinie.

Un algorithme comprend en général :

- ⋆ des entrées
- ⋆ un traitement (les différentes instructions)
- ⋆ des sorties

Un exemple bien connu d'algorithme est la recette de cuisine. En entrées[1] nous avons un certain nombre d'ingrédients. Le traitement consiste en les différentes étapes d'élaboration.

2.1 au pluriel, ce n'est pas l'entrée du menu ici

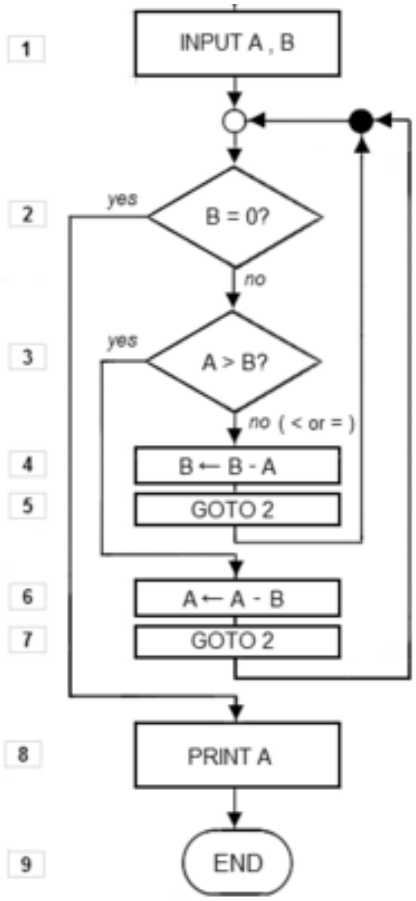
La sortie est le plat finalisé. Si cet exemple paraît naïf, cette logique algorithmique est poussée à l'extrême dans les usines agroalimentaires... et de manière générale la pensée algorithmique est l'un des ressorts du monde industriel rationnalisé.

Pour être un peu plus précis sur ce qui fait qu'un ensemble de commandes est, ou non, un algorithme, voici les cinq critères que l'éminent informaticien Donald Knuth[2] a énoncés :

- la **finitude** : « Un algorithme doit toujours se terminer après un nombre fini d'étapes. »
- **définition précise** : « Chaque étape d'un algorithme doit être définie précisément, les actions à transposer doivent être spécifiées rigoureusement et sans ambiguïté pour chaque cas. »
- **entrées** : « ... des quantités qui lui sont données avant qu'un algorithme ne commence. Ces entrées sont prises dans un ensemble d'objets spécifié. »
- **sorties** : « ... des quantités ayant une relation spécifiées avec les entrées. »
- **rendement** : « ... toutes les opérations que l'algorithme doit accomplir doivent être suffisamment basiques pour pouvoir être en principe réalisées dans une durée finie par un homme utilisant un papier et un crayon. »

[2.2] professeur émérite de l'université de Stanford né en 1938, connu comme l'auteur de *The art of computer programming*, un des rares ouvrages d'informatique des années 70 qui soit encore une référence aujourd'hui. Il est également l'inventeur des langages TEX et MetaPost qui servent encore aujourd'hui à la conception des livres de mathématiques du commerce et des documents plus modestes comme ce polycopié

Les algorithmes sont souvent représentés par un organigramme comme les exemples ci-dessous :



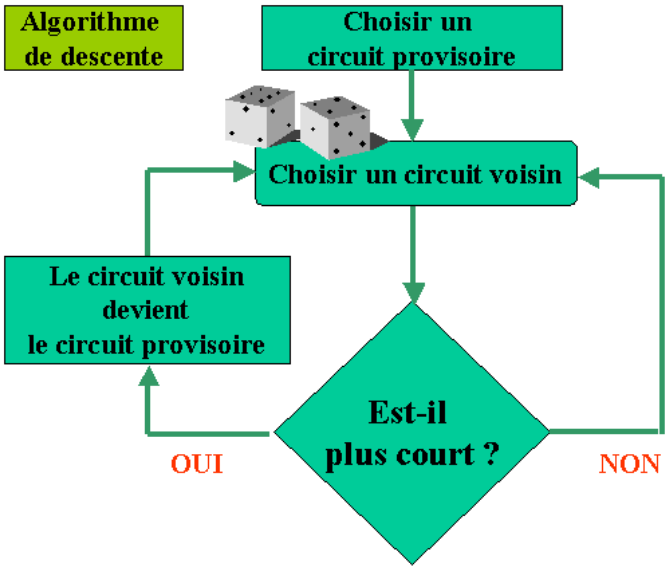


Tableau 2.1 :

Exercice 13 À quoi servent les deux algorithmes précédents ? En quoi le second (algorithme de descente) n'est-il pas un algorithme au sens de Knuth ?

2.2 Bien communiquer, c'est important : entrées et sorties

La communication avec une machine, ce sont les entrées et sorties de l'algorithme. Il faut donc bien retenir les commandes qui permettent à un programme de **demander** à l'utilisateur d'entrer une valeur, une chaîne... et les commandes qui permettent au programme d'afficher quelque chose (phrase, valeur, graphique ou combinaison de tout cela).
La commande qui permet au programme de demander quelque chose à l'utilisateur est `input`.

Certains langages, comme le C ou le Pascal demandent en plus de 'préparer le matériel', c.a.d. d'annoncer quels seront les espaces de mémoires à allouer, et pour quel type de données. Scilab , est un langage de haut niveau, le plus bas niveau étant le langage machine, le plus haut niveau la langue naturelle parlée et il s'adapte à ce qu'on lui propose. Il gère la mémoire au fil des instructions.

Par exemple, tapez en console :

```
n=input('entrez un nombre entier : ')
```

Qu'elle existât antérieurement ou non, la variable `n` aura désormais la valeur que vous lui aurez attribuée en répondant à cet `input`. Notez également que si vous ne souhaitez pas confirmation que la valeur a été enregistrée, vous pouvez terminer la commande par `;`

Scilab peut afficher du texte, des variables avec `disp`.
Vous pouvez essayer :

```
-->  disp(2*n,'le double du nombre est : ')
-->  disp('le double du nombre est : ', 2*n)
-->  disp([1,2],3)
```

Vous aurez remarqué que les variables sont affichées de la dernière à la première. C'est un point qui peut surprendre, et donc à bien retenir.

Figure 2.1: La syntaxe, retenir tu dois

2.3 Enregistrer son travail : Scinotes

Dans le menu `Applications` de Scilab , vous trouverez Scinotes . Il s'agit d'un éditeur de programme qui vous permettra de sauvegarder vos algorithmes et de les retrouver au besoin. Vous pouvez par exemple taper les commandes précédentes dans Scinotes pour en faire un programme :

```
 n=input('entrez un nombre : ');
disp(2*n,'le double du nombre est :'), // attention à l'ordre inversé
```

Si vous appuyez sur ▷, le programme se lancera mais il vous sera demandé de le sauvegarder avant exécution.

Dans votre espace personnel, créez un dossier `TP_Scilab` et placez vous dedans. Choisissez ensuite comme nom de fichier `TP1-exemple`, le logiciel lui accolera automatiquement l'extension `.sce` caractéristique des fichiers Scilab .

Quand on exécute le programme, la console Scilab affiche un message du genre `-->exec('/TP_Scilab/TP1-exemple.sce', -1)` qui confirme que le programme a bien été lancé.

Viennent ensuite le résultat des commandes effectuées au fil du programme.

Remarques

- Retenez l'utilisation de `//` pour écrire des commentaires qui sont affichés en italique vert par Scinotes
- Un bon code est un code **commenté**. Ce qui vous paraît évident aujourd'hui ne le sera peut-être pas dans un mois !
- On peut remarquer qu'à chaque modification, le script est enregistré avant d'être exécuté, mais le nom n'est demandé qu'au premier lancement. **N'oubliez pas de changer de fichier quand vous créez un nouveau programme !** En cas d'oubli, vous « écrasereriez » le fichier de l'exercice n avec avec le script de l'exercice $n+1$.
  Ceci a deux fâcheuses implications : vous perdez le script n et le script $n+1$ est enregistré sous un faux nom ! Soyez vigilants.
- Pensez à **organiser** vos scripts. On peut les organiser par nom ou par dossier. Ici, je propose une organisation par nom. Chaque fichier s'appellera `TP2-ex3.sce` par exemple ou `TP2-exemple1.sce` pour le 3^{me} exercice du TP n°2.
- À l'instar des noms de variables, des noms de fichiers bien pensés peuvent représenter un gain de temps pour une utilisation ultérieure.
- **Vous pouvez désormais enregistrer tous vos programmes.**

Exercice 14
Écrire un programme qui demande un nombre a et renvoie la valeur de $(a + \ln(1 + |a|))^2$
Certaines entrées donnent-elles un message d'erreur ?

2.4 Structures conditionnelles `if ... then ... else`

Un algorithme ne consiste pas seulement en une suite d'instructions mises bout à bout. Le comportement peut varier selon des cas prédéfinis à l'avance et effectuer telle ou telle opération selon que telle ou telle condition est remplie. La syntaxe Scilab correspondant est la suivante :

```
if (condition) then
   Instruction 1
   Instruction 2
   ...
```

Une *condition* ici est un booléen, c.a.d une expression du type :

- `a==1` pour $a = 1$
- `a<1` pour $a < 1$
- `a<>1` pour $a \neq 1$
- `(a>1)&(b<=2)` $a > 1$ et $b \leqslant 2$
- `(a==1)|(b>=2` pour $a = 1$ ou $b \geqslant 2$

Les `else` et `elseif` sont optionnels. Dans ce cas, il ne se passe rien si la condition n'est pas remplie.
`elseif` permet d'ajouter un test si le premier n'est pas rempli.

Exercice 15 : Voici un programme

```
birth=input("Année de naissance (ex:1996): ");
year=getdate();      // getdate() renvoie un tableau avec
l'année
age=year(1)-birth;
if age >= 18 then
   disp(" ans, vous êtes majeur !", age, "Vous avez ")
else
   disp(" ans, vous êtes mineur !", age, "Vous avez ")
end
```

1. Lisez attentivement cet exemple pour en comprendre le fonctionnement, que vous décrirez en une phrase.
2. Copier ce programme dans SCINOTES , enregistrer sous `age.sce`. Exécuter le programme plusieurs fois, en testant des entrées différentes.
3. Tester le programme avec une entrée supérieure à 2015 et le modifier le texte du programme pour que ce genre de cas renvoie une phrase d'alerte *ad hoc*

Remarques

- Remarquez l'indentation. SCINOTES est conçu pour la proposer automatiquement. **Lors de vos écrits, n'oubliez pas d'indenter** vos commandes entre `if ... then ... else`
- Le `else` n'est pas obligatoire si, par exemple, on souhaite exécuter des instructions uniquement si le test est vrai et ne rien faire sinon.
- Il est tout à fait possible d'imbriquer plusieurs alternatives les unes à l'intérieur des autres.

Exercice 16 *Racines carrées*

Écrire un programme qui demande un nombre et renvoie sa racine carrée ou message d'erreur *ad hoc* si l'entrée est un nombre négatif.

Exercice 17 *Fonctions affines*

Sauriez-vous écrire un algorithme qui prenne en entrée deux nombres a et b et

renvoie, en sortie, la résolution de l'équation $ax + b = 0$? L'algorithme devra proposer une sortie adéquate y compris si $a = 0$.

Exercice 18 *Second degré*

Tout un chacun en ECS sait résoudre une équation du type $ax^2 + bx + c = 0$. C'est toujours la même chose n'est-ce pas ? Derrière chaque « c'est toujours la même chose », se cache toujours un algorithme.
Sauriez-vous écrire un programme en Scilab, qui donnerait pour tous les cas (y compris le cas $a = 0$) une phrase présentant les solutions éventuelles.

Exercice 19 *Si vous êtes en avance...*

Écrire un programme `shifumi.sce` ... qui porte bien son nom

shi-fu-mi est le nom japonais du jeu « pierre, feuille, ciseaux ». Le programme devra donc afficher de manière aléatoire l'un des trois mots « pierre », « feuille », ou « ciseaux ». Et si vous êtes vraiment en avance, le programme pourrait afficher une invite vous demandant d'entrer 1, 2, ou 3 pour « pierre », « feuille », ou « ciseaux » puis tire au hasard son propre tirage avant de décider du vainqueur.
Si vous êtes toujours en avance, posez-vous la question : Ne rêveriez-vous pas secrètement de travailler dans l'informatique ?

2.5 Quelques corrigés :

```
// TP2-ex3 ////////////////////////////////////////
// majeur / mineur
birth=input("Année de naissance (ex:1996): ");
year=getdate();
// getdate() renvoie un tableau avec l'année
age=year(1)-birth;
if age<0 then // exclut le cas où l'age est négatif
    disp(year(1),"vous ne pouvez pas être né après ")
else
    if age >= 18 then
        disp(" ans, vous êtes majeur !", age, "Vous avez ")
    else
        disp(" ans, vous êtes mineur !", age, "Vous avez ")
    end
end
```

```
// TP2-ex4 ////////////////////////////////////////////
nombre=input("Donner le nombre dont vous voulez la racine
carrée : ");
if isreal(nombre)&(real(nombre)>=0)
    disp(sqrt(nombre))
else
    disp("le nombre rentré n''a pas de racine carrée")
end
```

```
// TP2-ex5  ////////////////////////////////////////////
disp("Veuillez entrer a puis b pour la résolution de l''équation
ax+b=0")
a=input("a=?")
b=input("b=?")
if a==0 then
    if b==0 then
        disp("tous les réels sont solution.")
        disp("il y a une infinité de solutions")
    else
        disp("l''équation n''a pas de solution")
    end
else
    disp((-b/a),"l''équation admet pour unique solution :
")
end
```

Ici, pour tester le programme, il faut au moins que les trois cas $a = b = 0$, $a = 0$ et $b = 1$, $a = 1$ et $b = 0$ aient été envisagés.

```
// TP2-ex6 ////////////////////////////////////////////
disp("Veuillez entrer a, b puis c pour la résolution de l''équation ax^2+bx+c=0")
a=input("a=?")
b=input("b=?")
c=input("c=?")
if a==0 then
    if b==0 then
        if c==0 then
            disp("tous les réels sont solution.")
            disp("il y a une infinité de solutions")
        else
            disp("l''équation n''a pas de solution")
        end
    else
        disp((-c/b),"l''équation admet pour unique solution : ")
    end
else
    delta=b^2-4*a*c
    if delta >0 then
        disp("l''équation admet deux solutions distinctes :")
        disp((-b+sqrt(delta))/(2*a))
        disp((-b-sqrt(delta))/(2*a),"et")
    else
        if delta==0 then
            disp(-b/(2*a),"l''unique solution est :")
        else
            disp("l''équation n''admet pas de solutions réelles.")
        end
    end
end
```

Ici encore, tous les cas particuliers doivent avoir été envisagés, toutes les combinaisons de o possibles.

```
// TP2-ex7 ////////////////////////////////////////////
// Un programme de "shi-fu-mi"
x=floor(3*rand())
if x==0 then
    disp('pierre')
else
    if x==1 then
        disp('feuille')
    else
        disp('ciseaux')
    end
end
```

TP3 | Boucles `for` et `while`

L'objet de ce TP sera l'utilisation des boucles, ou structures itératives, dans vos algorithmes. Au programme de ce TP :

- ⋆ la boucle `for` est indiquée quand on sait à l'avance le nombre d'itérations nécessaire.
- ⋆ la boucle `while` convient aux cas où on cherche à réaliser une condition sans savoir *a priori* le nombre d'itérations nécessaires pour y parvenir.

3.1 Boucle `for ... to ... end`

La **boucle itérative** `for` sert à exécuter une suite d'instruction un nombre prédéfini de fois. Ci-contre sa syntaxe générale :

```
for variable in valeur_debut:pas:valeur_fin
    instruction 1
    instruction 2
    . . .
end
```

La variable s'appelle aussi *compteur*. Les valeurs successives de cette variable peuvent être utilisées au fil de la boucle.

Exemples

Tapez les deux exemples ci-dessous dans SCINOTES :

```
a=2
for i=1:1:10
  a=a+3
end
```

Dans le premier cas, `i` a été utilisé purement pour compter. A chaque nouvelle valeur de `i`, `a` est augmenté de 3.

```
a=2
for i=3:2:10
   a=a+i
end
```

Dans le second exemple, `i` part de 3 et prend toutes les valeurs de 2 en 2 jusqu'à dépasser 10. Chacune de ces caleurs est tour à tour ajoutée à `a`
Autres exemples : Vous verrez mieux le fonctionnement de l'algorithme en plaçant l'affiche `disp` *dans* la boucle juste avant le `end`, pour voir comment les valeurs évoluent.

1) *Table de multiplication*

```
x=[] // crée un vecteur vide
table=input('table de ? ')
for i=1:1:10
    x=[x,i*table];
end
disp(x)
```

2) *Quelle somme ce programme calcule-t-il ?*

```
n=input("Entrer un nombre entier : ");
s=0;
for i=1:2:(2*n+1)
   s=s+1/i;
end
disp(s)
```

Le premier exemple calcul tous les *résultats* des tables de multiplication et les stocke dans un vecteur qui se construit au fur et à mesure. On pourrait enjoliver le résultat en affichant l'opération suivie du résultat.
Le second programme calcule $\sum_{k=1}^{n} \frac{1}{2k+1}$. On aurait pu écrire la boucle de manière plus intuitive avec :

```
n=input("Entrer un nombre entier : ");
s=0;
for k=1:1:n
   s=s+1/(2*k+1)i;
end
disp(s)
```

La version proposée est plus rapide (en termes de nombre d'opérations) et plus dans l'« esprit » Scilab.

À retenir...

- Noter que SCINOTES indente automatiquement les commandes entre `for` et `end`. Par souci de clarté, veillez à respecter cette convention dans la rédactions *à la main* de vos algorithmes.
- Retenir la syntaxe `x=[x,i*tab]` qui crée un vecteur avec le contenu de `x` « augmenté » de la nouvelle valeur
- n'hésitez pas à supprimer le `;` et refaire tourner le programme pour voir les étapes de construction du vecteur affiché en sortie.
- la commande `sum([1:2:2*n+1].^(-1))` permet de remplacer la boucle `for` pour ce genre de cas. Vous pouvez tester sur une ou deux valeurs.
- Dans le premier algorithme, on peut remplacer `1:1:9` par `1:9`. Je conseille d'utiliser systématiquement la syntaxe avec pas pour ne pas en oublier la tournure.
- Il est possible (et souvent utile) d'imbriquer plusieurs boucle `for`

Tableau 3.1 :

Exercices

▷ Pour comprendre un algorithme, il faut se mettre à la place de l'ordinateur. Suivez les instructions dans l'ordre et voyez comment les valeurs prises par les différentes variables évoluent au fil des commandes. Pour cela, on utilise en général un tableau comportant une colonne par variable, et une ligne par cycle.

Exercice 20

Pour chaque algorithme ci-dessous, décrire au brouillon les étapes de l'algorithme, puis l'affichage en sortie puis vérifier dans la console ou dans SCINOTES .

```
x=4 ;
for i=1:x
   y=0 ;
   y=y+i
;
end
y
```

```
x=4 ;
y=0 ;
for i=1:x
   y=y+i
;
end
y
```

```
a=5 ;
for i=1:3
   a=a+1
;
   x=a ;
end
x
```

```
a=5 ;
for i=1:3
   a=a+1
;
end
x=a ;
x
```

```
a=5 ;
x=a ;
for i=1:3
   a=a+1
;
end
x
```

Exercice 21

Écrire un programme qui demande à l'utilisateur un entier n et affiche en sortie le nombre $0,9^n \times 2$. Ce programme ne devra pas faire usage de la commande `^`

Exercice 22

Voici un algorithme :

```
u=0
for i=1:6
   u=2*u+3
   disp(u)
end
```

1. Reconnaître la suite (u_n) dont on calcule les termes et préciser le but de l'algorithme
2. Modifier l'algorithme pour qu'il calcule u_{100}

Exercice 23

Écrire un programme qui demande une valeur de n et calcule $n!$.

Si vous êtes en avance, améliorez votre code pour que le programme renvoie 1 pour $0!$ et un message d'erreur si n est négatif ou non entier.

Exercice 24 *Une somme*

Écrire un programme qui calcule la somme $S_n = \sum_{i=0}^{n} \sin(i)$ à l'aide d'une boucle for.

Si vous êtres en avance...

