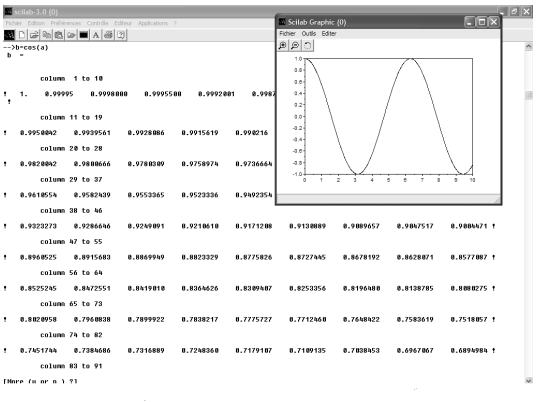
Quelques algorithmes mathématiques.



Utilisation du logiciel Scilab

Année scolaire 2006/2007

Table des matières

Avant-propos

En classe de troisième deux algorithmes ont été rencontré : l'algorithme de la différence et l'algorithme d'Euclide, tout les deux permettant de trouver le pgcd de deux entiers.

Mais qu'est-ce qu'un algorithme[1] ?

Un algorithme est un ensemble d'opérations que l'on répètent jusqu'à obtenir le résultat cherché. Par exemple l'algorithme d'Euclide est une répétition de divisions euclidiennes, ceci jusqu'à obtenir un reste égale à 0. Le pgcd cherché est alors le dernier diviseur.

De nombreux algorithmes existent en mathématiques, et bien souvent le nombre d'opérations qu'ils demandent est très élevé : de l'ordre de la centaine, voir du millier, voir encore plus... L'informatique devient alors indispensable.

Le but de ce fichier est donc de se familiariser avec les notions d'algorithmes et de programmation. Nous utiliserons pour cela le logiciel Scilab, logiciel distribué gratuitement par l'INRIA.

La première étape de ce manuel sera donc de télécharger et d'installer Scilab.

[1] cf annexe pour l'éthymologie du mot "algorithme"

Chapitre 1

Prise en main du logiciel Scilab

1.1 Installation de Scilab

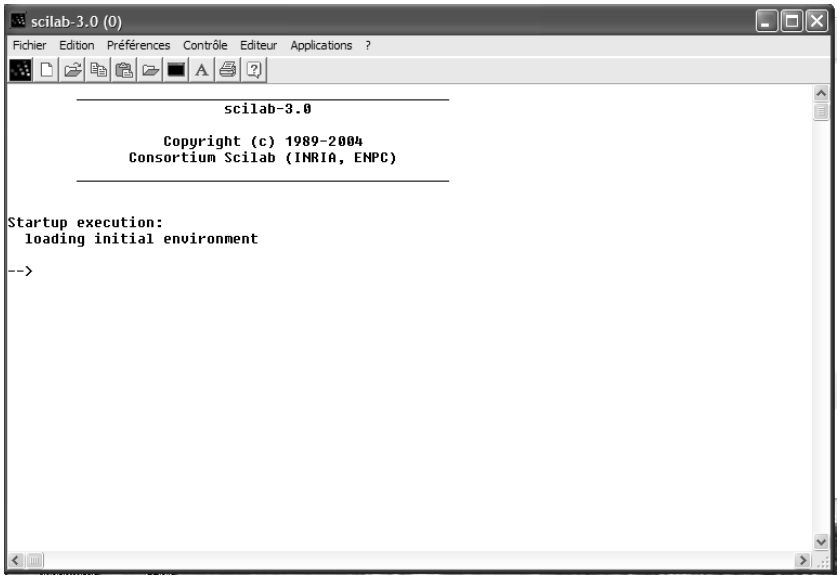
L'INRIA distribue gratuitement le logiciel Scilab qui se télécharge à partir de son site internet :

http ://scilabsoft.inria.fr

Il suffit alors de cliquer sur "Télécharger Scilab 4.1", d'enregistrer le fichier et ensuite de l'éxécuter.

1.2 Première utilisation

Au lancement du logiciel la fenêtre suivante apparaît :