- ⋆ Saurez-vous retrouver la commande sans boucle faisant intervenir les vecteurs ?
- ⋆ Modifiez le programme précédent pour qu'il renvoie en sortie un vecteur `[min ;max]` où `min` (resp. `max`) est la plus petite (resp. grande) valeur prise par S_n, pour $n \in [\![1; n]\!]$

Exercice 25 *Les tables, la suite*

Écrire un programme qui affiche 10 vecteurs ligne contenant les résultats des tables de multiplication des nombres de 1 à 10.

Si vous êtes en avance, saurez-vous modifier l'algorithme pour qu'il présente de belles tables de multiplication comme au dos des cahiers de votre enfance ?

Exercice 26 *une suite...*

(u_n) est la suite définie par $u_0 = 0$ et $u_{n+1} = \sqrt{u_n + 1}$.

Écrire un programme qui calcule u_{10}. Le modifier pour qu'il demande une valeur n et crée un vecteur colonne qui contiennent les n premiers termes de la suite. Que remarque-t-on ? Modifier u_0, faire des tests et émettre une conjecture sur (u_n).

Exercice 27 *suite de Fibonacci*

La suite de Fibonacci est définie par :

$$\begin{cases} u_0 = u_1 = 1 \\ u_{n+2} = u_{n+1} + u_n \end{cases}$$

Écrire un programme qui demande une valeur n à l'utilisateur et calcul u_n.

Faire des tests et conjecturer le comportement de la suite quand n devient grand. Modifier ensuite légèrement le programme pour qu'il affiche également les termes de la suite $\left(\frac{u_n}{u_{n-1}}\right)_{n\geqslant 2}$ et en conjecturer le comportement à l'infini.

3.2 Boucle `while ... condition`

La boucle `while`, ou *itération conditionnelle* sert à répéter des commandes *jusqu'à* ce qu'une certaine condition soit réalisée. De même que dans les structures conditionnelles, la condition est un booléen.

```
while (booleen)
   instruction 1
   instruction 2
   . . .
end
```

Remarques

- On peut aussi ne jamais entrer dans la boucle si, dès le départ la condition n'est pas remplie.
- Il faut prendre garde au fait que cette structure peut donner lieu à une itération sans fin. En effet, si la condition n'est jamais remplie, le programme ne s'arrêtera jamais. Il faut donc toujours s'assurer que le programme se termine. Pour cela, on peut :
  - **démontrer** mathématiquement que la condition finira par être remplie
  - implanter un **compteur**, qui correspond à un nombre maximal autorisé d'itérations.

  On peut même envisager les deux au cas où le nombre d'itérations nécessaire, bien que théoriquement fini, soit un très très grand nombre.
- **Si le programme ne s'arrête pas**, ou pas assez vite, on peut en interrompre le déroulement. En effet, les menus restent accessibles et on trouvera la commande `abandonner` dans le menu `Contrôle`

Exemples :

On peut démontrer que la suite définie par $u_0 = 1$ et $\forall n \in \mathbb{N},\ u_{n+1} = 1 - e^{u_n}$ converge vers 0.
On souhaite trouver la plus petite valeur de n pour laquelle on a $|u_n| < 10^{-2}$.

Le programme suivant répond à la question.

```
x=-1;
n=0;
while abs(x)>10^(-2)
    x=1-exp(x);
    n=n+1;
end
disp(n)
```

Vous pouvez le taper dans SciNotes et constater qu'il faut 59687 itérations pour aboutir à la précision souhaitée. Notez que Scilab s'occupe de ces quelques calculs en 0,559s sur mon ordinateur. Si vous voulez savoir combien de temps met le votre,

si vous lancez plusieurs fois le programme, vous obtiendrez une fluctuation du temps de calcul. Peut-être l'occasion de faire des statistiques plus tard dans l'année...

ajoutez `tic()` juste avant la boucle `while` et `t=toc()` juste après. La variable `t` contient alors le temps d'exécution.

Remplacez maintenant `10^(-2)` par `10^(-3)` dans l'algorithme et exécutez le. Le temps nécessaire est un peu plus conséquent. 49,804s chez moi pour effectuer les 5996985 opérations !

```
x=-1;
n=0;
tic();
while abs(x)>10^(-2)
    x=1-exp(x);
    n=n+1;
end
t=toc()
disp("n = "+string(n))
disp("calculé en "+string(t)+"s")
```

Remarque

Dans l'exemple ci-dessus, la syntaxe de `disp` est légèrement différente de celle vue au TP précédent. Ici, on *concatène* les éléments de texte après avoir transformé la variable numérique `t` en texte (chaîne de caractères) via la commande `string()`. Cette syntaxe présente deux avantages :

- Les éléments du texte sont affichés dans l'ordre d'écriture
- la valeur est intégrée à la phrase (et non affichée en dessous)

3.3 Exercices

Exercice 28 *deux dés*

Écrire un programme qui simule le lancer de deux dés à six faces, affiche le résultat, et répète l'opération jusqu'à obtention d'un double.

Rappel : On peut simuler un dé avec `1+floor(6*rand())`

Nous verrons dans un prochain TP que l'on peut aussi simuler un dé avec une variable aléatoire suivant une loi uniforme sur $[\![1;6]\!]$, les principales loi de probabilités sont déjà implémentées dans Scilab

Exercice 29 *mission*

Votre programme, si vous l'acceptez, s'appellera `bond` (j'aime `bond`). Un nombre $x = 0$ bondira de +1 ou -1 [1] avec la même probabilité.
La mission du programme est de compter le nombre n de tirage nécessaires avant que $|x| > 007$

[3.1] x est augmenté ou diminué de 1

si vous êtes en avance, *lancez vous dans cette mission bonus : Votre programme demandera un nombre* p *de répétition de l'expérience, stockera les valeurs* n *obtenues successivement dans un vecteur afin de calculer le nombre moyen de bonds nécessaires.*

Q met à votre disposition le rappel suivant :

▷ `v=[]` *pour créer un vecteur vide*
▷ `v=[v;n]` *pour ajouter la valeur* n *au vecteur*
▷ `length(v)` *donne le nombre de termes de* v

Quand votre programme est fonctionnel, vous pouvez alors tester jusqu'à $p = 10000$ *pour avoir une bonne approximation du nombre moyen de tirages nécessaires, et observer comme le résultat se stabilise avec l'augmentation des valeurs de* p

---

Exercice 30 *Devinette*

Écrire un programme qui choisit au hasard un entier entre 1 et 100. Ensuite, le programme demande à l'utilisateur de deviner quel nombre a été choisi et répond « trop grand » ou « trop petit » selon la valeur proposée.
Le programme propose jusqu'à ce que l'utilisateur ait trouvé.

Si vous êtes en avance*, modifiez le programme pour qu'il traque les incohérences stratégiques du joueur en affichant le message :* « on a déjà dit que c'était plus grand/petit que <valeur> *». Pour cela, le programme comportera une variable* **`max`** *(resp.* **`min`***) gardant une mémoire de la plus grande (resp. petite) valeur proposée antérieurement et comparant à chaque tirage.*

Exercice 31 *Fibonacci*

Reprendre le programme calculant les termes de la suite de Fibonacci. En copier/coller le texte dans une nouvelle fenêtre de Scinotes .

1. Adapter le code pour qu'il demande un entier p (exposant de 10^p) puis qu'il détermine le premier entier n à partir duquel ϕ_n dépasse 10^p
2. Copier/coller le *da Fibonacci code* dans une autre fenêtre . Sachant que la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_n = \frac{\phi_{n+1}}{\phi_n}$ converge vers le nombre d'or $\Phi = \frac{1+\sqrt{5}}{2}$, déterminer le plus petit entier n tel que $|u_n - \Phi| < 10^{-p}$ pour p demandé par une invite.

3.4 Quelques corrigés :

Exercices sur la boucle `for`

```
// toutes les tables de 1 à 10 ////////////////////////////
 disp("Tables de 1 à 10")
 for table=1:1:10
     x=[]
     for i=1:1:10
         x=[x,i*table];
     end
     disp(x)
 end
```

Tableau 3.2 :

Ici le programme crée un vecteur qu'il remplit pour chaque table, affiche quand il a finit puis l'écrase et recommence. L'affichage est amélioré en créant un tableau comme dans le script suivant :

```
//TP3 ////////////////////////////////////////////////
// toutes les tables de 1 à 10  ////////////
 disp("Tables de 1 à 10")
 T=[]
 for table=1:1:10
     x=[];
     for i=1:1:10
         x=[x,i*table];
     end
     T=[T;x];
 end
 disp(T)
```

Tableau 3.3 :

```
//TP3 ////////////////////////////////////////////////
// tables de 1 à 10 - version jolie /////////
 disp("Tables de 1 à 10")
 for t=1:1:10
     c=""
     for i=1:1:10
         d=string(t)+"*"+string(i)+"="+string(t*i)
         c=c+d+"   ";
     end
     disp(c)
     disp("___________")
 end
```

Tableau 3.4 :

```
//TP3 ////////////////////////////////////////////////
 u=0;
 for i=1:1:10
     u=sqrt(u+1);
 end
 disp(u)
```

Tableau 3.5 :

```
// TP3 ////////////////////////////////////////////////
//une suite
 n=input("entrez un entier n : ")
 u=5;x=[u];
 for i=1:1:n
     u=sqrt(u+1);
     x=[x;u];
 end
 disp(x)
```

Tableau 3.6 :

```
// TP3 ////////////////////////////////////////////////
// Suite de Fibonacci
 n=input("donner une valeur de n : ")
 u=1;v=1; //u représentera le un courant et v le terme précédent
 if n>=2 then
     for i=2:1:n // on commence à 2 pour que le premier calcul soit celui de u2
                 // et le dernier soit bien celui de un
         a=u;    //on met en mémoire la valeur courante
         u=u+v;  //on calcule la nouvelle valeur
         v=a;    //le terme précédent prend la valeur
                 //de l'ancienne valeur courante
 //        disp("u"+string(n)+" = "+string(u))
 // décommenter la ligne ci-dessus pour afficher tous les termes de la suite.
     end
     disp(u)
 else
     if n<0 then
         disp("on a dit positif !")
     else
         u=1;    //il est préférable de quand même donner une valeur à u
         disp(u)
     end
 end
```

Tableau 3.7 :

Exercices sur la boucle `while`

```
// TP3 - Lancer de deux dés ////////////////////////////////////////////
// jusqu'à obtention d'un double
//
 de=0    // on fixe des valeurs
 DE=1    // différentes pour entrer dans la boucle
 n=0     // mise à zero du compteur
 while de<>DE
     de=1+floor(6*rand())
     DE=1+floor(6*rand())
     n=n+1
     disp("lancer n°"+string(n)+" : "+string(de)+ " et "+string(DE))
 end
 disp("Vous avez obtenu une paire !")
```

Tableau 3.8 :

```
// TP3 ////////////////////////////////////////////////
// Bond 007 ///////////////////////////////////////////
 x=0
 n=0
 while abs(x)<007
     u=1-2*floor(2*rand())
     x=x+u
     n=n+1
 end
 disp("il a fallu "+string(n)+" bonds")
```

Tableau 3.9 :

```
// TP3 ////////////////////////////////////////////////
// Bond 007 ///////////////////////////////////////////
// avec nombre moyen de bonds nécessaires //////////////
 v=[]
 for i=1:1:10000
     x=0
     n=0
     while abs(x)<007
         u=1-2*floor(2*rand())
         x=x+u
         n=n+1
     end
     v=[v;n]
 end

 m=sum(v)/length(v)
 disp("bonds",m,"en moyenne, il faut :")
```

Tableau 3.10 :

```
// TP3 - devine le nombre ///////   version simple    //////////////
//
 nombre=1+floor(100*rand())
 test=0
 while test<>nombre
     test=input("Devine le nombre ! ")
     if test<nombre then
         disp("trop petit ! essaie encore :-)")
     else
         if test>nombre then
 // le message qui suit ne convient pas si test=nombre donc
 // il faut un else...if
              disp("trop grand ! essaie encore :-)")
         end
     end

 end
 disp("bravo, tu as trouvé !")
```

Tableau 3.11 :

```
// TP3 - devine le nombre ////////////////////////////////////////////////////
// ////////////////////  Version avec mémoire ////////////////////
nombre=1+floor(100*rand())
test=0
max=101;min=0
while test<>nombre
   test=input("Devine le nombre ! ")
   if test<nombre then
       if test<=min then
           disp("on a déjà dit que le nombre était plus grand que "+string(min))
       else
           min=test
       end
       disp("trop petit ! essaie encore :-)")
   else
       if test>nombre then
            if test>=max then
           disp("on a déjà dit que le nombre était plus petit que "+string(max))
       else
           max=test
       end
            disp("trop grand ! essaie encore :-)")
       end
   end

end
disp("bravo, tu as trouvé !")
```

Tableau 3.12 :

```
// TP3 ////////////////////////////////////////////////
// Suite de Fibonacci /////// infini  ////////////////////
 p=input("entrez l''exposant p de 10^p : " )
 u=1;v=1;
 n=0
 while u<10^p
         a=u;
         u=u+v;
         v=a;
         n=n+1;
 //        disp("u"+string(n)+" = "+string(u))
 end
     disp("n = "+string(n))
```

Tableau 3.13 :

TP4 | Matrices & vecteurs

Nous avons déjà vu que, pour Scilab , tout est vecteur, ou plutôt matrice. L'objet de ce TP sera de parcourir les principales commandes relatives au calcul matriciel et vectoriel. Nous en avons déjà vu plusieurs, qui seront reprises dans ce TP.

4.1 Vecteurs

Exemples

Quelques exemples sont plus parlants que des commandes théoriques alors tapez les commandes suivantes **dans la console**. Vous pourrez compléter à droite pour vous souvenir de l'effet de chacune d'elles.

Commençons par quelques exemples avec des vecteurs, pour la plupart déjà rencontrés

```
x=1:5
```

```
x=1:2:10
```

```
y=x'
```

```
x=3:-1:-2
```

```
y=1:-1:9
```

```
z=[1;50;0.2;0]
```

```
z=[1,50,0.2,0]
```

```
z=[1,50,0.2,0]'
```

```
x=[]
x=[x,1]
x=[x,2]
x=[x,3]
for i=4:10
    x=[x,i]
end
```

Reprendre les mêmes commandes avec un `;` (en console)

A retenir :

- on définit un vecteur vide avec `[]`
- on peut définir un vecteur par ses composantes séparées par `,` ou `;` (ligne ou colonne)
- `,` continue la ligne et `;` passe à la ligne suivante (cf matrices)
- un vecteur peut être complété au fur et à mesure avec `u=[u,complement]`
- ceci est bien pratique pour stocker des valeurs en vue de les représenter graphiquement ou y faire référence plus tard.

Remarque avancée (hors programme)

Quand on écrit une boucle `for`, comme `for i=1:10` la variable `i` prend sucessivement la valeur de chaque composante du vecteur `1:10`. Ainsi, si `u` est un vecteur, on peut définir une boucle `for i=u ... instructions ... end` où la variable numérique `i` prendra successivement les valeurs des composantes de `u`.

Encore une commande vectorielle utile

`linspace(a,b,k)` crée un vecteur comportant `k` valeurs (y compris `a` et `b` également réparties dans $[a;b]$. Ainsi, `linspace(a,b,k)` a le même effet que la commande `a:(b-a)/(k-1):b`. En effet k valeurs équiréparties entre a et b définissent $(k-1)$ intervalles.

Tapez en console :

```
linspace(0,10,10)
linspace(1,10,10)
linspace(0,1,10)
1:0.1:10
linspace(0,1,11)
```

Ces quelques exemples montrent à quel point il faut faire attention à la signification de chaque paramètre. La commande n'est pas indispensable à retenir mais peut s'avérer commode pour certaines manipulation. Nous en reparleront dans le TP sur l'intégration.

Exemples de calcul avec les vecteurs

Ici encore de la redite pour rafraîchir la mémoire. Vous pourrez compléter le tableau avec la description de l'action de chaque commande ci-dessous.

```
x=1:9
sum(x)
```

```
prod(x)
```

```
length(x)
```

```
2*x
```

```
x.^2
```

```
exp(x)
```

```
rand(1,10)
u=floor(2*rand(1,10))
v=floor(2*rand(1,10))
u==v
```

Les opérations que l'on fait habituellement avec un nombre, Scilab applique en blocs à chaque composante d'un vecteur.

Exercice 32 : **Table de valeurs**
On utiliser ceci pour obtenir un tableau de valeurs rapide, par exemple pour tracer une courbe. Intéressons-nous à la fonction « sinus ».

attention ! `-2%pi:0.1:2%pi` ne fonctionne pas, pensez aux `*`

- Créer un vecteur x dont les composantes parcourent l'intervalle $[-2\pi; 2\pi]$ à partir de -2π de 0,1 en 0,1 puis un vecteur y contenant les images des composantes de x par la fonction sinus.
- Vous pouvez ensuite afficher avec `plot(x,y)`

Si vous êtes en avance, allez faire un tour du côté de l'aide avec `help plot` *où vous trouverez un tas d'exemples. Mais nous reviendrons plus en détails sur cette commande plus tard.*

Rappel : Pour effacer la fenêtre graphique entre deux essais, tapez `clf()`

4.2 Matrices

Les matrices se définissent essentiellement de la même manière que les vecteurs.

Par exemple, vous pouvez concaténer des vecteurs, ou des matrices :

```
x=1:5
M=[x;x]
M=[M;M]
M=[M,M]
M=[M;2*M]
x=1:10;M=[];
for i=1:10
    M=[M;i*x]
end
```

Si vous voulez utiliser Scilab pour réfléchir à un exercice comportant des matrices, vous pouvez aussi définir les matrice par leurs coefficients. Il suffit de les rentrer ligne par ligne.

Les coefficients d'une même ligne sont séparés par `,` et `;` change de ligne

Par exemple :

$$M=\begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix}$$

```
M=[1,2,3;4,5,6;7,8,9]
```

Cette syntaxe est orientée ligne (dans le sens matrice d'un système). Si vous souhaitez définir la matrice par ses vecteurs colonnes (dans le sens image d'une base), vous pouvez avoir recours à l'astuce suivante :

Si les colonnes sont $\begin{pmatrix}1\\2\\3\end{pmatrix}$, $\begin{pmatrix}2\\4\\6\end{pmatrix}$ et $\begin{pmatrix}0\\1\\0\end{pmatrix}$

```
M=[1,2,3;2,4,6;0,1,0]'
```

Tout est dans le `'` qui renvoie la transposée de la matrice comme nous le rappellerons tout à l'heure.

Exemples de calcul avec les vecteurs

```
M
M'
```

```
rank(M)
```

```
size(M)
```

```
x=diag(M)
N=diag(1:5)
N=diag([1;-1;2;-2;3])
M
diag(diag(M))
```

```
M=[1,2,0;0,2,1;3,5,1]
inv(M)
M^(-1)
M^.(-1)
M*inv(M)
```

```
x=5:7
x'
M*x
M*x'
x'*M
```

```
A=[1,2,3;4,5,6]
B=[1,2;3,4;5,6]
A*B
B*A
x*x'
x'*x
inv(A)
```

```
M
M(1,2)
M(2,1)
M(2,1)=0
```

```
M^2
M.^2
```

```
M+1
M-2
```

```
min(M)
max(M)
```

Toutes les opérations « coefficient par coefficient » vues sur les vecteurs fonctionnent également avec les matrices.

Attention toutefois. On peut définir l'exponentielle de matrice, qui n'est pas la matrice dont les coefficients sont les exponentielles des coefficients de la matrice de départ, une racine carrée de matrice... Vous pouvez tester ces opérateurs avec `exp(M)`, `sqrt(M)`...

Faites bien attention que `M^2` calcule le carré de la matrice, au sens de la multiplication de matrice, alors que `M.^2` calcule le carré de chaque coefficient. De manière générale, le `.` devant une opération opère coefficient par coefficient.

Plus de définitions de matrices

Essayez aussi ces commandes et notez leur effet à côté

```
zeros(4,6)
zeros(6,4)
```

```
ones(4,6)
```

```
eye(4,6)
eye(6,4)
eye(3;3)
```

```
rand(4,6)
rand(4,6)
floor(2*rand(4,6))
floor(10*rand(4,6))
```

Pour vous souvenir de `eye`, pensez à la prononciation anglaise du I qui désigne la matrice identité

Exercice 33
Coder chacune des matrices suivantes dans Scilab.

$$A=\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix},\ B=\begin{pmatrix}1&3&3\\3&-1&3\\3&3&2\end{pmatrix},\ C=\begin{pmatrix}1&2&3\\2&4&6\\3&6&9\end{pmatrix},\ D=\begin{pmatrix}0&0&1&1\\0&0&1&1\\1&1&0&0\\1&1&0&0\end{pmatrix}$$

Exercice 34
Coder en Scilab la matrice

$$N=\begin{pmatrix}0&1&1&1\\0&0&1&1\\0&0&0&1\\0&0&0&0\end{pmatrix}$$ Puis calculer N^2, N^3, N^4,

---

Extraction, sous-matrices

Essayez aussi ces commandes et notez leur effet à côté

```
M=[1:5]'*[3:10]
M(1,:)
M(2,:)
M(:,1)
M(2,:)
M(:,1)=[1:5]'
M(1,:)=[1:8]
```

```
v=[1,2,5]
M(v,:)
M(:,v)
M(v,v)
```

4.3 Résolution de systèmes linéaires

L'étude de problèmes mathématiques amène souvent à résoudre des systèmes d'équations de la forme :

$$\begin{cases} a_{11}x + b_{12}y + c_{13} = b_1 \\ a_{21}x + b_{22}y + c_{23} = b_2 \\ a_{31}x + b_{32}y + c_{33} = b_3 \end{cases}$$

En posant $A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$, $X = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ et $B = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$,
le système peut s'écrire $AX = B$

Si A est inversible, on peut le résoudre en multipliant à gauche par l'inverse A^{-1} de A.
On trouve $X = A^{-1}B$ qui se code en Scilab `x=inv(A)*b`

Cette méthode suffira dans la plupart des cas mais la méthode « propre » à Scilab utilise la syntaxe `x=A\b`, appelée division à gauche.
C'est « comme si » on multipliait à gauche par A^{-1}. L'intérêt de cette syntaxe est qu'elle renvoie une solution (la meilleure au sens des moindre carrés) même si la matrice A n'est pas inversible, et en particulier si elle n'est pas carrée.

Néanmoins, dans les cas où la résolution théorique aboutirait à une infinité de solutions, Scilab ne les détermine pas toutes

---

Exercice 35 *Un système...*

Utiliser Scilab pour résoudre le système suivant :

$$\begin{cases} 3x - 2y + 5z = 27 \\ x - 3y + z = 11 \\ 2x + 3y + 7z = 24 \end{cases}$$

Exercice 36

1. Quelle est la commande Scilab qui permet de savoir si une matrice est inversible ?

2. Créer une dans SCINOTES une matrice 3×3 alétoire dont les coefficients peuvent valoir 0 ou 1 avec la même probabilité. Tester si elle est inversible.

3. Écrire un programme qui détermine une valeur approchée de la probabilité qu'une matrice dont les coefficients suivent la loi $\mathcal{U}(\{0;1\})$ uniforme sur $\{0;1\}$ (c'est à dire qui vaut 0 ou 1 avec la même probabilité) soit inversible.

4. Si vous êtes en avance, modifiez le programme pour tester si la probabilité est la même avec une matrice 4×4, puis avec une matrice 3×3 dont les coefficients suivent $\mathcal{U}(\llbracket 0;9 \rrbracket)$

4.4 Approfondissements

Échantillonnage, tests multiples

On peut envisager la commande `rand(n,m)` comme un moyen de générer une matrice aléatoire dont chaque composante suit une loi uniforme sur $[0;1]$.
On peut aussi voir cette matrice comme m échantillons de n répétitions d'une variable aléatoire. Essayez la commande `floor(10*rand(10,6))`

Maintenant, essayez `floor(10*rand(1,10)) ;`, puis `x==6`, `x==1`, `x>3`, `x<=6`...
Un test appliqué à un vecteur renvoie un vecteur correspondant aux résultats des tests de chaque coordonnée.
Pour compter le nombre de résultats égaux à 1, on peut définir un vecteur `v=[x==1]` puis en ajouter les composantes avec `sum(v)`. Les comptent pour 1 (comme en calcul binaire) et compte pour 0.

Exercice 37
Créer un vecteur x et un vecteur y comportant chacun 100 composantes aléatoires suivant chacune la loi $\mathcal{U}(\llbracket 0;9 \rrbracket)$
Ensuite écrivez une commande qui compte le nombre de composantes de x supérieures ou égales à 5. Comptez ensuite les composantes de x et y qui sont égales (c.a.d le nombre d'indices i tels que $x_i = y_i$). Enfin, le nombre de composantes de x strictement plus petites que la composante de y correspondante.

Exercice 38 On rappelle que la loi binômiale $\mathcal{B}(n;p)$ compte le nombre de succès obtenus en répétant n fois une épreuve à deux issues dont la probabilité de succès est p.

1. Écrire dans Scilab une commande qui crée un vecteur dont les composantes simulent la répétition de 10 épreuves à deux issues équiprobables. Compléter cette commande pour compter le nombre de succès
2. Écrire en Scilab un programme qui répète 100 fois ce qui précède et en conserve les résultats successifs dans un vecteur. Afficher le résultat avec `histplot(10,x)`

4.5 Un peu de culture G : Alan Turing et la pomme d'Apple

Un texte extrait du site tatoufaux.com

En décembre 2013, la reine d'Angleterre accordait à titre posthume la grâce royale à un mathématicien anglais, près de 60 ans après son décès. Condamné pour homosexualité dans les années 50, cette réhabilitation très réclamée reconnaissait implicitement l'absurdité de cette loi indigne et d'autant plus cruelle qu'elle avait probablement mené le « coupable » au suicide, alors même qu'on estime qu'il a personnellement permis, grâce à son travail, que soient épargnées des milliers de vies humaines.

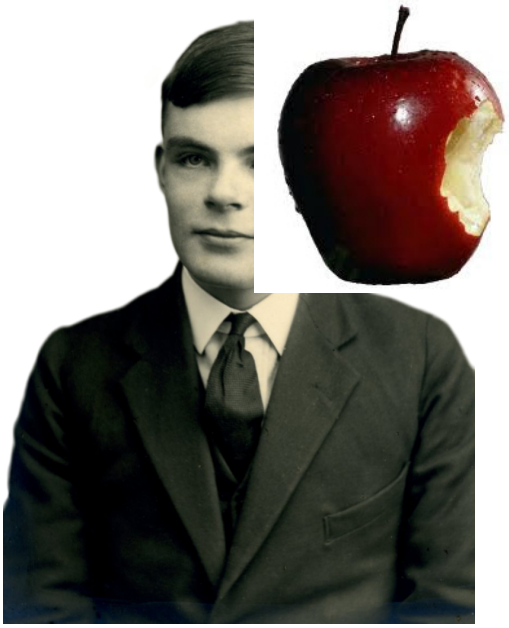


Figure 4.1 : Alan Turing

Au-delà de sa propre postérité, le nom d'Alan Turing est également associé à une société qui, plus que n'importe quelle autre, suscite passions et controverses : Apple.

Et plus particulièrement son logo, reconnaissable entre mille et qui représente une pomme croquée. Apple n'est pas qu'une marque d'ordinateur, c'est aussi un phénomène de société qui défie parfois toute logique. Le mythe créé autour de son créateur emblématique, idolâtré comme une rock-star par ses fans, ou les queues interminables devant les magasins des semaines avant la sortie d'un nouveau modèle de téléphone ou d'ordinateurs ne sont que quelques exemples de la place qu'a pris cette marque dans la culture populaire.

Rien de surprenant donc à ce que l'origine de ce logo fasse l'objet de théories plus ou moins exotiques, mais l'une d'elle particulièrement alambiquée reste relativement tenace : celle de l'hommage à Alan Turing qui a d'ailleurs au moins le mérite d'être poétique.

Revenons un petit peu en arrière. Nous sommes en 1954, soit une vingtaine d'années avant la création d'Apple. Le génial mathématicien anglais décède dans des conditions so romantic ! Condamné deux ans auparavant à la castration chimique après que son homosexualité ait été révélée (un délit en Grande-Bretagne jusqu'en 1967), Alan Turing meurt en croquant une pomme empoisonnée au cyanure[1].

Une fin tragique pour ce scientifique de haut vol qui a notamment mis au point les algorithmes permettant de décoder les messages de l'armée allemande, et en particuliers ceux réputés inviolables de la fameuse machine Enigma, jouant ainsi un rôle déterminant dans la victoire des Al-

4.1 Il n'existe pas de preuves formelles car la pomme retrouvée croquée à côté du corps n'a jamais été analysée mais Alan Turing était un inconditionnel de Blanche Neige et cette version de sa mort est très largement considérée comme la plus plausible.

liés.

Une pomme croquée, un mathématicien adulé, une fin de conte de fées, il n'en fallait pas plus pour que l'on voit dans cette pomme croquée de la firme Apple, un hommage des pères du Macintosh à l'un des pères de l'informatique moderne. Pour autant que l'on sache, il n'en est rien. Et cette explication est d'ailleurs relativement fantaisiste. L'hommage à Alan Turing pourrait se concevoir mais utiliser un signe de mort et d'empoisonnement pour une firme fabriquant des ordinateurs serait assez curieux. Le logo de 1976 est composé de six bandes de couleurs pouvant facilement être confondues avec le drapeau arc-en-ciel, mais en 1976 ces couleurs n'évoquaient pas comme aujourd'hui la communauté gay et d'ailleurs aucun des fondateurs d'Apple n'était gay, ce qui exclut un clin d'œil communautaire. D'autant que si l'on regarde bien, ce ne sont pas les mêmes couleurs.

À moins que l'origine du logo n'ait pas d'histoire (Steve Job mangeait des pommes, il décide d'appeler sa boite Apple, et rien de plus), le plus probable reste que la fameuse pomme soit celle de Newton, autre génie des sciences dont on dit qu'il aurait compris les lois de la gravité en en prenant une sur le coin du crâne. Ce rapport à Newton est facile à vérifier, le premier logo d'Apple computer compagnie le représentait adossé à un arbre, une pomme au-dessus de la tête. Difficile de faire moins allégorique.

Mais alors pourquoi une pomme croquée ?

Selon toute vraisemblance et de l'avis même de Rob Janoff, le graphiste qui a créé ce logo, c'était tout simplement pour éviter qu'elle soit confondue avec une tomate. Décevant, certes, mais probablement authentique.

Compléments au texte (inspirés de Wikipedia)

Après la guerre, il travaille sur un des tout premiers ordinateurs, puis contribue de manière provocatrice au débat déjà houleux à cette période sur la capacité des machines à penser, en établissant le test de Turing. Turing a mené également des réflexions fondamentales réunissant la science et la philosophie. Dans l'article « Computing Machinery and Intelligence » (Mind, octobre 1950), Turing explore le problème de l'intelligence artificielle et propose une expérience maintenant connue sous le nom de test de Turing, où il tente de définir un standard permettant de qualifier une machine de « consciente » ; Turing fait le « pari que d'ici cinquante ans, il n'y aura plus moyen de distinguer les réponses données par un homme ou un ordinateur, et ce sur n'importe quel

sujet. Il est en passe de gagner son pari quand on sait qu'en 2014, un ordinateur a réussi à se faire passer pour un adolescent dans un concours de discussion entre humains et intelligences artificielles. Vers la fin de sa vie, il s'intéresse à des modèles de morphogenèse du vivant conduisant aux « structures de Turing ».
Sur le même thème, vous pourrez (re)voir le biopic *Imitation Game* sur Alan Turing et l'extrapolation de nos réseaux sociaux dans le film *Her*.

4.6 Quelques corrigés

Pour l'exercice 2, on peut définir les matrices avec :

- `A=ones(3,3)-eye(3,3)`
- ou `A=ones(3,3);diag(A)=[0,0,0];A`
- `B=3*ones(3,3);B=B+(diag([1,-1,2]-3));B`
- attention, comparer à `B=3*ones(3,3);B=B+(diag([1,-1,2])-3);B`
- `u=[1,2,3];C=[u;2*u;3*u]`
- `A=zeros(2,2);B=ones(2,2);D=[A,B;B,A]`

Pour l'exercice 3,
`N=[0,1,1,1;0,0,1,1;0,0,0,1;0,0,0,0],N^2,N^3,N^4`

exercice 5

Essayer d'inverser une matrice ne permet pas de savoir si celle-ci est inversible. En effet, si elle ne l'est pas, vous obtiendrez un message d'erreur. Une matrice carrée est inversible ssi son rang est égal à son ordre.

Ici, la commande `rank(M)==3` renvoie donc %T si elle est inversible et %F si elle ne l'est pas.
`M=floor(2*rand(3,3));rank(M)==3` crée une matrice aléatoire et teste son inversibilité. Changez `;` pour `,` si vous voulez voir la matrice.

On peut estimer la probabilité qu'une telle matrice soit inversible avec le programme suivant :

```
// Estimation de la probabilité
// qu'une matrice aléatoire soit inversible
n=3 // dimension de la matrice
u=[]
N=input("Entrez un nombre entier : ")
for i=1:N
    M=floor(2*rand(n,n))
    u=[u,rank(M)==n] // on ajoute le résultat
                            //  du test d'inversibilité
end
p=sum(u)/length(u) // évalue la probabilité
disp(p,"probabilité que la matrice soit inversible : ")
```

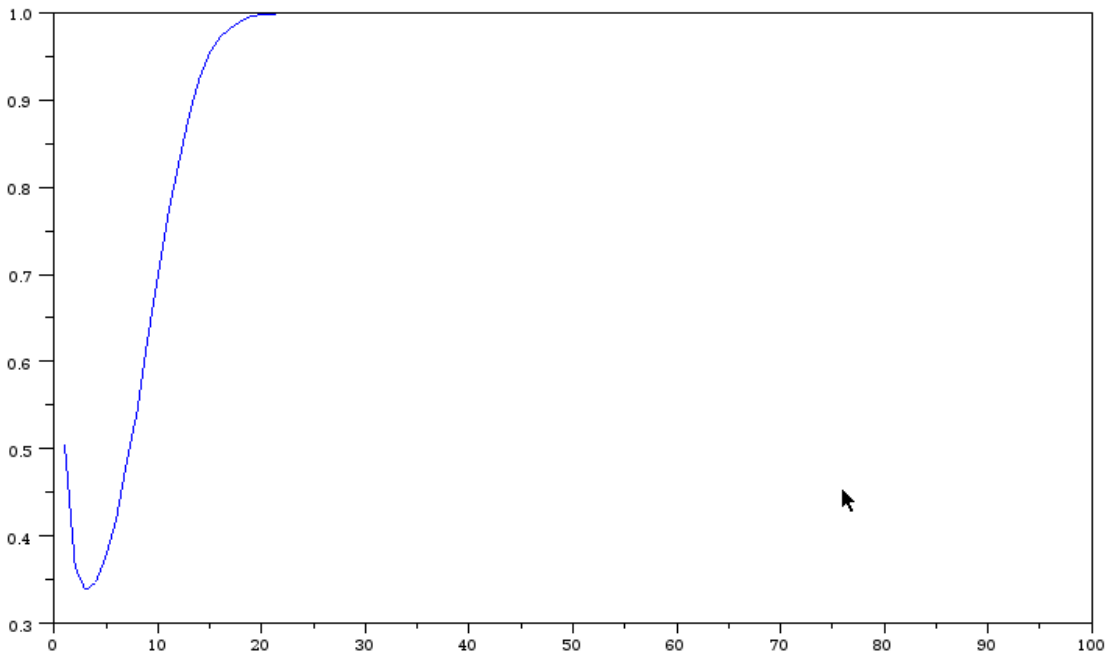
Le code ci-dessous est un peu plus général que ce qui était demandé et permet de changer facilement la dimension de la matrice. On peut remarquer avec $N = 10000$ itérations, que pour $n = 3$, la probabilité que la matrice soir inversible est d'environ $0,3$. Elle est approximativement de $0,7$ pour $n = 10$ et et 1 pour $n = 100$.
On pourrait calculer « à la main » de manière exacte, la probabilité que la matrice soit inversible pour les petites valeurs de n

prolongement

Le code qui représente graphiquement une estimation de la loi de la v.a.r X qui à un entier n non nul associe la probabilité qu'une matrice dont les coefficients suivent une loi uniforme sur $\{0; 1\}$ soit inversible.

```
clf()
x=[]
for n=1:100 // 100 permet un temps de calcul raisonnable
    u=[]
    for i=1:100
            M=floor(2*rand(n,n))
            u=[u,rank(M)==n] // on ajoute le résultat
                             //  du test d'inversibilité
    end
    x=[x,sum(u)/length(u)] // évalue la probabilité
end
plot(x)
```

Avec $n = 100$, le résultat est très approximatif. Si vous avez le temps (plusieurs minutes), essayez avec $n = 10000$ et après une longue pause café, vous obtiendrez la courbe ci-dessous, la probabilité qu'une matrice aléatoire soit inversible tend rapidement vers 1avec la taille de celle-ci.



TP5 | Récursivité, suites, récurrences *et caetera...*

Les exercices sur les suites sont légion et particulièrement propices aux exercices d'algorithmique. L'objet de ce TP sera de dresser un panorama des exercices d'algorithmique en lien avec les suites. Beaucoup de redite au programme mais je recommande chaudement la maîtrise de ces codes pour les écrits à venir. Nous ferons certainement un autre TP sur les suites plus tard en révisions.

Qui dit récurrence dit boucle itérative, les notions sont parallèles : l'une mathématique, l'autre algorithmique. Nous allons donc énormément utiliser ces notions après en avoir vu deux nouvelles :

5.1 Fonctions Scilab

Scilab est très riche en commandes et fonctions diverses. Toutefois, on peut programmer des fonctions personnelles, pour un usage spécifique et l'une des questions d'algorithmique les plus fréquentes est : « Écrire une fonction Scilab qui fait ceci ou cela. »

Voyons la mise en œuvre sur un exemple. Tapez dans Scinotes :

```
function [y] = puissance(x,n)
// calcule la puissance n-ieme
  y = 1 // valeur de y^0, et valeur initiale si n>0
  if n > 0 then
    for i = 1:n
      y = y*x // on multiplie n fois x par lui-même
    end
  end
endfunction
```

Puis enregistrez sous le nom `puissance`. Quand on exécute le code, il se ne se passe ... rien.
Rien de visible en tout cas. En effet, l'exécution de ce code a ajouté à la liste des fonctions une nouvelle. Si maintenant vous tapez en console :
`--> puissance(2,3)`
vous verrez que votre nouvelle fonction est disponible.

Il est possible que Scinote propose l'extension `.sci` au lieu du `.sce` habituel. C'est normal. Il s'agit de l'extension propre aux extensions de scilab (fichiers **i**nternes quand `.sce` est propre aux programmes (**e**xternes).

La syntaxe générale est la suivante :

```
function [<vecteurs sorties>] = nom_fonction(<entrées>)
  <instructions>
endfunction
```

Les [ ] sont facultatifs pour une fonction dont la sortie est une variable unique. Les crochets sont là pour rappeler que la sortie est un vecteur.

A retenir sur les fonctions

- on commence par `function` et on termine par `endfunction`
- Quand on lance une fonction, on la charge en mémoire mais aucun calcul n'est fait
- Pour utiliser une fonction, on l'appelle comme n'importe quelle autre commande de Scilab .
- Si la sortie comporte plusieurs nombres, plusieurs options sont possibles. Soit on programme la fonction avec `function u=fonction(x,y,z)` et u est évalué en tant que vecteur dans la fonction. Dans ce cas, la fonction renvoie ledit vecteur. Deuxième option, on implémente sous la forme `function [a,b]=fonction(x,y,z)` et, par défaut, la fonction renvoie la première variable a. Pour obtenir les deux, il faut appeler la fonction avec une syntaxe du type : `[u,v]=fonction(3,5)`. u contiendra alors la première variable et v la seconde.

Exercice 39 Écrire une fonction Scilab qui prend en entrée un réel x et lui associe e^{-x} si $x \geqslant 0$ et 0 dans tous les autres cas.

Exercice 40 Écrire une fonction Scilab qui prend en entrée un réel x et lui associe $x+1$ si $x \in [-1;0]$, $-x+1$ si $x \in [0;1]$ et 0 dans tous les autres cas.

Exercice 41

1. Écrire une fonction `sinxsurx` qui, à un réel x non nul associe $\dfrac{\sin x}{x}$ et associe 1 à 0. Testez votre nouvelle fonction.
2. *Facultatif : si vous avez trouvé sans hésitation alors que tout le monde cherche encore, sauriez prouver proprement que cette fonction est continue en 0*
3. Afficher le graphe de la fonction avec `x=[-10:0.1:10],plot(x,sinxsurx(x))`

En cas de problème avec le module graphique, essayez la commande `usecanvas(%T)`

L'affichage du graphe nécessite que la fonction puisse travailler avec des vecteurs. Or, si vous avez codé comme je l'imagine, la division par un vecteur n'est pas définie.
La division par un vecteur, terme à terme s'obtient avec `./` au lieu de `/` qui concerne les nombres uniquement.
Reste à régler le problème de la division par zéro. Pour cela, il suffit de remplacer tous les zéros de x par ε, le `%eps` qui désigne le « zéro machine ». Par exemple,

`x=max(abs(x),%eps)` puis `y=sin(x)./x)` dans la fonction font l'affaire.
Prenez un temps pour vous en convaincre et voir pourquoi cette valeur absolue ne change rien aux valeurs de la fonction.