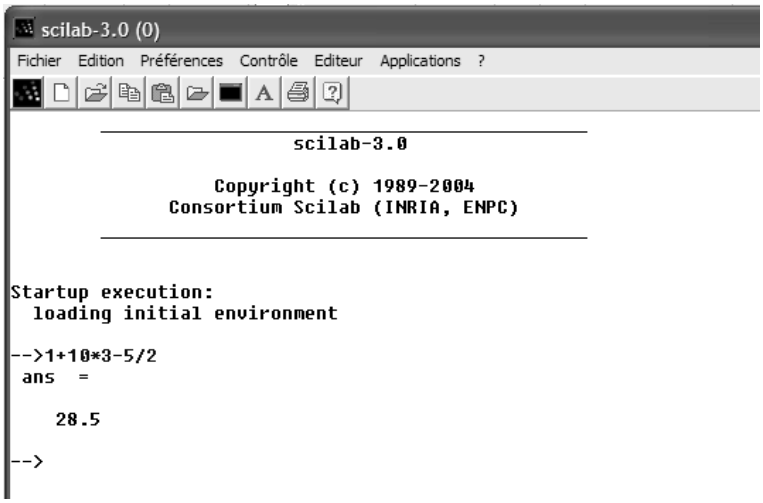


C'est la fenêtre principale d'éxécution. Toute opération, tout lancement d'algorithme (de fonction en fait) se fera à partir d'ici. C'est un peu l'équivalent de l'écran d'une calculatrice, avec plus de possibilité en fait.

1.2.1 Faire des calculs

Pour faire des calculs rien de plus simple, il vous suffit de taper l'opération voulu dans cette fenêtre, et d'appuyer sur la touche "Entrée".

Par exemple pour calculer "$1 + 10 \times 3 - \frac{5}{2}$", on écrira : "1+10*3-5/2", et on observe :



On remarquera d'abord, que les opérations usuelles se notent +, -, * et / pour respectivement l'addition, la soustraction, la multiplication et la division. Nous voyons ensuite que Scilab ne fournit que des résultats sous forme décimale et non fractionnaire.

Taper maintenant l'opération : *1/3*, puis sur la touche "Entrée".

Taper ensuite l'instruction *format(20)*, taper à nouveau *1/3* et sur la touche "Entrée" et observer la différence avec ce qui précéde.

Une autre remarque et que l'on aurait pu ne pas retaper *1/3*, mais taper deux fois sur la touche du clavier représentant la flèche vers le haut.

1.2.2 Racines carrées, cosinus, sinus, tangente, le nombre π

Pour calculer une racine carrée l'instruction est "sqrt".

Taper par exemple *sqrt(2)*.

Pour les cosinus, sinus et tangente, tout est intuitif : cos(0), sin(0) et tan(0), pour les cosinus, sinus et tangente de 0.

Pour voir ou calculer avec le nombre π, il faut écrire *%pi*.

Exercice 1
A l'aide de Scilab calculer : $cos(\frac{\pi}{3})$, $cos(\frac{\pi}{2})$, $cos(\frac{\pi}{4})$, $cos(\pi)$, $sin(\frac{\pi}{3})$, $sin(\frac{\pi}{2})$, $tan(\frac{\pi}{3})$, $tan(\frac{\pi}{4})$, $tan(\frac{\pi}{5})$

1.2.3 Variables

L'un des principes fondamentaux de la programmation et de l'algorithmique est d'utiliser des variables. Une variable est en fait une case mémoire de l'ordinateur à laquelle on attribue une valeur. Généralement on nomme les variables à l'aide d'une lettre.

Taper dans Scilab : *a=2*.

En faisant cela, le logiciel Scilab vient de choisir dans votre ordinateur une case mémoire qu'il appelle alors "a" et dans laquelle il stocke la valeur 2.

Ce processus là, ne s'appelle pas l'égalité, mais **l'affectation**. On prendra garde à bien retenir que le symbole "=" sous Scilab ne signifie pas l'égalité des mathématiciens.

Pour bien comprendre cela, taper l'instruction :*a=a+1*.

Le résultat observé est "3".

Pour Scilab l'instruction *a=a+1* veut dire : "Dans la case mémoire a tu stockes la valeur qu'il y a actuellement en a (c'est à dire 2) et tu lui ajoutes 1." On obtient donc 3...

On remarquera que l'équation mathématiques "a=a+1" est équivalente à "0=1", équation qui ne possède aucune solution. Donc pour Scilab, "a=a+1" est bien autre chose que des mathématiques ; c'est une affectation !

Pour un peu mieux comprendre le processus d'affectation, faisons les quelques manipulations suivantes :

Taper : *a=2*, puis sur "Entrée".

Taper : *b=5*, puis sur "Entrée".

Taper : *a*b*, et observer le résultat en tapant sur "Entrée".

On retiendra donc que les variables permettent de stocker dans une partie de l'ordinateur des informations. Pour Scilab se sera des nombres (ou des tableaux contenant des nombres comme nous le verrons ensuite).

Exercice 2 (difficile)
Soit a et b deux cases mémoires de l'ordinateur dans lesquelles sont stockées des nombres que nous ne connaissons pas.
Trouver un processus pour inverser les contenus de a et b.
L'explication de ce processus sera à envoyer par mail pour correction.