Tracez ensuite la courbe de la fonction avec `x=-5:0.1:5;plot(x,sinxsurx(x))`

5.2 Récursivité

Définition

définition : Une procédure récursive est une procédure récursive...

Plus sérieusement, on parle de récursivité quand une fonction, dans sa définition, **s'appelle elle-même**. Cette notion algorithmique est très liée à la notion de *récurrence* en mathématiques.

Observons les deux exemples ci-dessous :

```
function [f]=fact(n)
// la factorielle en recursif
   if n <= 1 then
      f = 1
   else
      f = n*fact(n-1)
   end
```

```
function [u]=fib(n)
   // calcul du n ieme terme
   // de la suite de Fibonnaci :
   // fib(0) = 1, fib(1) = 1,
   // fib(n+2) = fib(n+1) + fib(n)
   if n <= 1 then
      u = 1
   else
      u = fib(n-1) + fib(n-2)
   end
```

Le premier exemple est classique et à retenir. Le second est un exemple classique de *mauvaise* utilisation de la récursivité. En effet, l'itération classique (cf le TP sur les boucles) est plus efficace, au sens qu'elle demande moins de calculs à la machine (les termes inférieurs de la suite sont calculés plusieurs fois).
Toutefois, la récursivité permet une implémentation simple et surtout, elle fonctionne.
Aussi, si ça vous parle, n'hésitez pas à en faire usage car l'optimisation du code ne fait pas partie des compétences requises en ECS. Nous ferons donc comme si les calculs étaient instantanés et utiliserons la récursivité d'une manière parfois peu recommandable mais ... vous ne postulez pas à une école d'informatique !

On pourrait aussi le vérifier avec `tic()` et `toc()`

Exemples

Une idée que l'on retrouve dans toutes les récursivités est que, pour connaître le terme courant, on a besoin du précédent, qui nécessite le précédent, qui... Tous ces précédents forment ce que l'on appelle la « pile » et il est important que la fonction récursive contienne la valeur explicite du ou des premier(s) terme(s), le bas de la pile.

Une fonction récursive contiendra donc :

- ⋆ INITIALISATION : un **test** pour déterminer si le cas demandé correspond à la valeur initiale
- ⋆ HÉRÉDITÉ : un **appel à elle-même** au(x) rang(x) inférieurs $n-1$, éventuellement $n-2$...

5.3 Exercices

Exercice 42 : *You want to remember this one*
Pour chacune des suites suivantes, écrire une fonction récursive qui prend un entier n en entrée et renvoie u_n.

a. $u_0 = 1$ et $u_{n+1} = \sqrt{u_n} + 5$
b. $u_1 = 1$ et $u_{n+1} = u_n + \dfrac{1}{u_n}$
c. $u_0 = 0$ et $u_{n+1} = -\frac{1}{2}u_n - 3$

Pour chaque suite, faire quelques tests afin de conjecturer la convergence éventuelle et, le cas échéant, une valeur aprochée de la limite.

Travail à la maison : Déterminer, le cas échéant la valeur exacte l de la limite et écrire une procédure qui demande un entier p et renvoie le premier indice n tel que $|u_n - l| < 10^{-p}$. Réfléchissez à la pertinence ou non de faire usage de la fonction récursive écrite avant (même si ce n'est pas une compétence exigible).

Exercice 43 :
Écrire une fonction SCILAB récursive qui demande un nombre n en entrée et renvoie la valeur de u_n pour la suite definie par :

$$\begin{cases} u_0 = 0 \\ u_1 = 1 \\ u_{n+1} = -3u_n + 2u_{n-1} \end{cases}$$

Si vous êtes en avance, écrivez un programme itératif qui fait le même travail.

Remarquer par exemple que $V_{n+1} = MV_n$ où $V_n = \begin{pmatrix} u_{n+1} \\ u_n \end{pmatrix}$ et $M = \begin{pmatrix} -3 & 2 \\ 1 & 0 \end{pmatrix}$.

Exercice 44 : **Algorithmes de calcul des coefficients du binôme**
Le but de cet exercice est d'écrire une fonction `binome` qui prend en entrée deux entier n et p et renvoie $\binom{n}{p}$.
Nous allons en fait en voir trois différents.

1. a. démontrer la formule $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$
   b. utiliser cette relation pour écrire une fonction récursive de calcul des $\binom{n}{k}$
2. a. démontrer la fomule de Pascal : $\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}$
   b. utiliser cette formule pour écrire une fonction itérative de calcul des coefficients du binôme.

Vous pourrez démarrer avec le vecteur `u=[1,1]` et passer à l'étape suivante avec la scilab*esque* commande : `u=[u,0]+[0,u]` que vous pourrez tester en console pour ensuite l'implémenter dans une fonction.

Si vous avez de l'avance, pour réviser les boucles, écrivez un programme ou une fonction qui demande en entrée un nombre n et renvoie une matrice $M = (m_{ij}) \in \mathcal{M}_{n,n+1}(\mathbb{Z})$ telle que $\forall i, j \in [\![1; n]\!], m_{ij} = \binom{i}{j-1}$

Par exemple, pour $n = 5$, on obtiendrait la matrice :

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

Pensez à ce que vous feriez par exemple dans un tableur, et traduisez le en code, en faisant attention au nombre de départ et de fin dans les boucles !

On pourra commencer par créer une matrice de zéros aux bonnes dimensions, remplir la première colonne de 1 et compléter les coefficients de la matrice ligne par ligne grâce à la formule de Pascal. (cf triangle de Pascal)

Exercice 45 : Un classique !
On s'intéresse à la suite $(u_n)_n$ de premier terme $u_0 = 100$ et vérifiant.

$$\forall n \in \mathbb{N}, u_{n+1} = \sqrt{6 + \sqrt{3u_n}}$$

.
Cette suite converge.

- ⋆ Écrivez une fonction `fonc` simulant $x \mapsto \sqrt{6 + \sqrt{3x}}$
- ⋆ Écrivez une fonction `suite` qui calcule u_n pour tout n.
- ⋆ faites vous une idée de la limite puis écrivez un programme qui détermine le premier entier n tel que $|u_n - l| < 10^{-4}$

Exercice 46 ⋆⋆ : La suite de Syracuse est une suite d'entiers définie par un premier entier $u_0 = N \in \mathbb{N}^\star$ et la relation de récurrence

$$u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{est impair} \end{cases}$$

La conjecture de Syracuse affirme que, quel que soit le choix de u_0, il existe un rang p tel que $u_p = 1$. Le premier entier p vérifiant $u_p = 1$ est parfois appelé « longueur de vol »

1. Écrire un programme Scilab qui demande un entier N en entrée, et renvoie le plus petit entier p_N tel que $u_{p_N} = 1$ pour la suite de Syracuse de premier terme $u_0 = N$.
   On pourra utiliser une fonction `syr` *telle que* $u_{n+1} = syr(u_n)$ *puis une fonction* `vol` *qui détermine la longueur de vol pour un entier* N *donné en entrée.*
2. Adapter le programme précédent (que vous pourrez enregistrer sous un autre nom) pour qu'il renvoie un vecteur $x = (p_1, ..., p_{500})$ dont la k-ième coordonnée p_k contient le rang du premier terme égal à 1 de la suite de Syracuse de premier terme $u_0 = k$
3. En déduire la plus grande longueur de vol, puis la moyenne des longueurs de vol sur les entiers inférieurs à 500.

Figure 5.1 : Le nom de la suite vient de l'Université de Syracuse

5.4 Culture G : L'autoréférence

"la poésie dit ce qu'elle dit en le disant ", Jacques Roubaud

Figure 5.2: Les surréalistes étaient amateurs d'autoréférence

L'autoréférence ne désigne pas que la pratique de certains auteurs qui placent leur bibliographie en... bibliographie.

On la retrouve en mathématiques, musique, arts, littérature... et elle a de tous temps exercé une puissante fascination, Narcisse ne le démentirait pas. Universelle, archétypale, elle est présente dans les motifs traditionnels Shipibos ou Hopi d'Amérique du sud, dans la musique Africaine, dans les mandalas d'Inde, chez Zhuāng Zǐ qui, dans la Chine du IV^{me} siècle, qui rêve qu'il est un papillon, qui rêve qu'il est un être humain... qui ne sait plus qui il est.

En art, la symbolique du cercle et de la spirale l'illustrent. L'œuvre de M.C Escher y rend un impressionnant hommage quand d'autres en jouent de façon moins ostentatoire.

Pochette de l'album *Ummagumma*, de Pink Floyd

En musique, le larsen de Jimi Hendrix est un son autoréférent produit par l'enregistrement de lui-même, reproduit déformé réenregistré, ... jusqu'à saturation. J.S. Bach use de l'autoréférence dans la composition de ses fugues qui se déroulent à l'infini, sans parler de la musique comtemporaine de Philip Glass ou Steve Reich.

logo autoréférent de la Vache qui rit

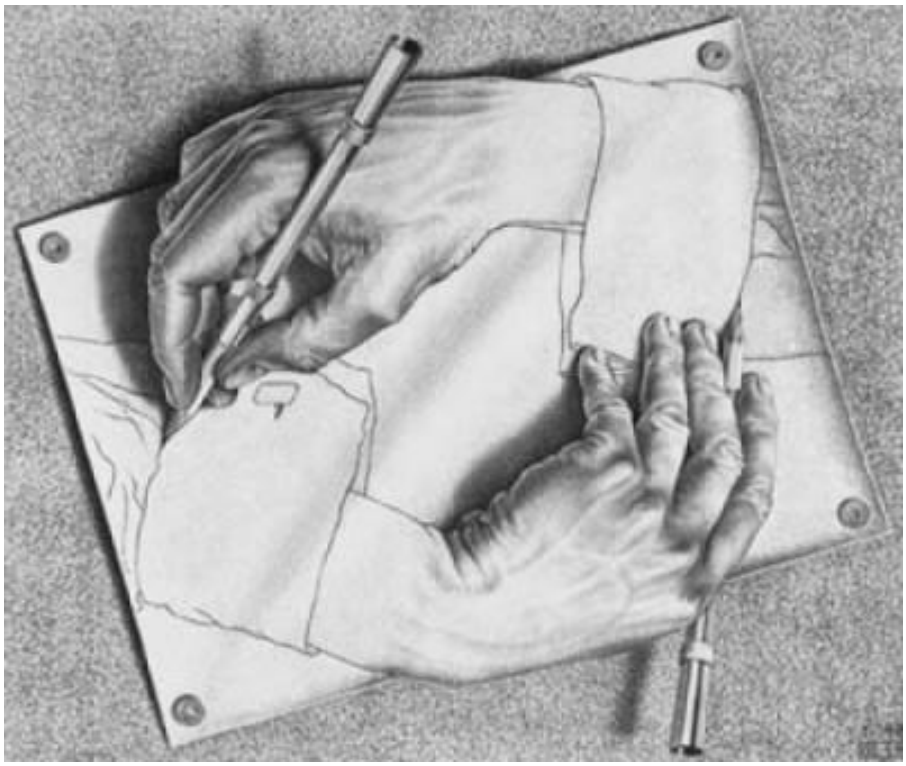
La mise en abîme est un procédé que l'on retrouve chez de nombreux auteurs de littérature comme Milan Kundera, Georges Luis Borges, Shakespeare, Pérec... ou dans des œuvres cinematographiques comme *La nuit américaine, Inception, ...* Sartre dans *L'Être et le néant*, disserte sur la capacité de l'homme à être conscient de lui-même et de ses actes puis de la capacité prendre conscience du fait d'être conscient. Philo toujours avec Wittgenstein qui s'interroge :« Ce livre ne sera peut-être compris que par qui aura déjà pensé lui-même les pensées qui s'y trouvent exprimées ».

En Mathématiques enfin, les fractales font la part belle à l'autoréférence ainsi que le théorème d'incomplétude de Kurt Gödel qui démontre que quel que soit le système d'axiomes choisi au départ, on peut formuler une proposition indécidable (on ne peut prouver sa vérité ou fausseté).
L'autoréférence est à l'origine de

maints paradoxes qui font les délices de Geluck et de son *Chat*. Et vous, sauriez-vous dire si la phrase « Cette phrase est vraie » est vraie ou fausse...

Pour prolonger le thème :

- un article de JP Delahaye paru dans *Pour la Science* : **http ://www.ecriture-art.com/autoreference.pdf**
- *Gödel, Escher, Bach, les brins d'une guirlande éternelle*, Douglas Hofstadter.
- œuvre de M.C Escher sur **http ://www.mcescher.com/**
- Wikipédia, **http ://fr.wikipedia.org/wiki/Mise_en_abyme**



il maestro, M.C. Escher

5.5 Quelques corrigés :

Fonctions

Exercice 2

```
function [y]=fonction(x)
    y=max(-abs(x)+1,0)
endfunction
```

Pour que la fonction reste compatible avec un traitement vectoriel donc, par exemple avec une utilisation graphique via
`x=[-5:0.1:5];plot(x,fonction(x))`
la syntaxe proposée ici est préférable à un test qui, appliqué à un vecteur ne sera jamais vérifié.
Retenir l'usage de `max()` ou `min()` pour ce type d'usage vectoriel.

```
// sinxsurx ///////
// version naive
// ne fonctionne qu avec des nombres
function [y]=sinxsurx(x)
if x<>0 then
     y=sin(x)/x
else
     y=1
end
endfunction
```

```
// sinxsurx ///////
// version vecteurs
function [y] = test(x,n)
x=max(abs(x),%eps)
// %eps est suffisamment petit
// pour ne pas changer la valeur approchée
// mais surtout %eps est non nul !
y=sin(x)./x // avec ./ pour diviser terme à terme
endfunction
```

Exercice 4

```
function [u]=suite1(n)
    if n==0 then
        u=1
    else
        u=sqrt(suite1(n-1))+5
    end
endfunction
```

```
function [u]=suite2(n)
    if n==1 then
// Attention la suite commence pour n=1
        u=1
    else
        u=suite2(n-1) //on "stocke"
        // le résultat pour éviter deux appels
        u=u+1/u    //car on en a besoin deux fois
    end
endfunction
```

```
function [u]=suite3(n)
    if n==0 then
        u=0
    else
        u=-0.5*suite3(n-1)-3
    end
endfunction
```

Rappelez vous d'éviter d'appeler deux fois la procédure dans une récursivité. On appelle une fois, on stocke la valeur que l'on utilise ensuite à notre gré.

Exercice 5

```
// suite récurrente linéaire d'ordre 2 ///////
// version récursive ///////////////
function [u]=rec_lin(n) // version récursive
    if n<=1 then
        u=n // astuce pour cette suite uniquement
    else
        u=-3*reclin(n-1)+2*reclin(n-2)
    end
endfunction
```

Bien que peu performante d'un point de vue complexité, cet algorithme a le mérite d'être très simple et intelligible. Plus performante, la version itérative ci-dessous qui ne réalise qu'une fois chaque calcul.

```
// suite récurrente linéaire d'ordre 2 ////
// version itérative ////////////
function [u]=rec_lin_it(n) // version itérative
    M=[-3,2;1,0]
    U=[1;0] // attention à bien créer une colonne
    for i=1:n
        U=M*U
    end
    u=U(2)
endfunction
```

Exercice 6 : Coefficients du binôme

```
// calcul des coefficients du binôme /////
// version récursive ////////////////////
function [C]=cbinom(n,k)
    if k=0 then
        C=1
    else
        C=(n/k)*cbinom(n-1,k-1)
    end
endfunction
```

J'apprécie l'élégance de la version itérative suivante, bien qu'elle comporte quelques pièges à retenir.

```
// calcul des coefficients du binôme //////
// version itérative //////////////
function [C]=cbinomit(n,k)
    u=[1,1] // lors de l'initialisation n=1
    for i=2:n // donc on commence à 2 ici
        u=[0,u]+[u,0] // peu visible
     // mais il s'agit bien dela formule de Pascal
    end
    C=u(k+1) // attention, le premier coefficient
    //correspond à 0
endfunction
```

Enfin, mais c'est vraiment pour s'entraîner sur les boucles, un algorithme d'affichage du triangle de Pascal dans une matrice.

```
n=input("entrez un nombre n : ")
M=zeros(n,n+1)
M(:,1)=1 // une première colonne de 1
M(1,2)=1 // le deuxième 1 pour
    //commencer le triangle de Pascal
for i=2:n // on va remplir chaque ligne
    for j=2:(i+1) // pas besoin d'aller
    // au bout de la ligne
        M(i,j)=M(i-1,j-1)+M(i-1,j)
    end
end
disp(M)
```

Exercice 8 : Suite de Syracuse

```
// syracuse /////////
function [y]=syr(x)
if modulo(x,2)==0 then // commande la plus simple
        //  pour savoir si un nombre est pair
// on peut aussi utiliser x-2*floor(x/2)==0
// qui n'est vrai que si x est pair
     y=x/2
else
     y=3*x+1
end
endfunction

u=input("u_0= ")
k=0,
while u<>1
   k=k+1
   u=syr(u)
end
disp(u,"longueur de vol  : ")
```

```
// syracuse 2 /////////
// on suppose implémentée la fonction syr
L=[]
for i=1:500
   k=0
   u=i
   while u<>1
      k=k+1
      u=syr(u)
   end
   L=[L,k]
end
disp(max(L), "la plus grande longueur de vol est  : ")
disp(sum(L)/length(L), "La longueur de vol moyenne est  : ")
```

TP6 | Résolution approchée d'équations

Les algorithmes classiques que nous verrons ici sont :

6.1 Introduction

Avertissement : Ce TP comporte un peu plus d'éléments théoriques que les autres, ceux-ci pourront être lus à tête reposée ultérieurement. Les exercices comportent certaines questions purement théoriques qui ne nécessitent donc aucun code.

La résolution d'équation est un thème central des mathématiques. Si l'on connait la résolution des équations polynomiale du premier, ou deuxième degré (pour lesquelles nous avons déjà écrit des algorithmes de résolution exacte).

Le programme d'ECS1 mentionne le *calcul approché de la racine d'une équation du type* $f(x) = 0$ dans la liste de savoir-faire exigibles en première année. *On utilisera différentes méthodes dont certaines résulteront d'une étude mathématique (suites récurrentes, encadrements, dichotomie).*

rappel : en Scilab , tout est approché, mais le même algorithme dans un langage de calcul formel donnerait une solution exacte.

Nous allons voir dans ce TP différents algorithmes de résolution approchée de l'équation $f(x) = 0$, où f est une fonction continue sur un intervalle $[a; b]$ et pour laquelle on sait que $f(a)$ et $f(b)$ sont de signe contraires.

Si au concours la partie algorithmique portait sur le thème de la résolution d'équation, celle-ci s'insérerait dans un problème plus général sur les suites et la résolution d'équation, nous en profiterons donc pour rappeler quelques résultats théoriques, que vous aurez peut-être besoin de replacer dans les questions précédentes.

Exercice 47

1. Pourquoi $f(x) = 0$ et pas les autres équations ?
2. Pourquoi les hypothèses ci-dessus permettent-elles de conclure l'existence d'une solution α à l'équation $f(x) = 0$. Énoncez le théorème.
3. Quelle hypothèse supplémentaire permettrait d'assurer l'unicité de la solution ?
4. Quel test (calcul) permet de vérifier que $f(a)$ et $f(b)$ sont de signe contraires.

6.2 Méthode de dichotomie

Normalement déjà abordée au lycée, c'est la méthode classique.

Principe

Initialisation : Connaissant deux valeurs a_0 et b_0 dont les images ont des signes contraires, on sait donc que f s'annule entre ces deux valeurs en un réel x_0.

Itération : Supposons connus a_n et b_n tels que $f(a_n)f(b_n) < 0$, c'est à dire encadrant la solution cherchée. On calcule alors $c_n = \frac{a_n+b_n}{2}$ et on en calcule l'image.
Si $f(a_n)$ et $f(c_n)$ ont même signe, c'est que la solution se trouve entre c_n et b_n. On pose alors $a_{n+1} = c_n$ et $b_{n+1} = b_n$.
Dans le cas contraire, la solution se trouve entre a_n et c_n. On pose alors $a_{n+1} = a_n$ et $b_{n+1} = c_n$.

Arrêt : On s'arrête quand on a atteint la précision souhaitée, c'est à dire quand $b_n - a_n$ est inférieure à celle-ci.

Justification de la convergence

Par construction, pour tout entier n, $a_n \leqslant b_n$. De plus, la suite (a_n) est croissante et la suite (b_n) est décroissante. En effet, Pour tout n, on a :

$$a_n \leqslant \frac{a_n + b_n}{2} \leqslant b_n$$

Ainsi, que l'on ait $a_{n+1} = a_n$ ou $a_{n+1} = \frac{a_n+b_n}{2}$, on a bien $a_{n+1} \geqslant a_n$.
De même, on montre que $b_{n+1} \leqslant b_n$

On peut le prouver par récurrence

D'autre part, $\lim\limits_{n\to\infty}(b_n - a_n) = \lim\limits_{n\to\infty}\frac{b_0 - a_0}{2^n} = 0$.

Les suites (a_n) et (b_n) sont donc adjacentes, et par suite convergentes vers une même limite l. Puisque pour tout n, on a $a_n \leqslant x_0 \leqslant b_n$, en passant à la limite, on obtient $l \leqslant x_0 \leqslant l$ et donc $l = x_0$

Mise en œuvre

On suppose implémentée une fonction `TP` simulant une fonction réelle f pour laquelle on connaît deux valeurs a et b encadrant une solution de l'équation. Le script ci correspond à l'implémentation SCILAB de l'algorithme de dichotomie.

```
e = input("précision : ")
a = valeurinferieure
b = valeursuperieure
while b - a > e
   c=(a + b)/2
   if TP(a)*TP(c) > 0 then
      a=c
   else
      b=c
   end
end
disp(a,"la solution est environ ")
```

Cet algorithme utilise certaines valeurs plusieurs fois, et effectue donc des calculs redondants. On pourrait améliorer l'algorithme en stockant les valeurs dans deux autres variables. Toutefois, cette version me parait plus lisible et convient pour les besoins d'ECS1.

Exercice 48

⋆ Prenez quelques instants pour commenter le code ci-dessus avec // et vos remarques pour vous souvenir de la démarche.

⋆ Quand vous avez l'algorithme en tête, cachez le (si si). Vous pouvez maintenant écrire (vous avez caché l'algorithme ?) une fonction `tp6` qui calcule $f(x) = x^3 + 2x^2 + 3x + 4$ et écrivez (toujours sans regarder !) un algorithme de dichotomie pour approcher la racine de cette fonction en prenant comme valeurs initiales $a = -2$ et $b = -1$.

⋆ *Si vous avez terminé en avance...*
Démontrez que l'équation $f(x) = 0$ *admet une unique solution dans* $[-2; -1]$

dans un sujet, on fait ceci avant d'écrire un algorithme

6.3 Approximation par une suite itérative

Un grand nombre de problèmes sur les suites consiste en la donnée d'une fonction φ **continue** et d'un réel u_0. On étudie la suite définie par :

$$\begin{cases} u_0 \in \mathbb{R} \\ u_{n+1} = f(u_n) \end{cases}$$

⋆ Pourquoi les mots en gras sont-ils importants ?

On sait alors que **si** (u_n) converge vers une limite l, celle-ci vérifie $\varphi(l) = l$.

Exercice 49 On veut déterminer une valeur approchée de $\sqrt{2}$. Pour cela, on cherche à résoudre l'équation $(\mathcal{E})$ $x^2 - 2 = 0$ avec une suite itérative.

1. Montrer que $(\mathcal{E})$ équivaut à $\varphi(x) = x$ avec $\varphi : x \mapsto 2x - \frac{2}{x}$
2. On pose $u_0 := 2$ et $u_{n+1} = \varphi(u_n)$. Taper la fonction récursive `suite` suivante qui prend en entrée un nombre `n` et calcule u_n. Faire quelques essais et constater que la suite ne converge pas.

```
function y=phi(x)
    y=2*x-2/x
endfunction

function u=suite(n)
    if n==0 then
        u=2
    else
        u=phi(suite(n-1))
    end
endfunction
```

Il serait plus efficace de calculer les valeurs au fur et à mesure avec la fonction `phi`. Toutefois, le code est plus simple avec cette fonction `suite`. Faisons donc comme si les calculs étaient instantanés.

3. Modifier la fonction `phi` pour qu'elle renvoie $\varphi(x) = \frac{1}{3}\left(2x + \frac{2}{x}\right)$. Vérifiez rapidement que $\sqrt{2}$ est bien un point fixe de φpuis écrivez un programme `approx` qui prend en entrée une précision `p` et renvoie une approximation de a à la précision `p` ainsi que le nombre d'itérations nécessaires pour arriver à cette précision. On pourra utiliser la fonction `suitephi` définie précédemment.
4. Même question avec $\varphi(x) = \frac{1}{2}\left(x + \frac{2}{x}\right)$

6.4 Exercices

Exercice 50 : **Une résolution par dichotomie**

1. Démontrer que l'équation $\tan x - x = 0$ admet une unique solution $\alpha \in \left]\frac{\pi}{2}; \frac{3\pi}{2}\right[$.

2. Vérifier que $4 < \alpha < 4,7$ puis écrire une procédure SCILAB qui détermine, par dichotomie, une valeur approchée à 10^{-4} près de α.

On pourra écrire dans un premier temps une fonction qui calcule $\tan x - x$ *pour une entrée* x.

Exercice 51 **Une résolution par itération**

1. Démontrer que l'équation $\cos x - x = 0$ admet une unique solution $\alpha \in [0; 1]$.

2. On définit la suite (u_n) par $u_0 = 0$ et $u_{n+1} = \cos(u_n)$

   a. Prouver que $\forall n \in \mathbb{N},\ |u_n - \alpha| \leqslant (\sin(1))^n$.

   b. Déduire la convergence de (u_n) vers α

3. Écrire un programme permettant de calculer une valeur approchée de α à 10^{-6}

Exercice 52 : **Une équation, deux méthodes**

On s'intéresse ici à l'équation $f(x) - x = 0$ où f est la fonction définie sur $\mathbb{R}$ par $\forall x \in \mathbb{R},\ f(x) = (x^2 + 1)\, e^{-x}$

1. Démontrer que l'équation $f(x) - x = 0$ admet une unique solution $\alpha \in \left]\frac{1}{2}; 1\right[$.
   On rappelle que $2 < e < 3$

2. Écrire un programme qui calcule par dichotomie une valeur approchée de α à 10^{-6} près.

3. Nous allons maintenant approcher α par itérations successives.
   On considère la suite définie par : $u_0 = 1$ et $\forall n \in \mathbb{N},\ u_{n+1} = f(u_n)$

   a. Prouver que $\forall x \in \left[\frac{1}{2}; 1\right],\ |f'(x)| \leqslant \frac{1}{4}$ et $\frac{1}{2} \leqslant f(x) \leqslant 1$

   b. Montrer que $\forall n \in \mathbb{N},\ |u_n - \alpha| \leqslant \left(\frac{1}{4}\right)^n$

   c. En déduire que (u_n) converge vers α

   d. Écrire un programme utilisant la suite (u_n) pour approcher α à 10^{-6} près.

6.5 Complément : Méthode de Newton

Cette méthode n'est pas explicitement au programme. Néanmoins, tous les éléments nécessaires à son étude sont au programme de première année. Nous allons donc en voir les grandes lignes ainsi que l'algorithme d'application.

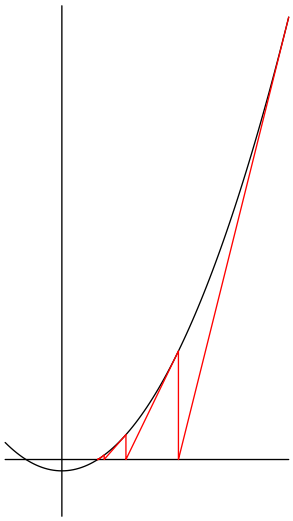


Figure 6.1: Méthode de Newton pour résoudre les équations.

L'algorithme se comprend bien graphiquement. Partant d'un point pas trop éloigné de la racine, on suit la tangente jusqu'à l'axe des abscisses, ce qui nous rapproche du point d'intersection de la courbe de f avec (Ox). On remonte jusqu'à la courbe et on suit à nouveau la tangente...

Exercice 53 Supposons connue une valeur approchée x_n de la solution. Déterminer l'abscisse x_{n+1} du point d'intersection de la tangente à $\mathcal{C}_f$ au point d'abscisse x_n avec l'axe des abscisses.

Méthode de Newton :
On suppose f de classe $\mathcal{C}^2$, convexe et croissante ($f' > 0,\ f'' > 0$) sur un intervalle $[a;b]$ dans lequel f s'annule en r (unique d'après les hypothèses). Alors la suite définie par $x_0 = b$ et $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$ est strictement décroissante et de limite r.

Démonstration : Posons $\varphi : x \mapsto x - \frac{f(x)}{f'(x)}$ *de sorte que* $x_{n+1} = \varphi(x_n)$. *On a* $\varphi' = \frac{ff''}{f'^2} > 0$ *d'après les hypothèses donc* φ *est strictement croissante et la suite* (x_n) *est monotone. Puisque* $x_1 = f(x_0) = f(b) = b - \frac{f(b)}{f'(b)} < b = x_0$, *et donc*[1] *pour tout entier* n, $r < x_{n+1} < x_n$. *La suite est décroissante, minorée donc convergente vers l'unique point fixe* r.

[6.1] récurrence classique qu'il faut savoir rédiger

La formule de Taylor-Lagrange appliquée à f *en* x_n *assure l'existence de* ξ, *tel que :*

$$\forall n \in \mathbb{N}, f(x_n + (r - x_n)) = f(x_n) + (r - x_n)f'(r) + (r - u_n)^2 \frac{f''(\xi)}{2}$$

$$(x_n - r)f'(x_n) - f(x_n) = (r - u_n)^2 \frac{f''(\xi)}{2}$$

$$x_n - r - \frac{f(x_n)}{f'(x_n)} = (r - u_n)^2 \frac{2f''(\xi)}{f'(x_n)}$$

$$|x_{n+1} - r| \leqslant |r - u_n|^2 \frac{|f''(\xi)|}{2|f'(x_n)|}$$

$$\text{donc } \forall n \in \mathbb{N}, |x_n - r| \leqslant \frac{1}{K}(K|x_0 - r|)^{2^n}$$

On a posé ici $K = \dfrac{\max\limits_{[a;b]} f''}{2 \min\limits_{[a;b]} f'}$ *qui est bien défini car* f' *et* f'' *sont continues donc bornées sur le segment* $[a;b]$.

Bref, la convergence est très rapide (quadratique) si $x_0 - r$ est suffisamment petit.

Exercice 54 : Programmer la méthode de Newton pour résoudre l'équation $x + e^x - 2 = 0$ en prenant comme point de départ $x_0 = 1$.

Et si vous êtes en avance, prenez donc quelques instants pour vérifier que la fonction vérifie bien les hypothèses.

Voici un code dont vous pouvez vous inspirer. Il est agrémenté de quelques commandes graphiques et commence à 10 pour voir plus de choses.

```
function y=fonction(x)
    y=(x+exp(x)-2)
endfunction

function y=derivee(x)
y=(1+exp(x))
endfunction

clf()
x=[0.5:0.05:10];plot(x,fonction(x))
x=10
a=[x;x]
b=[0;fonction(x)]
plot2d(a,b)
for i=1:10
    x=x-fonction(x)/derivee(x)
    a=[a;x;x];b=[b;0;fonction(x)]
end
plot2d(a,b)
```

Pour construire le graphique, on a besoin de la fonction f, appelée ici `fonction` et de sa dérivée f' appelée `derivee`.

Mais s'il s'agit uniquement d'approcher la solution de l'équation, on pourra préférer une fonction `phi` qui code $\frac{f}{f'}$

6.6 Complément : Méthode de la sécante

Également appelée « Méthode de Lagrange », nous allons d'abord en illustrer le principe avant de voir dans quelles conditions elle est applicable.

Principe

Le principe se comprend très bien visuellement. Connaissant un encadrement $[a;b]$ de la solution r cherchée, on trace la corde reliant $A(a;f(a))$ à $(b;f(b))$ et l'abscisse x_1 du point d'intersection de la corde avec l'axe des abscisses est plus proche de r que a. On itère le procédé.

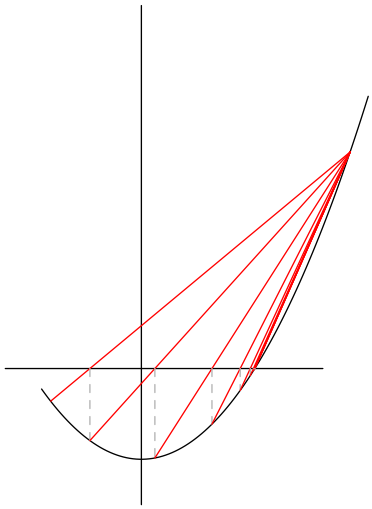


Figure 6.2: Méthode de la sécante.

Méthode de la sécante :
On considère un intervalle $[a, b]$ et une fonction f de classe $\mathcal{C}^2$ sur $[a, b]$ strictement convexe ($f'' > 0$) vérifiant $f(b) > 0$ et $f(a) < 0$.
La suite (x_n) définie par $x_0 := a$ et $x_{n+1} := \dfrac{x_n f(b) - b f(x_n)}{f(b) - f(x_n)}$ converge en croissant vers r.

Démonstration : On a $a < r < b$. *Supposons construit* x_n *tel que* $a < x_n < r < b$.
Alors $x_{n+1} = \lambda x_n + \mu b$ *avec* $\lambda = \frac{f(b)}{f(b)-f(x_n)} > 0$ *et* $\mu = \frac{-f(x_n)}{f(b)-f(x_n)} > 0$ *qui vérifient* $\lambda + \mu = 1$.
Ainsi, $x_{n+1} \in]x_n; b[$ *et, par convexité de* f,
$0 = \lambda f(x_n) + \mu f(b) \geqslant f(\lambda x_n + \mu b) = f(x_{n+1})$.
Puisque r *est l'unique valeur pour laquelle* f *s'annule,*
on en déduit $x_n \leqslant x_{n+1} \leqslant r \leqslant b$. *La suite est donc croissante et majorée par* r *donc convergente vers un réel* $l \leqslant r$ *qui vérifie :* $l = \dfrac{l f(b) - b f(l)}{f(b) - f(l)}$ *qui donne* $f(l) = 0$ *puisque* $l < b$ *puis* $l = r$

Voici un exemple de la méthode de la sécante appliquée à la fonction $x \mapsto x^2 + e^x - 3$ avec $a = 0$ et $b = 3$

```
function y=fonction(x)
    y=(x^2+exp(x)-3)
endfunction

clf()
a=0
b=3
x=[0:0.05:b];plot(x,fonction(x))
x=a
M=[a;b]
N=[fonction(a);fonction(b)]
plot2d(M,N)
for i=1:10
    x=(x*fonction(b)-b*fonction(x))/(fonction(b)-fonction(x))
    M=[x;b]
    N=[fonction(x);fonction(b)]
    plot2d(M,N)
end
```

Exercice 55 Adapter le code ci-dessus pour qu'il donne une solution approchée de l'équation $x^2 - x - 6 = 0$ avec $a = -3$ et $b = 0$ puis avec $a = 0$ et $b = 5$. Le premier cas ne correspond pas tout à fait aux hypothèses présentées ci-avant mais l'algorithme converge tout de même.

6.7 Quelques corrigés

Solution pour les exercices du début de la séquence

Exercice 2 :
Approche de la solution de l'équation $f(x) = 0$ comprise dans l'intervalle $[-2; -1]$, où $f(x) = x^3 + 2x^2 + 3x + 4$.

```
function y=TP(x)
    y=x^3+2*x^2+3*x+4
endfunction

e = input("précision : ")
a = -2
b = -1
while b - a > e
   c=(a + b)/2
   if TP(a)*TP(c) > 0 then
      a=c
   else
      b=c
   end
end
disp(a,"la solution est environ ")
```

Exercice 4 :
Voici les trois fonctions dont on a besoin pour tester, légèrement différentes.

```
function y=phi(x)
    y=2*x-2/x
endfunction
```

```
function y=phi(x)
    y=(2*x-2/x)/3
endfunction
```

```
function y=phi(x)
    y=(2*x-2/x)/2
endfunction
```

```
function u=suite(n)
    if n==0 then
        u=2
    else
        u=phi(suite(n-1))
    end
endfunction
```

Quelle que soit la fonction qu'on utilise pour définir la suite, le code est le même pour calculer u_n

Quand la suite est convergente, on obtient le nombre d'itérations nécessaires pour approcher $\sqrt{2}$ à moins de 10^{-6} avec :

```
n=0
while suite(n)-sqrt(2)>=10^(-6)
    n=n+1
end
disp(n)
```

Même code pour les deux mais on trouve

Corrigé succinct des exercices de la section 4

Je livre ici une proposition de code pour la partie algorithmique. Si j'ai le temps, je compléterai la partie mathématique ultérieurement.

Dichotomie sur $\tan x - x = 0$

La seule chose qui change, c'est la fonction, tout le reste de l'algorithme est identique (à part les valeurs initiales de a et b, et la précision.

```
function y=f(x)
    y=tan(x)-x
endfunction

a=4
b=4.7
while b-a>=10^(-4)
    c=(a+b)/2
    if f(a)*f(c)>0 then
        a=c
    else
        b=c
    end
end

disp(a) //ne pas oublier
```

Itération sur $\cos x - x = 0$

```
u=0
n=0

while sin(1)^n>=10^(-6)
    n=n+1
    u=cos(u)
end

disp(u)
```

Ici encore, l'algorithme n'est pas du tout optimal, on pourrait stocker les valeurs successives de $\sin(1)^n$ dans une variable pour réduire le nombre de calculs.

Une équation, deux méthodes

Il est souvent plus pratique de définir une fonction avant l'algorithme. Attention, la fonction ne sera pas la même si l'algorithme utilise la dichotomie ou une itération vers un point fixe.

Dichotomie :

```
function y=f(x)
    y=(x^2+1)*exp(-x)-x
endfunction

a=0.5
b=1
while b-a>=10^(-6)
    c=(a+b)/2
    if f(a)*f(c)>0 then
        a=c
    else
        b=c
    end
end

disp(a)
```

Itérative :

```
function y=f(x)
    y=(x^2+1)*exp(-x)
endfunction

u=0
n=0
while 0.25^n>=10^(-6)
    n=n+1
    u=f(u)
end

disp(u)
```

TP7 | Variables aléatoires

Les probabilités sont un des impondérables du programme de mathématiques, qui se prête particulièrement bien à l'expérimentation en général, informatique en particulier.

7.1 Simulation de variables aléatoires discrètes

Nous avons déjà vu la commande `rand` et utiliserons plus ici la commande `grand`, qu'il faut entendre comme g-rand, *generate randomness*, qui génère des nombres aléatoires de meilleure manière, et surtout avec plus d'options que son analogue `rand`.

dans tous les algorithmes qui suivent, les commandes sont compatibles avec les vecteurs ».

`grand` permet de simuler directement les principales lois de probabilités discrètes ou continues, et de les simuler en grand nombre sous forme de matrices pour des simulations, estimations... C'est la commande idéale, prête à l'emploi.

Toutefois, il peut être intéressant de savoir les implémenter à partir d'une loi uniforme sur $[0;1]$ obtenue avec la commande `rand()`. Dans les exemples ci-dessous, le premier code est la simulation avec `rand()`, le second sa contrepartie `grand`.

Ainsi, `rand()` peut être remplacé par `rand(5,7)` par exemple si l'on souhaite une matrice de valeurs aléatoires.

Loi de Bernoulli

Pour simuler une expérience de Bernoulli qui renvoie 1 avec une probabilité $p \in]0;1[$ contenue dans une variable `p` avec la commande :

```
X=(rand()<p)
```

`X` contient `T` avec une probabilité `p` ou `F` qui peuvent respectivement être utilisés comme 1 et 0 dans d'éventuels calculs.