1.2.4 Tracer des graphiques

Scilab, tout comme les calculatrices permet de tracer des graphiques. Nous allons ici tracer les courbes représentatives de certaines fonctions.

Pour cela commençons par créer un tableau de nombres, qui sera en fait l'ensemble des abscisses des points de qui on caculera les images par la fonction.

Taper : *x=[-10 :0.01 :10]*, puis sur "Entrée".

Cette instruction a en fait affecté à la variable x tous les nombres de -10 jusqu'à 10 en avançant de 0,01 en 0,01.

Taper : *y=x^2*, puis sur "Entrée".

Cette instruction affecte à la variable y les nombres contenus dans la variable x, mais élevés au carrée.

Taper : *plot(x,y)*, puis sur "Entrée".

Scilab trace alors un graphique avec pour abscisses les nombres stockés dans x et pour ordonnées les nombres stockés dans y. En fait le premier point aura pour abscisse le premier nombre de x et pour ordonnée le premier nombre de y, le deuxième point aura pour abscisse le deuxième nombre de x et pour ordonnée le deuxième nombre de y, etc.

Exercice 3
Tracer les courbes représentatives des fonctions inverse et racine carrée.
Ces deux graphiques seront à sauvegarder et à envoyer par mail pour correction.

1.2.5 Créer des fonctions

Nous allons écrire et enregistrer de petits programmes à l'aide du langage Scilab. Pour cela il va falloir se créer un répertoire où tous ces fichiers seront stockés. Il est conseillé de créer donc ce nouveau dossier dans le répertoire "scilab-3.0" et de l'appeler par exemple "programmes".

Une fois cela fait nous allons créer notre première fonction, et pour l'écrire nous allons ouvrir l'éditeur de Scilab. Il suffit d'écrire l'instruction "scipad" dans la fenêtre d'éxécution de Scilab. L'éditeur s'ouvre alors.

Dans l'éditeur taper les instructions suivantes :

function y=f(x)

*y=3*x-5*

Cliquer alors sur "fichier", "enregistrer sous", et sauvegarder ce fichier dans le répertoire créer à cet effet. On remarquera que Scilab propose d'appeler ce fichier "f.sci". On le sauvegardera ainsi.

Pour éxécuter cette fonction, on va retourner dans la fenêtre d'éxécution de Scilab. Dans "fichier", sélectionner "changer de répertoire" et choisir le répertoire dans lequel la fonction précédente a été sauvée.

Il va falloir maintenant la rentrer dans la mémoire de Scilab. Pour cela taper l'instruction "exec f.sci".

Il ne reste plus qu'à taper par exemple : *f(2)* puis sur "Entrée", de même avec *f(10)*, ou n'importe qu'elle autre nombre. On remarquera bien que le résultat de l'instruction f(2) est bien 1.

Essayons maintenant de comprendre la structure d'écriture d'une fonction.

Observons à nouveau la liste d'instruction de la fonction f.sci :

function y=f(x)

*y=3*x-5*

Le mot *function* veut dire que ce que l'on est entrain de créer est une fonction. C'est à

dire un objet qui a un (ou plusieurs) antécédents et qui fournit un ou plusieurs résultats. (On remarquera la différence avec une fonction mathématique...)

Les instructions *y=f(x)* veulent dire que la fonction s'appele f, que la variable s'appelle x et que le résultat est stocké dans la variable y.

Ce qui se trouve après tout cela est le corps de la fonction : c'est un ensemble d'instructions (ici très court, mais il peut être très long) qui fait des calculs à partir de la variable (ici x) et qui après tous ces calculs stocke un résultat dans y. (ici y=3*x-5)

Il n'y a donc aucune surprise à voir que l'instruction f(2) nous fournit comme résultat 1.

Exercice 4

Créer une fonction g qui pour tout nombre x nous fournit le résultat : $2x^2 - 3x + 4$.
Créer une fonction h qui pour tout nombre t nous fournit le résultat : $t\sqrt{t-1} + 3$.
Ces deux fonctions seront à envoyer par mail pour correction.

Chapitre 2

Conditions et boucles

2.1 Les conditions : la commande *if*

En informatique nous avons très souvent à faire des tests entre diverses variables. Par exemple, savoir si une variable est plus grande ou plus petite que le nombre 100, savoir si la précision d'un résultat est suffisante, et encore bien d'autres choses. Ainsi, dans tout les langages de programmation se trouve une fonction de test : ce sera le plus souvent, comme dans la Scilab la fonction *if*.

2.1.1 Syntaxe de la commande *if*

De manière générale pour Scilab la syntaxe sera :

if "condition" *then* "instruction" ; *end*.

Ce qui veut dire que "si" la condition est vérifiée "alors" (*then*) Scilab exécute les instructions demandées.

Voyons cela sur un exemple :

Dans la fenêtre d'éxécution taper : *v=5*, puis taper sur "Entrée".

Taper ensuite : *if v==5 then disp("'c"est vrai")*;*end*, puis taper sur "Entrée".

Ou encore taper : *if v=<154 then disp("v est bien plus petit que 154...")*;*end*

Dans le premier cas la condition est "v==5" : Scilab regarde donc si dans la variable v est stocké le nombre 5. Et si c'est vrai alors Scilab affiche à l'écran (c'est la commande *disp()*) la chaine de caractère "c'est vrai". A remarquer que le texte à écrire est écrit entre guillemets et que l'apostrophe doit être doublé.

Le deuxième cas est structuré de la même manière, à ceci près que l'on ne teste pas une égalité (==) mais une inégalité (=< inférieur ou égale).

2.1.2 Les tests

Voici les symboles pour tester une égalité ou une inégalité :

égale	différent	supérieur strict	supérieur ou égale	inférieur strict	inférieur ou égale
==	<>	>	>=	<	=<

2.1.3 La commande *else*

A la suite d'un test effectué avec la commande *if* on peut continuer en rajoutant la commande *else* (sinon). Voyons cela de manière générale :

if "condition" *then* "instruction 1"*else* "instruction 2" ;*end*

Ce qui se comprend par : si telle condition est vérifiée alors exécuter l'instruction 1 sinon (si la condition n'est pas vérifiée) exécuter l'instruction 2.

Voyons cela sur un exemple assez explicite.

Dans la fenêtre d'éxécution taper : *a=rand(1)*

Cela nous donne un nombre au hasard entre 0 et 1.

Taper ensuite : *if a>0.5 then disp("C"est plus que la moitié") ; else disp("C"est moins de la moitié") ;end*

On peut éxécuter à nouveau ces instructions pour obtenir d'autres valeurs pour le nombre a.

Observons une nouvelle commande. Taper : *a=input("Quelle est votre âge ?")*

Cette commande demande à l'utilisateur d'écrire un nombre et ensuite d'appuyer sur la touche "Entrée". Le nombre qu'écrit l'utilisateur sera stocké dans la variable a.

Exercice 5
A l'aide des commandes *input*, *if* et *else*, écrire dans l'éditeur un programme qui demande l'âge de l'utilisateur et qui, suivant la réponse donnée, écrit à l'écran "trop jeune" ou "trop vieux".
Ce programme sera appelé test.sce, et pour être éxécuter par Scilab il faudra écrire dans la fenêtre d'éxécution : exec test.sce
Ce programme sera envoyé par mail pour correction.

2.2 Les boucles

Tout langage de programmation possède également d'autres objets autant essentiels que les test : les boucles. Il en existe deux types, les boucles *while* et les boucles *for*.

2.2.1 Les boucles *while*

Observons la structure générale d'une boucle *while* :

while "condition" *do* "instruction" ; *end*

Ce que nous pouvons traduire par : tant que la condition est vérifiée on exécute les instructions. Et pour savoir jusqu'où vont les instructions on doit écrire un *end*. Une fois que Scilab

lit le *end* il retourne là où est écrit le *while* et regarde si la condition est remplie ou non. Si elle l'est alors Scilab refait les instructions, sinon il passe les instructions et sort de la boucle *while*.

Taper les instructions suivantes dans la fenêtre d'éxécution, (taper sur "Entrée" après chacune d'elles) :

p=0.2
y=rand(1)
compteur=1
while y>=p do compteur=compteur+1 ; y=rand(1) ; end
compteur

Faisons une petite traduction de cela :

p=0.2	On stocke le nombre 0,2 dans la variable p.
y=rand(1)	On stocke un nombre pris au hasard entre 0 et 1 dans la variable y.
compteur=1	On stocke le nombre 1 dans la variable compteur.
while y>=p do compteur=compteur+1 ; compteur=compteur+1 ; y=rand(1) ;end	"tant que" y est plus grand que p "faire" : rajouter 1 au compteur, reprendre un nombre au hasard entre 0 et 1 pour y.
compteur	Affiche la valeur du compteur. En lisant cette valeur on saura combien de fois Scilab a emprunté la boucle *while*.

Voici un autre exemple à taper dans le Scipad, et à enregistrer par exemple sous le nom de question.sce.

text="aaa" ;

while text<>"juventus" do
text=input("Quelle est la meilleur équipe de foot ? (écrire votre réponse entre guillements)") ;
if text<>"juventus" then disp("Mauvaise réponse...") ;end ;
end ;
disp("Bonne réponse")

Dans la fenêtre d'éxécution de Scilab écrire : *exec question.sce* (après s'être placé dans le répertoire qui contient le programme question.sce) et taper sur "Entrée".