Loi uniforme sur $[\![1\ ;n]\!]$

Si `n` contient la valeur d'un nombre entier, le code suivant...

```
X=floor(n*rand())+1
Y=grand(m, n, "uin", min, max)
```

X contient un nombre entier au hasard entre 1 et n inclus.
Y est une matrice $m \times n$ de nombres aléatoires suivant $\mathcal{U}([\![min; max]\!])$

Loi binomiale

```
X=sum(rand(1,n)<p)
Y=grand(m, n, "bin", N, p)
```

X compte le nombre de succès obtenus lors de n répétitions d'une épreuve de Bernoulli de paramètre p.
Y est une matrice $m \times n$ de nombres aléatoires suivant $\mathcal{B}(n, p)$

Loi géométrique

```
while rand()>=p
    X=X+1
end
```

X compte le nombre de tirages nécessaires suivant une loi de Bernoulli de paramètre p avant obtention du premier succès.

```
Y = grand(m, n, "geom", p)
```

Y est une matrice $m \times n$ de nombres aléatoires suivant la loi géométrique de paramètre p

Loi de Poisson

```
Y = grand(m, n, "poi", mu)
```

Y est une matrice $m \times n$ de nombres aléatoires suivant

7.2 Exercices

Exercice 56

1. Une urne contient B boules blanches et N boules noires ($N \leqslant B$). Compléter la fonction ci-dessous qui prend en entrée trois nombres : $n \leqslant N$, N et B, simule le tirage sans remise de n boules dans l'urne puis renvoie le nombre de boules noires obtenues.

```
function c=noires(n,N,B)
c=0
for i=1:n
    if (N+B)*rand()<=N then
        - - - - - - - - - -
        - - - - - - - - - -
    else
        - - - - - - - - - -
    end
end
endfunction
```

2. Que fait le programme ci dessous ?

```
n=10
N=10
B=20
clf()
S=zeros(1,N+1)
for i=1:10000
    r=noires(n,N,B)+1
    S(r)=S(r)+1
end
S=S/10000
bar(0:N,S)
```

3. *Si vous avez de l'avance... le nombre* n *de boules tirées a été restreint à* $n \leqslant N$ *pour éviter la possibilité d'épuiser les boules noires. Modifier la fonction pour qu'elle prenne en compte l'éventualité d'arriver au bout d'une des deux couleurs de boules. Ainsi, on peut choisir* $n \leqslant N + B$

Exercice 57 On considère une suite de lancers successifs et indépendants d'une pièce de monnaie. On suppose que la probabilité p d'obtenir Pile est de $\frac{2}{3}$. On note X le rang d'apparition du premier « double pile » (c.a.d. le rang d'apprition du deuxième des deux piles)
Compléter le code suivant pour qu'il demande un entier n et simule n fois l'expérience ci-dessus puis calcule le nombre moyen de tirages nécessaires à l'obtention de la première paire de Pile.
On code pile par 1 (ou %T) et face par 0 (ou %F)

```
n = input('Nombre de simulations : ')
S = zeros(1,n)
for i = 1:n
    k = 0
    x = 0
    y = 0
    while x*y == 0
        _ _ _ _ _ _ _ _ _ _ _ _ _ _
        _ _ _ _ _ _ _ _ _ _ _ _ _ _
        _ _ _ _ _ _ _ _ _ _ _ _ _ _
    end
    S(i) = k
end
moyenne = _ _ _ _ _ _ _ _ _ _ _ _ _ _
disp (moyenne, 'le nombre moyen de tirage est :')
```

Exercice 58 : **ESC1999**
Soit $n \in \mathbb{N}^*$. Une urne U_n contient n boules numérotées de 1 à n. On effectue dans cette urne une succession de tirages d'une boule, en appliquant la règle suivante : si une boule tirée porte le numéro k, avant de procéder au tirage suivant, on enlève toutes les boules dont le numéro est supérieur ou égal à k.
On note X_n la variable aléatoire égale au nombre de tirages nécessaires pour vider l'urne U_n de toutes ses boules.
Compléter le programme suivant afin qu'il simule l'expérience :

```
// d'après ESC1999
b = input('Entrez le nombre initial de boules ')
tirages=0
while _ _ _ _ _
    numero= _ _ _ _ _ _ rand()  _ _ _
    b= _ _ _
    tirages = _ _ _ _ _ _
end
disp ('tirages' , tirages , 'L''urne est vidée en ')
```

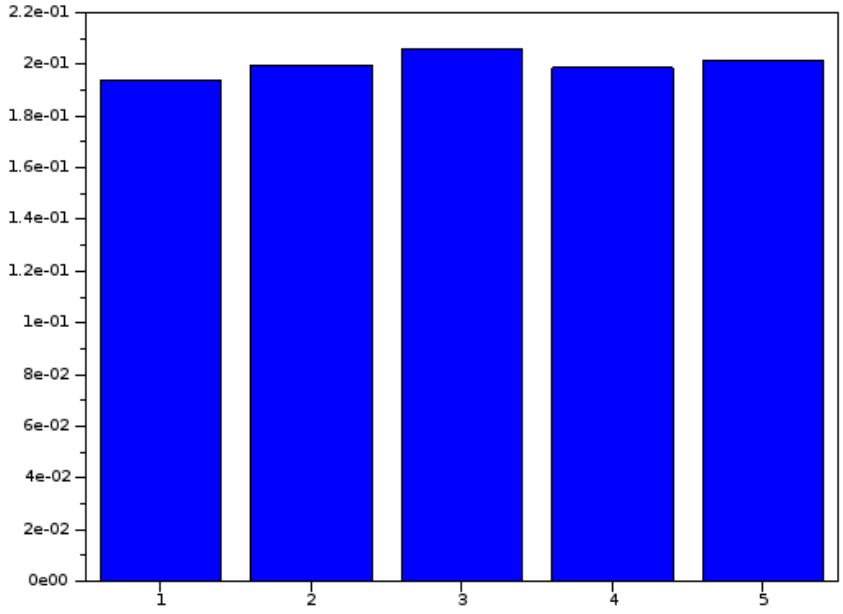
Exercice 59 : ÉCRICOME ECE 2015

L'urne U_1 contient N boules : $(N-1)$ boules blanches et 1 boule noire.
On effectue des tirages **sans remise** dans l'urne U_1, jusqu'à l'obtention de la boule noire. On note X la variable aléatoire qui prend pour valeur le nombre de tirages nécessaires pour l'obtention de la boule noire.

1. On simule 10000 fois cette expérience aléatoire.
   Recopier et compléter le programme SCILAB suivant pour qu'il affiche l'histogramme donnant la fréquence d'apparition du rang d'obtention de la boule noire :

```
N = input('Donner un entier naturel non nul');
S = zeros(1,N)
for k = 1:10000
    i = 1;
    M = N;
    while  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
        i = i+1
        M = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
    end
    S(i) = S(i) + 1;
end
disp (S/10000)
bar (S/10000)
```

2. On exécute le programme complété ci-dessus. On entre 5 au clavier et on obtient l'histogramme suivant :



Quelle conjecture pouvez-vous émettre sur la loi de la variable aléatoire X ?

Exercice 60 : Paradoxe du Duc de Toscane
On lance trois dés simultanément et on appelle X la variable aléatoire égale à la somme des trois numéros obtenus.

1. Écrire une fonction qui simule X.
2. Compléter cette fonction en un programme qui effectue 20000 lancers et compte le nombre fois où X vaut 9 et celui où X vaut 10.

Remarque : Le paradoxe vient du fait qu'il y a exactement le même nombre de décompositions pour obtenir 9 ou 10 (il y en a 6) mais que $\mathbb{P}(X = 9) \neq \mathbb{P}(X = 10)$. Saurez-vous résoudre ce paradoxe ?

Exercice 61 : Petit défi bonus, redresser une pièce truquée
Supposons que vous disposez d'une fonction `cheat` qui simule une expérience de Bernoulli de paramètre $\frac{1}{3}$ (une pièce truquée). Écrivez un code qui simule le lancer d'une pièce bien équilibrée à partir de cette « pièce truquée ».
Indication : Si vous lancez plusieurs fois la pièce, certains événements sont équiprobables... Indication

7.3 Simulation de variables aléatoires continues

Pour simuler une loi de probabilité continue dont la densité est non non nulle sur un intervalle ouvert I et nulle en dehors, utiliser sa fonction de répartition, qui sera continue et strictement croissante sur I[1] et réalisera donc une bijection de I sur $]0;1[$. Dans le cas de la loi exponentielle, nous pouvons expliciter une réciproque.

[7-1] à vérifier !

Loi uniforme continue

```
Y = grand(m, n, "unf", Low, High)
```

Y est une matrice $m \times n$ de nombres aléatoires suivant

Loi exponentielle

Pour simuler une loi de probabilité continue dont la densité est non non nulle sur un intervalle ouvert I et nulle en dehors, utiliser sa fonction de répartition, qui sera continue et strictement croissante sur I^2 et réalisera donc une bijection de I sur $]0;1[$. Dans le cas de la loi exponentielle, nous pouvons expliciter une réciproque.

7-2 à vérifier !

```
Y = grand(m, n, "exp", E)
```


```
Y = grand(m, n, "exp", E)
```


Y est une matrice $m \times n$ de nombres aléatoires suivant une loi exponentielle d'espérance $E = \frac{1}{\lambda}$

Loi normale

```
Y = grand(m, n, "nor", mu, sigma)
```

Y est une matrice $m \times n$ de nombres aléatoires suivant une loi normale de moyenne mu et d'écart-type sigma

7.4 Phénomènes limites.

Simulation de $\mathcal{B}(n; \frac{\lambda}{n})$ et comparaison avec la loi de Poisson

7.5 Échantillonnage

7.6 Solution des exercices

Exercices sur les lois discrètes

tirage sans remise de n boules parmi N noires et B blanches

```
function c = noires(n,N,B)
c = 0
for i = 1:n
    if (N+B)*rand()<=N then
        N = N-1
        c = c+1
    else
        B = B-1
    end
end
endfunction
```

Le deuxième code simule 10000 expériences avec 10 boules noires, 20 boules blanches, parmi lesquelles on tire 10 boules, remplit au fur et à mesure un vecteur qui contient le nombre d'apparition de chaque valeur possible, puis calcule la fréquence d'apparition de chaque valeur et affiche le résultat sous forme de diagramme en barres.

lancers jusqu'à Pile-Pile

```
n = input('Nombre de simulations : ')
S = zeros(1,n)
for i = 1:n
    k = 0
    x = 0
    y = 0
    while x*y == 0
        x = y
        y = (3*rand()<2)
        k = k+1
    end
    S(i) = k
end
moyenne = sum(S)/n
disp (moyenne, 'le nombre moyen de tirage est :')
```

ESC1999

```
// d'après ESC1999
b=input('Entrez le nombre initial de boules ')
tirages=0
while b>0
    numero = floor(b*rand()) + 1
```

```
    disp(numero)
    b= min(numero-1,b)
    tirages = tirages + 1
end
disp('tirages',tirages,'L''urne est vidée en ')
```

ÉCRICOME ECE 2015

```
N = input('Donner un entier naturel non nul');
S = zeros(1,N)
for k = 1:10000
    i = 1;
    M = N;
    while M*rand()>1
        i = i+1
        M = M-1
    end
    S(i) = S(i)+1;
end
disp (S/10000)
bar (S/10000)
```

Il semble que X suive une loi uniforme sur $[\![1;5]\!]$

TP8 | Intégration

Les intégrales sont partout dans les mathématiques appliquées. Certaines fonctions, même très classiques comme $x \mapsto exp(-x^2)$ n'ont pourtant pas de primitive explicite. Dans ce cas, ou dans le cas de fonctions définies par une table de valeur relevées expérimentalement par exemple, les méthodes numériques sont nécessaires pour calculer l'intégrale sur un intervalle de ces fonctions.

Il existe un grand nombre[1] de méthodes pour approcher $\int_a^b f(x)dx$. Elles diffèrent par leur complexité, leur rapidité de convergence, leur précision...

[1] cherchez par exemple *Calcul numérique d'une intégrale* sur Wikipedia pour vous faire une idée

Nous verrons ici :

8.1 Méthode des Rectangles

Le principe de l'algorithme est simple :

- on subdivise l'intervalle $[a;b]$ en $x_0 = a,\ x_1,\ \dots,x_{n-1},\ x_n = b$, par exemple avec la commande `linspace(a,b,n+1)`. Nous avons donc $n+1$ valeurs qui définissent n intervalles.
- on calcule les hauteurs des rectangles (images par f des x_i)
- on somme les aires des rectangles.

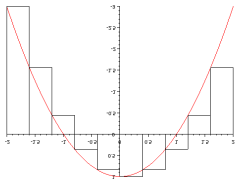


Figure 8.1: rectangles à gauche

Il y a trois manières de définir les rectangles :
1. méthode dite « rectangles à gauche »
2. méthode dite « rectangles à droite »
3. méthode dite « du point médian » qui est la plus efficace.

Un petit dessin valant mieux qu'un long discours, en voici l'illustration ci-contre où on a appliqué les méthodes de rectangles à l'application $x \mapsto x^2 - 1$:

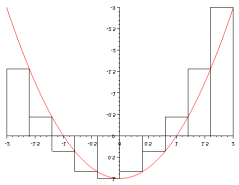


Figure 8.2: rectangles à droite

les formules

Les formules (qui ne seront pas démontrées ici) utilisées dans l'algorithme sont :

$$\int_a^b f(x)dx = \lim_{n\to\infty} \frac{b-a}{n} \sum_{k=0}^{n-1} f(a + k\frac{b-a}{n})$$
$$= \lim_{n\to\infty} \frac{b-a}{n} \sum_{k=1}^{n} f(a + k\frac{b-a}{n})$$

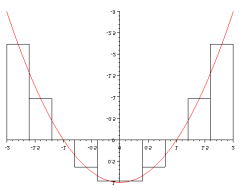


Figure 8.3: méthode du point milieu

$$= \lim_{n\to\infty} \frac{b-a}{n} \sum_{k=0}^{n-1} f\left(a + (k+\frac{1}{2})\frac{b-a}{n}\right)$$

La méthode du point milieu est visuellement plus équilibrée et on peut démontrer mathématiquement qu'elle est plus efficace que les deux autres.

Exercice 62 Saurez-vous retrouver quelle formule correspond à quelle méthode (gauche, droite, milieu ?)

Disons que l'on souhaite calculer $\int_0^2 e^{x^2} dx$.

8.2 attention à bien appliquer les points, comme `.^`, ou `./`, *etc.*

On commence par définir la fonction `f`, *d'une manière compatible avec les vecteurs*[2], puis une subdivision de l'intervalle d'intégration avec `x=linspace(0,2,101)`. Nous avons donc ici $n = 100$ intervalles et un vecteur `x` comportant $n+1$ valeurs. On modifie alors `x` pour ne garder que les valeurs utiles à la définition des « rectangles ».

- `x=x(1:100)` pour les rectangles à gauche
- `x=x(2:101)` pour les rectangles à droite
- `x=0.5*(x(1:100)+x(2:101))` pour le point milieu[3]

8.3 vous êtes bien convaincu que ça convient ?

on peut alors calculer `y=f(x)` la valeur approchée de l'intégrale est donnée par `2*sum(y)/100`

Quand vous expérimentez sur l'intégration avec Scilab, vous pouvez utiliser la commande `integrate` pour vérifier la précision de vos calculs. Celle-ci n'est pas au programme mais fait appel à des algorithmes optimisés pour calculer votre intégrale et peut donc être considérée comme une référence fiable.

```
X=integrate('f(x)','x',0,2);X
```

Voici un code facilement adaptable à tous les cas :

```
function y=f(x)
    y=exp(x.^2)
endfunction

n=100000;
a=0;
b=2;
x=linspace(a,b,n+1);
x=x(1:n); // ou x=x(2:n+1)
// ou x=0.5*(x(1:n)+x(2:n+1))
y=f(x);
integrale=(b-a)/n*sum(y)
```

Pour cette fonction, positive et croissante, votre résultat sera systématiquement inférieur avec les rectangles à gauche, supérieur avec les rectangles à droite, et mieux équilibrée avec la méthode du point milieu. Dans tous les cas, l'approximation sera d'autant meilleure que n sera grand. Cela se comprend bien sur un dessin.

N'hésitez pas à tester avec des valeurs importantes comme 1000, 10000 voire 100000, le calcul reste rapide.

Résumé des étapes

Pour mettre en œuvre la méthode des rectangles, on pourra suivre les étapes suivantes :

1. définir la fonction à intégrer (de manière vecteur-proof)
2. définir la subdivision x de l'intervalle d'intégration en n valeurs `x=linspace(a,b,n+1)`
3. modifier x en vecteur des valeurs à utiliser selon le type de rectangle choisi `x=x(1:n)` pour les rectangles à gauche
4. calculer le vecteur y des valeurs correspondantes `y=f(x)`
5. appliquer la formule de Riemann `integrale=sum(y)*(b-a)/n`

Exercice 63

1. Écrire une fonction Scilab qui modélise la densité de la loi normale centrée réduite $f : x \mapsto \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$

2. On rappelle que $\int_{-\infty}^{0} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx = \frac{1}{2}$, écrire une fonction Scilab qui modélise la fonction de répartition $F : x \mapsto \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt$

3. Comparer avec ce que l'on peut obtenir en utilisant la commande `integrate`

Exercice 64 Écrire un programme Scilab qui demande un nombre p en entrée, et calcule pour toutes les valeurs de $k \in [\![1; p]\!]$ le nombre d'itérations nécessaires pour arriver à une approximation de $\int_0^1 3x^2 dx$ avec une précision 10^{-k}.
On rappelle que cette intégrale vaut 1

Remarque sur la vitesse de convergence

On peut démontrer que les méthodes des rectangles à gauche et à droite ont une précision en $\frac{C}{n}$ où C est une constante dépendant de la longueur de l'intervalle d'intégration et de la fonction.
La méthode du point milieu offre quant à elle une précision en $\frac{C'}{n^2}$, ce qui est considérablement plus efficace.

8.2 Méthode de Monte Carlo

Monte Carlo, c'est un rocher, des princes et des princesses, un Grand Prix, un Masters, RMC, TMC et ... une méthode d'intégration numérique !

Le principe est le suivant : Disons que vous souhaitez déterminer la surface d'une mare et que pour cela vous disposez de moyens illimités (d'une imagination illimitée) qui vous permettent via un hélicoptère de disperser uniformément des cailloux blancs dans un rectangle de surface connue S englobant la mare.

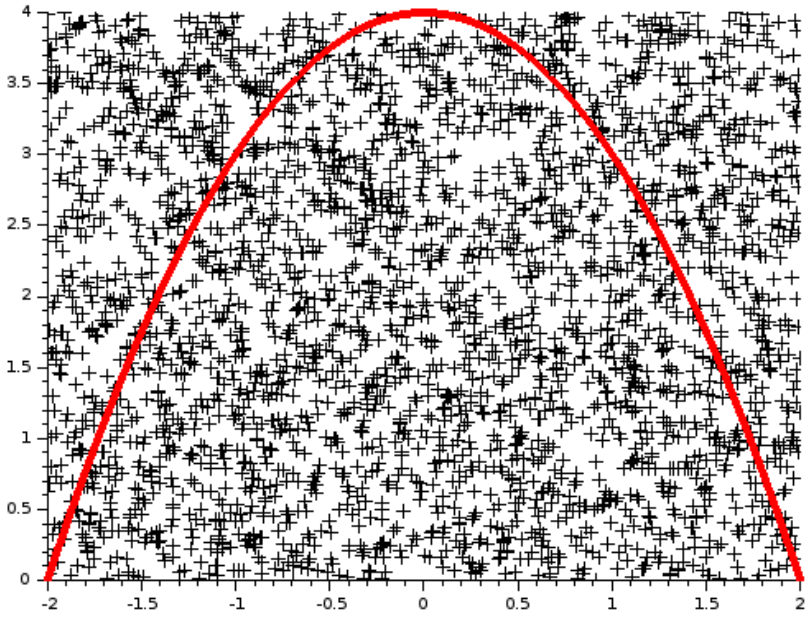
Quel est le rapport ?

Et bien justement, le rapport entre le nombre de cailloux tombés dans la mare, appliqué à la surface S donne une bonne approximation de la surface de la mare si le nombre de pierres lancées est suffisant.

On peut utiliser la même idée pour approcher la valeur d'une intégrale de fonction.

- On détermine n valeurs $(x_k; y_x)_{1 \leqslant k \leqslant n}$, les x_k suivant une loi uniforme sur $[a; b]$ et les y_k une loi uniforme sur $f([a; b])$
- Pour chaque couple de valeur $(x_k; y_k)$, on compare y_k et $f(x_k)$
- on détermine la proportion de points sous la courbe que l'on multiplie par l'aire du rectangle.

Le schéma ci-dessous illustre la méthode :



Exercice 65

On rappelle que l'équation d'un cercle de centre O et de rayon 1 est $x^2 + y^2 = 1$.

1. Écrire un programme qui utilise une méthode de type *Monte Carlo* pour évaluer la surface de ce disque (π).
2. Modifier le code pour déterminer le nombre d'itérations nécessaires pour approcher πà moins de 10^{-4} près

---

Exercice 66

On rappelle que l'équation d'un cercle de centre O et de rayon 1 est $x^2 + y^2 = 1$.