On essaiera alors de bien comprendre le fonctionnement de ce programme, pour pouvoir faire l'exercice suivant.

Exercice 6
Ecrire un programme (dans le Scipad) qui demande l'âge de l'utilisateur et tant que l'âge répondu n'est pas compris entre 20 et 30, le programme repose la question.
Ce programme sera à envoyer par mail pour correction.

2.2.2 Les boucles *for*

Les boucles *for* sont légérement différentes des boucles *while*. Leur structure générale est la suivante :

for variable=nombre1 : nombre2 do "instruction" ; end ;

Voyons cela sur un exemple pour mieux comprendre.
Créer dans le Scipad un programme bouclefor.sce en tappant ce qui suit :

for i=1 :10 do disp (i) ;end ;

Exécuter et observer ce qui ce qui apparaît à l'écran.
Décrivons maintenant le fonctionnement d'une boucle *for*. Une telle boucle va en fait donner successivement toutes les valeurs comprises entre nombre1 et nombre 2 à la variable. Dans notre exemple i va prendre successivement toutes les valeurs de 1 jusqu'à 10. Et pour chaque valeur de i la boucle va éxécuter les instructions. Dans notre exemple la boucle fait écrire 10 fois la valeur de i à l'écran, et comme i vaut 1, puis 2, puis 3, ... puis 10, nous voyons s'afficher à l'écran les nombres 1, 2, 3, ... jusqu'à 10.

Pour un peu mieux comprendre les possibilités qu'offrent les boucles *for* recopier dans le Scipad la fonction suivante, que vous sauvegarderez en la nommant facto.sci :

function y=facto(n)
y=1 ;
*for i=1 :n do y=y*i ;end ;*
endfunction

Exécuter cette fonction, et lancer là pour des valeurs de n assez petites : taper par exemple dans la fenêtre d'éxécution : $facto(3)$, puis $facto(5)$ ou encore pour d'autres valeurs entières de votre choix.

Explication de la fonction facto.sci :
Lorsqu'on éxécute $facto(6)$, par exemple, observons ce fait Scilab :

y=1	la variable y prend la valeur 1
for i=1 :n do y=y*i ;end ;	pour i=1, y=y× 1, donc y=1. pour i=2, y=y× 2, donc y=2. pour i=3, y=y× 3, donc y=6. pour i=4, y=y× 4, donc y=24. pour i=5, y=y× 5, donc y=120. pour i=6, y=y× 6, donc y=720.
endfunction	fin de la fonction facto

Exercice 7

Créer une fonction Scilab qui prend en variable un entier n et qui renvoie un nombre y valant la somme de tous les entiers de 1 jusqu'à n. Cette fonction sera sauvegardée sous le nom de somme.sci et envoyer par mail pour correction.

Chapitre 3

Quelques algorithmes

Ce qui précéde va nous permettre maintenant de programmer des algorithmes, il est donc essentiel de savoir manipuler les tests avec la fonction *if*, les boucles avec les instructions *while* et *for*.

3.1 Le jeu du "plus grand/plus petit"

3.1.1 Les instructions *rand* et *floor*

L' instruction *rand* nous fournit un nombre aléatoire. Par exemple *rand(1)* nous donne un nombre pris au hasard entre 0 et 1.

Ainsi si on tape, dans la fenêtre d'éxécution *100*rand(1)+1*, on obtient un nombre décimal compris entre 1 et 100.

A l'aide de l'instruction *floor* nous allons obtenir un nombre entier. En fait *floor(x)* nous fournit la partie entière de x. A savoir que la partie entière d'un nombre réel x est le nombre entier inférieur à x qui lui est le plus proche.

Taper par exemple *floor(2.8)*.

Ainsi si on tape *floor(100*rand(1)+1)* on obtient un nombre entier compris entre 1 et 100.

3.1.2 Le jeu en lui-même

Imaginons un jeu entre deux personnes. La première pense, en silence, à un nombre compris entre 1 et 100. La deuxième essaie de deviner ce nombre et propose une solution. La première personne lui répond alors "plus grand", "plus petit" ou "gagné", cela jusqu'à ce que le nombre soit deviné. Le but étant de trouver le nombre le plus vite possible.

Exercice 8
Programmer ce jeu en Scilab, la première "personne" qui pense en silence à un nombre étant l'ordinateur, la deuxième l'utilisateur. Par contre, on ne cherchera pas un nombre entre 1 et 100, mais entre 1 et 10 000.
Les commandes Scilab à utiliser sont *rand, floor, while, if, input et disp.*
Ce programme sera à envoyer par mail pour correction.

3.2 La dichotomie et l'approximation des racines carrées

L'une des difficultés en mathématiques, pendant des siècles a été de connaitre les valeurs approchées des racines carrées. On imaginera la difficulté à connaitre la valeur approchée de $\sqrt{46783}$ sans calculatrice ni ordinateur.

3.2.1 L'équation $x^2 - a = 0$

Soit a un entier naturel non nul. Considèrons l'équation $x^2 - a = 0$. L'une de ces solutions (la positive) est $\sqrt{a}$. Par ailleurs si x est proche du nombre $\sqrt{a}$ alors le nombre $x^2 - a$ sera proche de 0.

Plus précisemment, considérons le nombre $P(x)$ tel que $P(x) = x^2 - a$, et regardons ce que vaut ce nombre pour x proche de $\sqrt{a}$, mais plus petit que $\sqrt{a}$.

En fait dans ce cas précis (x proche de $\sqrt{a}$, mais plus petit que $\sqrt{a}$) $P(x)$ sera négatif. Dans l'autre cas (x proche de $\sqrt{a}$, mais plus grand que $\sqrt{a}$) $P(x)$ sera positif.

Et de manière encore plus générale, nous avons pour x positif que :

Si $P(x) < 0$ alors $x < \sqrt{a}$.
Si $P(x) > 0$ alors $x > \sqrt{a}$.

Grace à ce constat nous allons établir une méthode pour trouver une valeur approchée de $\sqrt{a}$: la dichotomie.

Considérons l'intervalle I=[1 ;a]. On sait que pour $a \geq 1$ le nombre $\sqrt{a}$ est un nombre de l'intervalle I. Remarquons que le centre de l'intervalle est le nombre $m=\frac{a+1}{2}$.

Le nombre $\sqrt{a}$ est-il dans la partie gauche de l'intervalle (entre 1 et m) ou est-il dans la partie droite (entre m et a) ?

Pour le savoir il suffit de calculer le nombre $P(m)$, avec $P(x) = x^2 - a$. Si $P(m)$ est négatif alors cela veut dire que m est plus petit que $\sqrt{a}$ donc que $\sqrt{a}$ est dans la partie droite de l'intervalle I. Si $P(m)$ est positif cela veut dire que m est plus grand que $\sqrt{a}$ et donc que $\sqrt{a}$ est dans la partie gauche de l'intervalle I.

On trouve donc un autre intervalle où l'on sait que se situe le nombre $\sqrt{a}$, l'avantage étant que ce nouvel intervalle est plus petit que I. Il est même deux fois plus petit.

Pour améliorer la précision on va donc appliquer à nouveau cette méthode, non pas à l'intervalle I, mais au nouveau qui est la moitié de I. Et on fera cela avec des intervalles de plus

en plus petit, jusqu'à obtenir un intervalle très petit qui nous donnera une précision suffisante pour la valeur approchée de $\sqrt{a}$.

Voici une idée d'algorithme pour cette méthode, algorithme écrit en langage courant et non en langage Scilab.

Algorithme	*Commentaires*
fonction y=racine(a)	*On déclare le nom de la fonction le résultat sera stocké dans y.*
gauche=1	*La borne gauche de l'intervalle vaut 1*
droite=a	*la borne droite de l'intervalle vaut a*
milieu=$\frac{a+1}{2}$	*le nombre m est le milieu de l'intervalle [droite ;gauche]*
pre=a-1	*On appelle pre la longueur de l'intervalle, qui sera la précision.*
tant que pre>0,0000000000001 faire	*tant que la précision n'est pas suffisante faire :*
si p(m) est positif alors gauche=gauche, droite=m	*on change d'intervalle*
si p(m) est négatif alors gauche=m, droite=droite	
pre=droite-gauche	*on change donc de précision*
m=$\frac{droite+gauche}{2}$	*et aussi de milieu*
fin de boucle	
y=m	*après tant de boucles le milieu de l'intervalle est assez proche de* $\sqrt{a}$

Exercice 9
Programmer l'algorithme de la fonction racine en Scilab. Cette fonction sera à envoyer par mail pour correction.