1. Écrire un programme qui utilise une méthode de type *Monte Carlo* pour évaluer la surface de ce disque (π). On pourra prendre 1000 couples de valeurs
2. Modifier le code pour déterminer le nombre d'itérations nécessaires pour approcher π à moins de 10^{-4} près
3. Modifier encore le code pour qu'il demande en entrée le nombre n de valeurs dans la Méthode, puis évalue sur 1000 simulations distinctes l'écart-type par rapport à π.
4. En remarquant que la courbe d'équation $y = \sqrt{1 - x^2}$ est celle d'un demi cercle de centre O et de rayon 1, comparer le nombre d'itérations nécessaires pour obtenir une approximation de πde précision comparable avec une méthode de rectangles sur point milieu ou méthode de Monte Carlo.

Remarque : La Méthode de Monte Carlo n'est pas particulièrement efficace pour calculer les intégrales auxquelles on peut se retrouver confronté en prépa. Elles sont toutefois concrètement utilisées pour des problèmes de dimensions plus grande, avec par exemple de nombreux paramètres comme pour des décisions stratégiques, ou en finance

8.3 Propositions de solutions

Rectangles

Ici, on calcule l'intégrale entre 0 et x, que l'on ajoute ou soustrait à $\frac{1}{2}$ en fonction du signe de x.
En effet, on a :

$$\int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt = \int_{-\infty}^{0} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt + \int_{0}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt = \frac{1}{2} + \int_{0}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt$$

```
function y=f(x)
    y=exp(-x.^2/2)/sqrt(2*%pi)
endfunction

function y=frepartition(x)
n=500
u=linspace(0,abs(x),n+1)
u=0.5*(u(1:n)+u(2:n+1))
valeurs=f(u)
integrale=sum(valeurs)*abs(x)/n
if x>=0 then
    y=0.5+integrale
else
    y=0.5-integrale
end
endfunction
```

La commande `integrate` ne donne rien de pertinent avec une borne inférieure trop petite comme -10^{99}. Vous pouvez vous convaincre que -100 suffit en tapant `X=integrate('f(x)','x',-100,1);X`

Vous pourrez ensuite comparer par exemple le résultat des commandes :

```
X=integrate('f(x)','x',-100,a);X
frepartition(a)
```

pour $a = 1$, $a = 4$, $a = -10$, et toutes les valeurs que vous voulez.

Tant que l'on reste dans le domaine « utile » (à quelques σ de 0), les résultats sont très précis avec une subdivision de taille 500. Au delà de 5σ, on gagnerait à optimiser la subdivision en « resserrant » les valeurs autour de zéro. Ce type de découpage à pas variable sort du cadre de ces TP.

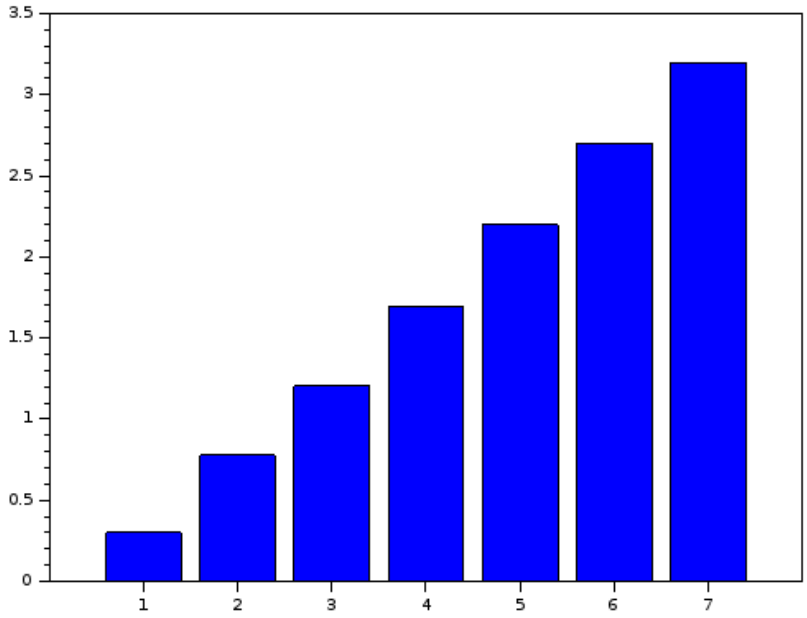
Voici le code qui permet de représenter graphiquement le nombre d'itérations nécessaires à l'obtention d'une précision donnée. Cette précision étant définie en 10^k $(1 \leqslant k \leqslant p)$, j'ai choisi de représenter `log10(S)`.

```
function y=f(x)
    y=3*x.^2
endfunction

clf()
p=input('précision maximale :')
S=zeros(1,p)
n=1
for k=1:p
    integrale=0
    while abs(integrale-1)>10^(-k)
        n=n+1
        x=linspace(0,1,n+1)
        x=0.5*(x(1:n)+x(2:n+1))
        integ=sum(f(x))/n
    end
    S(k)=n
end

bar(log10(S))
```

Noter la position du `n=1`. On ne remet pas le compteur à zéro à chaque fois, en effet, on peut supposer que pour atteindre une plus grande précision, on aura besoin d'un découpage plus fin. Pas besoin, donc, de refaire les premiers calculs. Pour $p = 7$, on obtient le graphique suivant. Pour des valeurs plus grandes, le programme tel quel peut prendre énormément de temps. On peut améliorer l'algorithme en remplaçant `n=n+1` par `n=n+1000` si `S(k-1)` dépasse une certaine valeur.



Saurez vous interpréter ce graphe[4] ? et à quoi sert le `log10`

[8.4] la croissance linéaire

Monte-Carlo

```
// approximation de pi
// par Montecarlo
// version vectorielle
n=20000
x=2*rand(1,n)-1
y=2*rand(1,n)-1
r=x.^2+y.^2 // distances à 0
proportion=sum(r<1)/n
aire=proportion*4
disp(aire)
```

Si je trouve le code précédent plus élégant, on peut envisager une version itérative dans l'optique de modifier le code pour la question 2.

```
// approximation de pi
// par Montecarlo
// version iterative
n=20000
compteur=0
for i=1:n
    compteur=compteur+(((rand()*2-1)^2+(rand()*2-1)^2)<1)
end
proportion=compteur/n
aire=proportion*4
disp(aire)
```

Pour la question 2...

```
// approximation de pi
// par Montecarlo
// nombre d'iterations avant approximation suffisante
precision=0.0001
compteur=0
nbiter=1
while abs(4*compteur/nbiter-%pi)>precision
    compteur=compteur+(((rand()*2-1)^2+(rand()*2-1)^2)<1)
    nbiter=nbiter+1
end
disp(nbiter,'le nombre d''itérations nécessaires est : ')
```

Quelques tests vous montrerons que le résultat est très fluctuant. Notamment, si on sait que la méthode converge, un bon résultat peut être momentanément « empiré » par des tests supplémentaires.

Il est certainement plus sensé de procéder à de l'échantillonnage pour évaluer la méthode, en s'arrêtant au bout d'un nombre arbitraire, mais relativement important, de constater la précision atteinte, et de répéter le processus suffisamment pour pouvoir effectuer des statistiques avec les données.

```
// évaluation de la précision de la Méthode de Monte Carlo
n=input('nombre d''itérations : ')
echantillon=[]
for i=1:100
    compteur=0
    for i=1:n
        compteur=compteur+(((rand()*2-1)^2+(rand()*2-1)^2)<1)
    end
    proportion=compteur/n
    echantillon=[echantillon,proportion*4]
end
ecart=echantillon-%pi
ecarttype=sqrt(sum(ecart.^2))
disp(ecarttype)
```

Avec un échantillon de taille 100, j'ai obtenu un écart type de l'ordre de 0,12 pour 10000 itérations, et de l'ordre de 0,03 pour 100000 itérations (ne le lancez que si vous aimez prendre un café devant votre ordinateur, c'est long...)

Moralité, on *peut* obtenir un bon résultat très rapidement, mais le nombre d'itérations nécessaires pour obtenir un intervalle de fluctuation suffisamment petit pour que le résultat soit exploitable est trop important. Aussi, on préférera une méthode numérique de type « point milieu » ou« Simpson »(très couramment utilisée).

TP9 | Intégration

Les intégrales sont partout dans les mathématiques appliquées. Certaines fonctions, même très classiques comme $x \mapsto exp(-x^2)$ n'ont pourtant pas de primitive explicite. Dans ce cas, ou dans le cas de fonctions définies par une table de valeur relevées expérimentalement par exemple, les méthodes numériques sont nécessaires pour calculer l'intégrale sur un intervalle de ces fonctions.

Il existe un grand nombre[1] de méthodes pour approcher $\int_a^b f(x)dx$. Elles diffèrent par leur complexité, leur rapidité de convergence, leur précision...

[1] cherchez par exemple *Calcul numérique d'une intégrale* sur Wikipedia pour vous faire une idée

Nous verrons ici :

9.1 Méthode des Rectangles

Le principe de l'algorithme est simple :

- on subdivise l'intervalle $[a;b]$ en $x_0 = a,\ x_1,\ \dots,x_{n-1},\ x_n = b$, par exemple avec la commande `linspace(a,b,n+1)`. Nous avons donc $n+1$ valeurs qui définissent n intervalles.
- on calcule les hauteurs des rectangles (images par f des x_i)
- on somme les aires des rectangles.

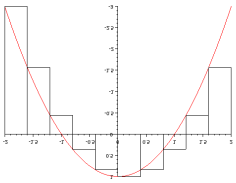


Figure 9.1: rectangles à gauche

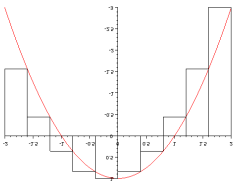


Figure 9.2: rectangles à droite

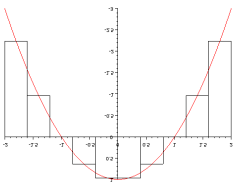


Figure 9.3: méthode du point milieu

Il y a trois manières de définir les rectangles :
1. méthode dite « rectangles à gauche »
2. méthode dite « rectangles à droite »
3. méthode dite « du point médian » qui est la plus efficace.

Un petit dessin valant mieux qu'un long discours, en voici l'illustration ci-contre où on a appliqué les méthodes de rectangles à l'application $x \mapsto x^2 - 1$:

les formules

Les formules (qui ne seront pas démontrées ici) utilisées dans l'algorithme sont :

$$\begin{aligned}\int_a^b f(x)dx &= \lim_{n\to\infty} \frac{b-a}{n}\sum_{k=0}^{n-1} f(a+k\frac{b-a}{n}) \\ &= \lim_{n\to\infty} \frac{b-a}{n}\sum_{k=1}^{n} f(a+k\frac{b-a}{n}) \\ &= \lim_{n\to\infty} \frac{b-a}{n}\sum_{k=0}^{n-1} f\left(a+(k+\frac{1}{2})\frac{b-a}{n}\right)\end{aligned}$$

Exercice 67 Saurez-vous retrouver quelle formule correspond à quelle méthode (gauche, droite, mileu ?)

La méthode du point milieu est visuellement plus équilibrée et on peut démontrer mathématiquement qu'elle est plus efficace que les deux autres.

Disons que l'on souhaite calculer $\int_0^2 e^{x^2}$.

On commence par définir la fonction `f`, *d'une manière compatible avec les vecteurs*, puis une subdivision de l'intervalle d'intégration avec `x=linspace(0,2,101)`. Nous avons donc ici $n = 100$ intervalles et un vecteur `x` comportant $n+1$ valeurs.
On modifie alors `x` pour ne garder que les valeurs utiles à la définition des « rectangles ».

- `x=x(1:100)` pour les rectangles à gauche
- `x=x(2:101)` pour les rectangles à droite
- `x=0.5*(x(1:100)+x(2:101))` pour le point milieu[2]

on peut alors calculer `y=f(x)` la valeur approchée de l'intégrale est donnée par `2*sum(y)/100`

[2] vous êtes bien convaincu que ça convient ?

Quand vous expérimentez sur l'intégration avec Scilab, vous pouvez utiliser la commande `integrate` pour vérifier vos calculs. Celle-ci n'est pas au programme mais fait appel à des algorithmes optimisés pour calculer votre intégrale.
`X=integrate('f(x)','x',0,2);X`

Voici un code facilement adaptable à tous les cas :

```
n=100000;
a=0;
b=2;
x=linspace(a,b,n+1);
x=x(1:n); // ou x=x(2:n+1)
// ou x=0.5*(x(1:n)+x(2:n+1))
y=f(x);
integrale=(b-a)/n*sum(y)
```

Pour cette fonction, positive et croissante, votre résultat sera systématiquement inférieur avec les rectangles à gauche, supérieur avec les rectangles à droite, et mieux équilibrée avec la méthode du point milieu. Dans tous les cas, l'approximation sera d'autant meilleure que n sera grand. Cela se comprend bien sur un dessin.

N'hésitez pas à tester avec des valeurs importantes comme 1000, 10000 voire 100000, le calcul reste rapide.

Exercice 68

1. Écrire une fonction Scilab qui modélise la densité de la loi normale centrée réduite $f : x \mapsto \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$

2. On rappelle que $\int_{-\infty}^{0} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx = \frac{1}{2}$, écrire une fonction Scilab qui modélise la fonction de répartition $F : x \mapsto \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt$

3. Comparer avec ce que l'on peut obtenir en utilisant la commande `integrate`

Exercice 69 Écrire un programme Scilab qui demande un nombre p en entrée, et calcule pour toutes les valeurs de $k \in [\![0; p]\!]$ le nombre d'itérations nécessaires pour arriver à une approximation de $\int_0^1 3x^2 dx$ avec une précision 10^{-k}.
On rappelle que cette intégrale vaut 1

Remarque sur la vitesse de convergence

On peut démontrer que les méthodes des rectangles à gauche et à droite ont une précision en $\frac{C}{n}$ où C est une constante dépendant de la longueur de l'intervalle d'intégration et de la fonction.
La méthode du point milieu offre quant à elle une précision en $\frac{C'}{n}$, ce qui est considérablement plus efficace.

9.2 Méthode de Monte Carlo

Monte Carlo, c'est un rocher, des princes et des princesses, un Grand Prix, un Masters, RMC, TMC et ... une méthode d'intégration numérique !

Le principe est le suivant : Disons que vous souhaitez déterminer la surface d'une mare et que pour cela vous disposez de moyens illimités (d'une imagination illimitée) qui vous permettent via un hélicoptère de disperser uniformément des cailloux blancs dans un rectangle de surface connue S englobant la mare.

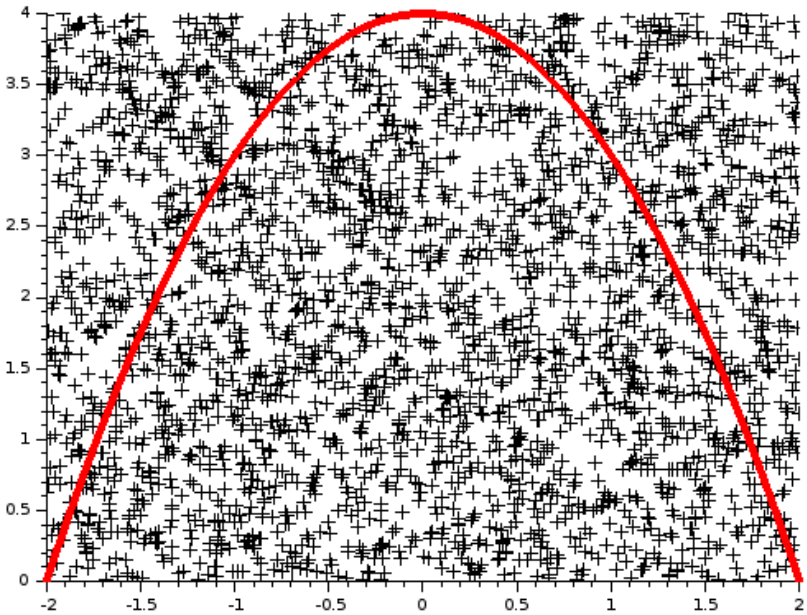
Quel est le rapport ?

Et bien justement, le rapport entre le nombre de cailloux tombés dans la mare, appliqué à la surface S donne une bonne approximation de la surface de la mare si le nombre de pierres lancées est suffisant.

On peut utiliser la même idée pour approcher la valeur d'une intégrale de fonction.

- On détermine n valeurs $(x_k; y_x)_{1 \leqslant k \leqslant n}$, les x_k suivant une loi uniforme sur $[a; b]$ et les y_k une loi uniforme sur $f([a; b])$
- Pour chaque couple de valeur $(x_k; y_k)$, on compare y_k et $f(x_k)$
- on détermine la proportion de points sous la courbe que l'on multiplie par l'aire du rectangle.

Le schéma ci-dessous illustre la méthode :



Exercice 70
On rappelle que l'équation d'un cercle de centre O et de rayon 1 est $x^2 + y^2 = 1$.

1. Écrire un programme qui utilise une méthode de type *Monte Carlo* pour évaluer la surface de ce disque (π).

2. Modifier le code pour déterminer le nombre d'itérations nécessaires pour approcher πà moins de 10^{-4} près

---

Exercice 71

On rappelle que l'équation d'un cercle de centre O et de rayon 1 est $x^2 + y^2 = 1$.

1. Écrire un programme qui utilise une méthode de type *Monte Carlo* pour évaluer la surface de ce disque (π). On pourra prendre 1000 couples de valeurs

2. Modifier le code pour déterminer le nombre d'itérations nécessaires pour approcher πà moins de 10^{-4} près

3. Modifier encore le code pour qu'il demande en entrée le nombre n de valeurs dans la Méthode, puis évalue sur 1000 simulations distinctes l'écart-type par rapport à π.

4. En remarquant que la courbe d'équation $y = \sqrt{1 - x^2}$ est celle d'un demi cercle de centre O et de rayon 1, comparer le nombre d'itérations nécessaires pour obtenir une approximation de πde précision comparable avec une méthode de rectangles sur point milieu ou méthode de Monte Carlo.

Remarque : La Méthode de Monte Carlo n'est pas particulièrement efficace pour calculer les intégrales auxquelles on peut se retrouver confronté en prépa. Elles sont toutefois concrètement utilisées pour des problèmes de dimensions plus grande, avec par exemple de nombreux paramètres comme pour des décisions stratégiques, ou en finance

9.3 Propositions de solutions

Rectangles

Ici, on calcule l'intégrale entre 0 et x, que l'on ajoute ou soustrait à $\frac{1}{2}$ en fonction du signe de x. En effet, on a :

$$\int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt = \int_{-\infty}^{0} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt + \int_{0}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt = \frac{1}{2} + \int_{0}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt$$

```
function y=f(x)
    y=exp(-x.^2/2)/sqrt(2*%pi)
endfunction

function y=frepartition(x)
n=500
u=linspace(0,abs(x),n+1)
u=0.5*(u(1:n)+u(2:n+1))
valeurs=f(u)
int=sum(valeurs)*abs(x)/n
if x>=0 then
    y=0.5+int
else
    y=0.5-int
end
endfunction
```

La commande integrate ne donne rien de pertinent avec une borne inférieure trop petite comme -10^99. Vous pouvez vous convaincre que -100 suffit en tapant `X=integrate('f(x)','x',-100,1);X`
Vous pourrez ensuite comparer par exemple le résultat de
`X=integrate('f(x)','x',-100,a);X` et de
`frepartition(a)`,
pour $a = 1$, $a = 4$, $a = -10$, et toutes les valeurs que vous voulez.

Tant que l'on reste dans le domaine « utile » (à quelques σde 0), les résultats sont très précis avec une subdivision de taille 500. Au delà de 5σ, on gagnerait à optimiser la subdivision en « resserrant » les valeurs autour de zéro. Ce type de découpage à pas non constant sort du cadre de ces TP.