3.3 Une méthode plus performante pour approcher les racines carrées : la méthode de la sécante

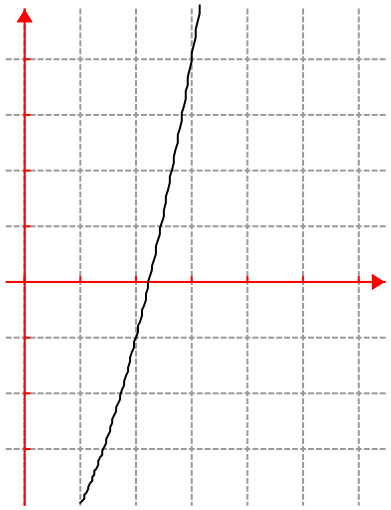
Dans la fonction racine précédente on pourrait insérer un compteur et regarder combien de fois la boucle s'est éxécutée. Et on verrait que ce compteur affiche un nombre assez élevé. On peut faire beaucoup mieux. Il existe d'autres méthodes plus performantes que la dichotomie qui nous permettent de faire "moins de boucles" et donc d'obtenir bien plus rapidement une bonne approximation d'une racine carrée. En particulier la méthode des sécantes.

3.3.1 Description de la méthode

En informatique, surtout pour obtenir des valeurs approchées avec de nombreuses décimales on ne peut pas éviter d'utiliser les mathématiques (désolé...). Et au plus on veut une grande

précision et au plus les mathématiques utilisées risque d'être complexes.

Considérons la fonction f définie sur l'intervalle I=[1 ;a] par $f(x) = x^2 - a$, et on observons sa représentation graphique dans un repère orthonormé :

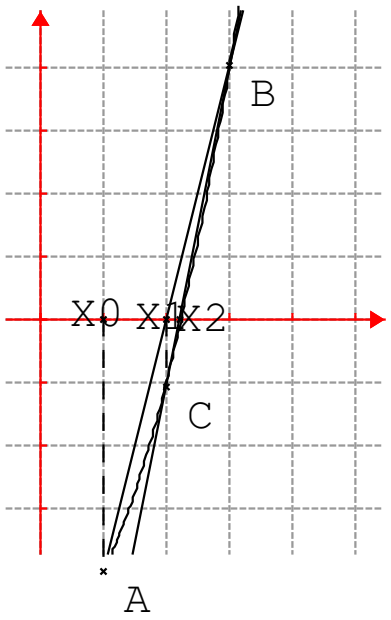


Nous voyons donc que le point d'intersection entre la courbe et l'axe des abscisses est le point d'abscisse $\sqrt{a}$.

Plaçons maintenant sur l'axe des abscisses le point x_0 d'abscisse 1. Plaçons le point A de la courbe d'abscisse x_0. Plaçons également le point B de la courbe d'absisse a, et traçons le segment [AB].

Ce segment coupe l'axe des absicces en un point x_1.

Plaçons maintenant le point C de la courbe d'abscisse x_1 et traçons le segment [CB]. Celui-ci coupe l'axe des abscisses en un point x_2.



Nous voyons alors que les points x_2 et $\sqrt{a}$ sont presque confondus.

Mathématiquement parlant, on sait calculer l'équation de la droite (AB), et donc trouver la valeur x_1.

Une fois que l'on connait x_1 on peut trouver l'équation de la droite (CD) et on pourra connaitre alors x_2.

A partir de là, on peut en fait continuer la méthode, c'est à dire trouver un point x_3, puis x_4, puis x_5, etc. En fait tous ces x_n seront de plus en plus proche de $\sqrt{a}$. C'est ce que l'on appelle la méthode de la sécante.

Exercice 10 (très difficile)
Programmer une fonction Scilab correspondant à la méthode des sécantes.
Cette fonction sera à envoyer par mail pour correction.

Exercice 11
Trouver un procédé pour voir dans quelle mesure la méthode des sécantes et plus performante que la dichotomie.
Ce "procédé" sera expliquer par mail pour correction.

Conclusion

Nous avons vu au cours de ce fichier d'introduction à la programmation en Scilab comment approcher les valeurs des racines carrées d'un nombre entier. Cela correspond en fait à un cas particulier du problème général en mathématiques qui est la résolution d'équation.

Depuis la classe de quatrième on sait résoudre des équations du premier degré (équation où il n'y a que des x, mais pas de x^2), en classe de troisième, seconde, mais surtout première (certaines classes de premières...) on apprend à résoudre les équations du second degré (avec des x et des x^2, mais pas de x^3 ou plus). En fait il existe des méthodes de résolution des équations jusqu'au degré 4, mais à partir du degré 5, on n'est assuré de rien. Cela a été montré à l'aide des travaux d'Evariste Galois (dont on a parlé en classe...).

D'où l'intérêt de l'informatique pour approcher les solutions de certaines équations, équations dont les solutions sont souvent (très souvent) importantes : dans l'industrie, l'ingénieurie, l'architecture, etc.

Annexe A

Muhammad AL-KHWÂRÎZMÎ

Ce qui suit est extrait du site internet :

http ://mathematiques.ac-bordeaux.fr/viemaths/hist/mthacc/alkhwarizmi.htm

Nous devons beaucoup à al-Khwarizmi (780 env. - 850 env.), qui n'a pas sans doute en Europe la célébrité qu'il mérite. Nous lui devons rien moins que notre système décimal de numération, et deux mots fondamentaux dans le vocabulaire des mathématiques, celui d'Algorithme et celui d'Algèbre. Il accomplit dans ce dernier domaine un progrès notable, par son traitement systématique des équations de degré 2. Il représente ici la nombreuse et valeureuse cohorte des mathématiciens arabes, parmi lesquels on peut citer Thâbit ibn Qurra (mort en 901), Ibn al-Haytham, dit Alhazen (965-1039), al Karaji,qui vécut vers l'an 1000, Omar al-Khayyam (1048-1122), al-Tüsi (vers 1170), al-Samaw'al (mort en 1174), et bien d'autres... Il se trouve qu'al-Khwarizmi est chronologiquement le premier d'entre eux. Il fut membre de la "Maison de la Sagesse", à Bagdad (une sorte de centre de recherche, on peut penser au "Musée" d'Alexandrie, dont firent partie Euclide et Eratosthène). Fondateur donc des mathématiques arabes, il fut aussi l'introducteur dans son aire culturelle d'une grande partie des connaissances de l'Inde en mathématiques. Il écrivit un traité, que l'on n'a pas retrouvé (mais que l'on connaît par l'intermédiaire de ses traductions en latin), dans lequel il exposait le système décimal indien, les méthodes de calcul dans ce système (addition, soustraction, multiplication...), mais aussi les fractions, certains calculs en système sexagésimal, les racines carrées... S'il donna aux Arabes les connaissances indiennes, al-Khwarizmi les offrit aussi à l'Occident, puisque c'est grâce à la traduction de ses livres en latin que les Européens purent connaître et adopter le système décimal indien, largement perfectionné entre temps par les Arabes. Ces traductions n'apparaissent en Europe qu'au XIIème siècle, sous les titres significatifs de Dixit Algorizmi ("Al-Khwarizmi a dit que..."), ou (pour le même livre) "De numero Indorum (Le Nombre des Indiens), ou Liber Alchorismi (le livre d'Al-Khwarizmi). Ce nouvel et magnifique système de calcul fut donc désigné par les Européens du nom d'algorisme, en hommage à l'auteur de ces ouvrages. On peut donc dire qu'al-Khwarizmi fut le messager du système décimal indien, qui est désormais un élément important de la culture universelle, non seulement dans sa propre civilisation, mais aussi vers l'Europe, par l'intermédiaire des traductions en latin de ses ouvrages. Al-Khwarizmi a un autre titre de gloire, c'est d'avoir écrit un traité sur les équations du

second degré. Celles-ci étaient connues depuis des millénaires, par les Babyloniens par exemple, et des méthodes générales de résolution avaient été données pour certains types d'entre elles par Euclide. Mais c'est al-Khwarismi qui élabore le premier une classification générale, grâce à une vision globale du problème. Il n'utilise aucune notation littérale, et toutes ses résolutions se font en langage "courant". L'inconnue est désignée comme "la racine" ou "la chose". Les résolutions se font, non pas au moyen de calculs, mais à l'aide de constructions géométriques, dans le style euclidien. Le titre de son ouvrage est Kitâb al-jabr wa al-muqâbala, ce que l'on peut traduire par "le livre du rajout et de l'équilibre". L'ouvrage fut traduit en latin au XIIème siècle, sous le titre d'Algèbre (Algebra en latin), car le traducteur conserva le mot arabe (le même phénomène se produit de nos jours pour des mots anglais comme "bug", ou "plug-in", que les traducteurs n'ont pas l'énergie de traduire en français). Voici donc l'origine d'un des mots les plus importants des mathématiques. Mais outre le mot, Al-Khwarizmi introduisit dans les pays arabes comme en Occident l'intérêt pour la disciplina elle-même, éveillé également par les ouvrages d'Euclide ou de Diophante. Le successeurs arabes firent faire des progrès remarquables à l'algèbre. Il ne restait plus à la Renaissance qu'à résoudre de façon générale l'équation du troisième degré (voir Cardan). . . avant celle de degré 5 (résolution impossible, comme l'a démontré Évariste Galois).

al-jabr et al-muqabula sont des "transpositions" :
al-jabr correspond à transformer une soustraction dans un membre en une addition dans l'autre membre.
Par exemple : $2x^2 + 100 - 20x = 58$
devient par al-jabr : $2x^2 + 100 = 58 + 20x$.
al-muqabala (le balancement) revient à supprimer dans les deux membres l'addition d'un même nombre.
Par exemple : $2x^2 + 100 = 58 + 20x$
devient par al-muqabala : $2x^2 + 42 = 20x$.