Monte-Carlo

```
// approximation de pi
// par Montecarlo
// version vectorielle
n=20000
x=2*rand(1,n)-1
y=2*rand(1,n)-1
r=x.^2+y.^2 // distances à 0
proportion=sum(r<1)/n
```

```
aire=proportion*4
disp(aire)
```

Si je trouve le code précédent plus élégant, on peut envisager une version itérative dans l'optique de modifier le code pour la question 2.

```
// approximation de pi
// par Montecarlo
// version iterative
n=20000
compteur=0
for i=1:n
    compteur=compteur+(((rand()*2-1)^2+(rand()*2-1)^2)<1)
end
proportion=compteur/n
aire=proportion*4
disp(aire)
```

Pour la question 2...

```
// approximation de pi
// par Montecarlo
// nombre d'iterations avant approximation suffisante
precision=0.0001
compteur=0
nbiter=1
while abs(4*compteur/nbiter-%pi)>precision
    compteur=compteur+(((rand()*2-1)^2+(rand()*2-1)^2)<1)
    nbiter=nbiter+1
end
disp(nbiter,'le nombre d''itérations nécessaires est : ')
```

Quelques tests vous montrerons que le résultat est très fluctuant. Notamment, si on sait que la méthode converge, un bon résultat peut être momentanément « empiré » par des tests supplémentaires.
Il est certainement plus sensé de procéder à de l'échantillonnage pour évaluer la méthode, en s'arrêtant au bout d'un nombre arbitraire, mais relativement important, de constater la précision atteinte, et de répéter le processus suffisamment pour pouvoir effectuer des statistiques avec les données.

```
// évaluation de la précision de la Méthode de Monte Carlo
n=input('nombre d''itérations : ')
echantillon=[]
for i=1:100
    compteur=0
    for i=1:n
        compteur=compteur+(((rand()*2-1)^2+(rand()*2-1)^2)<1)
    end
    proportion=compteur/n
    echantillon=[echantillon,proportion*4]
end
ecart=echantillon-%pi
```

```
ecarttype=sqrt(sum(ecart.^2))
disp(ecarttype)
```

Avec un échantillon de taille 100, j'ai obtenu un écart type de l'ordre de 0,12 pour 10000 itérations, et de l'ordre de 0,03 pour 100000 itérations (ne le lancez que si vous aimez prendre un café devant votre ordinateur, c'est long...)

Moralité, on *peut* obtenir un bon résultat très rapidement, mais le nombre d'itérations nécessaires pour obtenir un intervalle de fluctuation suffisamment petit pour que le résultat soit exploitable est trop important. Aussi, on préférera une méthode numérique de type « point milieu » ou« Simpson »(très couramment utilisée).

TP 10 | Dernier TP, Bilan, exercices

Pour ce dernier TP, beaucoup de révisions, rappel des commandes que vous devez connaître, de petits morceaux de code à savoir replacer, et des exercices.

10.1 Principales Commandes

Voici la liste des commandes au programme de première année. Il faut savoir les utiliser pour écrire ou compléter un programme.

Logique, comparaisons	
opérations arithmétiques	`+ - * / ^+ - .* ./ .^`
comparaisons, tests (→`if, while`)	`== > <` `>= <= <>`
and, or	`& |`
Constantes à connaître	
π	`%pi`
e	`%e`
True (booléen)	`%T`
False (booléen)	`%F`
plus petite valeur connue par Scilab (peut utilement remplacer 0)	`%eps`

Tableau 10.1 : Nombres, constantes

Fonctions de référence	
x^n	`x^n`
$\sqrt{x}$	`sqrt(x)`
$\|x\|$	`abs(x)`
$\cos(x)$	`cos(x)`
$\sin(x)$	`sin(x)`
$\tan(x)$	`tan(x)`
$\arccos(x)$	`acos(x)`
$\arcsin(x)$	`asin(x)`
$\arctan(x)$	`atan(x)`
e^x	`exp(x)`
$\ln(x)$	`log(x)`
$\log_{10}(x)$	`log10(x)`
partie entière $\lfloor x \rfloor$	`floor(x)`
Hasard	
nombre aléatoire dans $]0;1[$	`rand()`
lois de probabilité	`grand`

Tableau 10.2 : Fonctions

Vecteurs	
vecteur ligne $u := (1;2;3)$	`u=[1,2,3]`
vecteur colonne $u := \begin{pmatrix}1\\2\\3\end{pmatrix}$	`u=[1;2;3]`
dimension d'un vecteur u	`length(u)`
Matrices	
$M := \begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix}$	`M=[1,2,3;4,5,6;7,8,9]`
${}^tM$	`M'`
M^{-1}	`inv(M)` ou `M^(-1)`
M^n	`M^n`
rang d'une matrice M	`rank(M)`
dimensions d'une matrice	`size(M)`

Tableau 10.3 : Commandes matricielles

liste des variables utilisées	`who`
efface toutes les variables	`clear`
efface la variable x	`clear(x)`

Tableau 10.4 : Quelques commandes utiles

$u = (u_1; ...u_n)$ **étant un vecteur...**	
$\sum_{k=1}^{n} u_k$	`sum(u)`
$\sum_{\substack{1 \leqslant i \leqslant n \\ i=1+3p(p \in \mathbb{Z})}} i^2$	`sum((1:3:n)^2)`
$\prod_{k=1}^{n} \sin(k)$	`prod(sin(1:1:n))`
$\prod_{k=1}^{n} u_k$	`prod(u)`

Tableau 10.5 : Sommes et produits

Pour effacer la fenêtre graphique en cours	`clf()`
Pour tracer le graphe d'une fonction f sur l'intervalle couvert par le vecteur `x`	`clf();plot(x,f(x))` `clf();plot2d(x,f(x))`
Mise en forme de données **par exemple :** `d=rand(1,10000,'normal');` **(simule un échantillon gaussien de taille 10000)**	
diagramme en barre des données x	`clf();x=-2:5;bar(x)`
diagramme en barre des données y en fonction de x	`clf();x=-2:5;y=x.^2` `bar(x,y,'red')`
	`clf();histplot(20,d)`

Tableau 10.6 : Commandes graphiques

10.2 Quelques codes à connaître

Mais attention, ce n'est pas suffisant pour être à l'abri des surprises.

Pas de rappels sur la syntaxe ou autre, aucune prétention à l'exhaustivité ici, juste une succession d'exemples, des commentaires laconiques, pour revoir l'essentiel en peu de temps.

Attention : Il n'y a ici aucune volonté d'optimisation des algorithmes proposés ici. Le parti pris est celui de la simplicité, lisibilité, transposabilité du code, souvent au détriment d'un certain « réalisme » informatique (minimisation du nombre d'opérations, gestion de la mémoire...). Mais il ne s'agit pas d'un concours d'entrée en école d'informatique...

Généralités

- Attention à l'indentation, qui fait partie des attendus.
- Écrivez avec une calligraphie propice, sans ambiguïté entre `m`, `n`....
- Utilisez les noms de variable de l'énoncé et faites en sorte que vos initiatives portent sur des noms explicites.

donc attentions aux habitudes qui sont renforcées en situation de stress

- Certains caractères ne s'écrivent pas en code, comme les lettres grecques α, γ, ε... auxquelles vous préférerez `alpha`, `gamma`, `eps`... mais aussi à certains symboles comme $\times$: qui deviennent `* /` ou 10^{-3} qui deviendra `10^(-3)` ou tout simplement `0.001` (et pas de virgule...)
- Attention aux `, ; :`
- Pensez au double `==` dans les tests `if` ou `while`
- vous pouvez commenter avec `//`

Quelques sommes

- On calcule `n!` avec `prod(1:n)` ou `prod([1:n])`
- On calcule $\sum_{k=1}^{n} k$ avec `sum(1:n)` ou `sum([1:n])`
- On calcule $\sum_{k=1}^{n} k^2$ avec `sum([1:n].^2)`
- On calcule $\sum_{k=1}^{n} \frac{1}{k}$ avec `sum([1:n].^(-1))`
- On calcule $\sum_{k=0}^{n} q^k$ avec `sum(q^[0:n])`
- On calcule $\sum_{k=0}^{n} k.q^k$ avec `sum([0:n].*q^[0:n])`

Retenez le 'point' qui assure que l'opération entre vecteurs soit réalisée composante par composante. En particulier `.^(-1)` pour les inverser les composantes du vecteur.

Quelques compléments sur les fonctions Scilab

Si l'énoncé demande « *Écrire une fonction...* », le code doit commencer par `function...` et finir par `endfunction`. Si on vous demande une *fonction récursive*, cette fonction devra s'appeler elle-même.
On peut presque tout écrire avec des fonctions si apprécie la tournure, et les appeler après pour en utiliser le résultat dans un programme.

Qu'est-ce qui différencie une fonction d'un programme

Dans les deux cas, il y a une liste plus ou moins longue d'instructions qui s'enchaînent pour réaliser une tâche. Il y a toutefois quelques différences majeures entre les deux :

- Un programme s'exécute tandis qu'une fonction s'appelle.
  Quand vous exécutez un programme, vous exécutez les instructions qu'il contient dans l'ordre où elles sont écrites. Quand on place une définition de fonction dans un code Scilab et qu'on exécute ce code, il ne se passe rien (de visible). On charge alors en mémoire une *nouvelle* fonction, qui s'ajoute à la liste des `exp` et autre `floor` fonctions de base de Scilab . Une fois chargée en mémoire, on peut appeler la fonction quand on veut.

- Un programme peut appeler une fonction, une fonction peut appeler une autre fonction, et même s'appeler elle-même dans le cas d'une fonction récursive mais une fonction n'exécutera pas de programme, a priori. Écrire une fonction peut rendre plus claire l'écriture d'un programme qui y fait référence.

- Enfin, l'autre grande nuance se joue au niveau des entrées/sorties, et plus généralement du statut des variables. Un programme travaille avec des variables globales. Toutes les variables en mémoire au moment de l'exécution du programme sont à disposition. Toutes les variables crées, ou modifiées par un programme sont accessibles après l'exécution du programme. C'est pourquoi on les appelle globales. La gestion des variables par les fonctions sera détaillée dans le paragraphe suivant.

Entrées et sorties d'une fonction

Une fonction peut demander plusieurs valeurs en entrée et renvoyer plusieurs variables en sortie. Par exemple :

```
function [a,b]=exemple(x,y,z)
   a=x.^2+y.^2
   c=x+y+z
   b=(z<sqrt(a))
endfunction
```

Ici, `c` est une variable interne. Exécuter le code pour mettre la fonction en mémoire n'affecte aucune variable du système autre que `exemple` qui est crée en tant que nouvelle fonction disponible. Pour appeler la fonction, on tape un des deux codes suivant :

```
exemple(1,2,3)
[e,f]=exemple(1,2,3)
```

La première commande attribue les valeurs 1, 2 et 3 à x, y et z, *le temps d'effectuer les calculs* nécessaires pour définir la sortie. Après exécution, x, y et z retrouvent les valeurs qu'ils avaient avant s'ils étaient définis, ou redeviennent « non définis » dans le cas contraire. Ce sont des **variables internes**. Elles ne servent que de lien. Dans l'exemple de fonction, c ne sert également qu'en interne (et pour cette fonction, ne sert à rien).

Le premier appel `exemple(1,2,3)` renvoie uniquement la valeur attribuée à a lors de l'appel de la fonction (mais a est interne également et n'a pas de valeur.

Le second appel `[e,f]=exemple(1,2,3)` calcule en interne les sorties a et b et les attribue aux variables e et f en global lors de l'appel de la fonction

À retenir : pas de `input` ou de `disp` dans une fonction.
Ces commandes ne sont présentes que dans les programmes.

Fonction récursive pour une suite

On définit la suite (u_n) par :

$$\begin{cases} u_0 = 0 \\ u_{n+1} = \frac{u_n}{2} + \frac{1}{n+1} \end{cases}$$

Écrire une fonction récursive `suite` qui prend en entrée n et renvoie u_n.

```
function u=suite(n)
    if n==0 then
        u=0
    else
        u=suite(n-1) //séparé pour un code plus lisible
        u=u/2-1/n // attention 1/n
    end
endfunction
```

Fonction récursive pour une récurrence double

Écrire une fonction `suite` qui prend un entier n en entrée et renvoie a_n où (a_n) est la suite définie par $a_0 = 1, a_1 = 2$ et pour tout $n \in \mathbb{N}, a_{n+2} = -3a_{n+1} + a_n$.

```
function u=suite(n)
    if n==0 then
        u=1
    elseif n==1 then
        u=2
    else
        u=-3*suite(n-1)+suite(n-2)
    end
endfunction
```

Fonction récursive pour calculer les coefficients du binôme

En utilisant l'identité $\binom{n}{k} = \frac{n}{k}\binom{n}{k}$, écrire une fonction `binom` qui à deux entiers n, k vérifiant $0 \leqslant k \leqslant n$, associe $\binom{n}{k}$.

```
function b=binom(n,k)
    if k==0 then
        b=1
    else
        b=(n/k)*binom(n-1,k-1)
    end
endfunction
```

10.3 Convergences

Déterminer le plus petit entier n...

On reprend la suite (u_n) par $u_0 = 0$ et $u_{n+1} = \frac{u_n}{2} + \frac{1}{n+1}$.
On suppose programmée une fonction `suite` qui calcule u_n quand on lui donne n en entrée et on admet que $\lim_{n\to+\infty} n.u_n = 2$.

Écrire un programme qui détermine le plus petit entier n tel que $|nu_n - 2| < 10^{-3}$

```
n=0
while abs(n*suite(n)-2)>=0.001
    n=n+1
end
disp(n)
```

L'essentiel des programmes de type *while* avec une suite enregistrée dans une fonction `suite` peuvent se coder ainsi.

Quelques remarques :

- Si le problème parle de deux suites (u_n) et (v_n) adjacentes, et que la valeur de la limite n'est pas connue, on peut utiliser comme test d'arrêt `v-u > precision` on peut alors prendre comme valeur approchée à `precision` près de la limite la moyenne `(u+v)/2` des termes de la suite auxquels on est arrivés.
- si votre suite est monotone, on peut omettre le `abs`

Une somme

Écrire une fonction qui demande un nombre n et un nombre x puis affiche $S_n = \sum_{k=0}^{n} \frac{x^n}{n!}$

```
function s=somme(x,n)
    if n==0 then
```

```
        s=1
    else
        s=x^n/prod(1:n)+somme(x,n-1) // prod(1:n) calcule n!
    end
endfunction

n=input('Entrez un nombre n : ')
x=input('Entrez un nombre x : ')
disp(somme(x,n))
```

10.4 Simulation de variables aléatoires classiques

Les commandes suivantes renvoient une matrice $m \times n$ dont les coefficients suivent la loi ...

- Loi uniforme entière : `grand(m,n,"uin",n,p)` coefficients dans $[\![n;p]\!]$.
- Loi binomiale $\mathcal{B}(N;p)$: `grand(m,n,"bin",N,p)`
- Loi de Bernoulli de paramètre p : $\mathcal{B}(1;p)$: `grand(m,n,"bin",1,p)`
- Loi géométrique $\mathcal{G}(p)$: `grand(m,n,"geom",p)`
- Loi de Poisson $\mathcal{P}(\lambda)$d'espérance λ : `grand(m,n,"poi",lambda)`
- Loi uniforme continue sur $[a;b[$: `grand(m,n,"unf",a,b)`
- Loi exponentielle $\mathcal{E}(\lambda)$: `grand(m,n,"exp",1/lambda)` dernier paramètre = espérance
- Loi normale $\mathcal{N}(\mu,\sigma^2)$: `grand(m,n,"nor",mu,sigma)`

Pour simuler des lois non-standards, il peut être utile de savoir simuler « à la main » certaines de ces variables aléatoires, notamment les discrètes. On peut utiliser `rand()` qui renvoie un nombre aléatoire de $[0;1[$ que l'on peut comparer à la probabilité p pour simuler un succès/échec.
Toutes ces commandes peuvent au besoin être intégrées dans une fonction Scilab.

- simuler une épreuve de Bernoulli de paramètre p. `X=(rand()<p)` renvoie 1 avec une probabilité p et 0 sinon.
- simuler n épreuves de Bernoulli de paramètre p. `X=(rand(n,1)<p)` renvoie un vecteur aléatoire
- simuler $\mathcal{B}(n;p)$: `X=sum(rand(n,1)<p)`
- simuler une loi géométrique :

```
X=1
while (rand()>=p)
    X=X+1
end
```

- simuler une loi uniforme sur $[a;b]$: `X=a+(b-a)*rand()`

Après chacun de ces codes, on peut afficher la valeur du résultat obtenu avec `disp(X)` mais il peut être plus avantageux de créer une fonction Scilab qui simule la variable aléatoire.

10.5 Résolution d'équations, dichotomie

La dichotomie n'est pas la seule méthode, mais les autres méthodes, par des suites du type $u_{n+1} = f(u_n)$ découlent d'éléments vus précédemment. La résolution approchée par dichotomie est un algorithme à part, que nous allons illustrer avec l'approximation de la solution de l'équation $(x^2+1)\,e^{-x} - x = 0$ dont on peu prouver qu'elle admet une unique solution dans comprise entre 0,5 et 1. L'algorithme suivant en donne une valeur approchée à 10^{-6} près :

```
function y=f(x)
    y=(x^2+1)*exp(-x)-x
endfunction

a=0.5
b=1
while b-a>=10^(-6)
    c=(a+b)/2
    if f(a)*f(c)>0 then
        a=c
    else
        b=c
    end
end

disp(a)
```

10.6 Exercices

Exercice 72 : EML 2010 (ECS)
On note $f : \mathbb{R} \to \mathbb{R}$ l'application définie par : $\forall x \in \mathbb{R},\ f(x) = x - \ln(1+x^2)$
On considère la suite la suite (u_n) définie par $u_0 = 1$ et $u_{n+1} = f(u_n)$.

1. Écrire un programme qui calcule et affiche le plus petit entier n tel que $u_n \leqslant 10^{-3}$

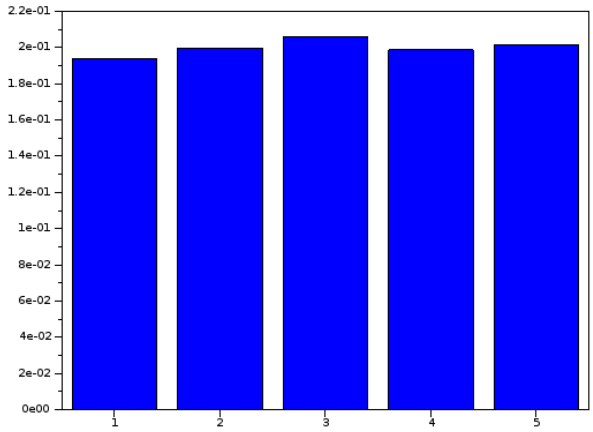
2. a. Établir $\forall x \in [0;1],\ f(x) \leqslant x - \frac{x^2}{2}$
   b. Déduire $\forall n \in \mathbb{N},\ u_n^2 \leqslant 2(u_n - u_{n+1})$
   c. Montrer que la série $\sum_{n\geqslant 0} u_n^2$ converge et que
   $\forall n \in \mathbb{N},\ \sum_{k=n+1}^{+\infty} u_k^2 \leqslant 2u_{n+1}$
   d. Écrire un programme qui calcule et affiche une valeur approchée de $\sum_{n=0}^{+\infty} u_n^2$ à 10^{-4} près.

Exercice 73 L'urne U_1 contient N boules : $(N-1)$ boules blanches et 1 boule noire. On effectue des tirages sans remise dans l'urne U_1 , jusqu'à l'obtention de la boule noire. On note X la variable aléatoire qui prend pour valeur le nombre de tirages nécessaires pour l'obtention de la boule noire.

1. On simule 10000 fois cette expérience aléatoire. Recopier et compléter le programme Scilab suivant pour qu'il affiche l'histo- gramme donnant la fréquence d'apparition du rang d'obtention de la boule noire :

```
N=input('Donner un entier naturel non nul');
S=zeros(1,N)
for k=1:10000
   i=1;
   M=N;
   while _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
      i=i+1
      M= _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
   end
   S(i)=S(i)+1;
end
disp(S/10000)
bar(S/10000)
```

2. On exécute le programme complété ci-dessus. On entre 5 au clavier et on obtient l'histogramme suivant :



Quelle conjecture pouvez-vous émettre sur la loi de la variable aléatoire X ?

Exercice 74

On lance n fois une pièce de monnaie équilibrée, on note Y la variable aléatoire qui vaut 0 si *pile* tombe au plus une fois, et qui dans le cas contraire prend pour valeur le rang du deuxième *pile*.

1. Écrire une fonction Scilab nommée `varY` permettant de simuler la variable aléatoire Y

2. Écrire une fonction Scilab nommée `table` prenant en entrée un entier n correspondant au nombre de tirages, et qui renvoie un vecteur contenant le nombre d'occurrence de chaque valeur. (la première coordonnée contiendra $\mathbb{P}(Y=0)$, la seconde $\mathbb{P}(Y=2)$ ... la k-ième $\mathbb{P}(Y=k)$

3. En déduire une approximation de $\mathbb{E}(Y)$

4. Afficher un diagramme en barres présentant les fréquences d'apparition de chaque valeur.

penser à `usecanvas(%T)`

Index

Index des commandes