ALGORITHMIQUE
&
CALCUL NUMÉRIQUE

Travaux pratiques résolus

Programmation avec les logiciels Scilab et Python

Licence
Préparation aux concours

José OUIN
Ingénieur INSA Toulouse
Professeur agrégé de Génie civil
Professeur agrégé de Mathématiques

ellipses

2- Les instructions

Les instructions constituent la structure des algorithmes et leur assemblage, dans un ordre précis, conduit au résultat attendu. Elles sont écrites en pseudo-code. Des exemples concrets d'écriture en langage Scilab et en langage Python sont également proposés.

2-1. Les instructions pour traiter les données

Il s'agit d'instructions de base comme la lecture de données, l'affectation dans des variables et l'écriture de données.

2-1.1 L'affectation de données dans des variables

L'affectation permet d'attribuer une valeur à une variable désignée par son identificateur. Un identificateur est une suite de lettres et chiffres (sans espaces) qui doit être choisi judicieusement pour que l'algorithme soit immédiatement lisible et interprétable.

- **Affectation en pseudo-code**

```
identificateur prend la valeur 5
```

Exemple :

```
d prend la valeur 5
```

L'affectation remplace la valeur précédente de la variable par la nouvelle. Ainsi l'instruction "`d prend la valeur 5`" affecte la valeur 5 à la variable dont d est l'identificateur et ceci quelle que soit la valeur contenue au préalable dans la variable d (laquelle sera perdue).

- **Affectation en langage Scilab**

```
d = 5;
```

Le point-virgule ";" est optionnel. Il permet de ne pas afficher la valeur de la variable d dans la console au cours du déroulement du programme.

- **Affectation en langage Python**

```
d = 5
```

2-1.2 La lecture (ou entrée) des données

La lecture de données peut se faire par interrogation de l'utilisateur ou par extraction à partir d'un fichier.

- **Lecture des données en pseudo-code**

Saisir identificateur.

Exemple :

```
Saisir p
```

- **Lecture des données en langage Scilab**

On ne considère que le cas de la lecture de données par interrogation de l'utilisateur. La procédure de lecture de données dans un fichier dépend en effet de la disposition des données dans ce fichier et ne peut donc pas être décrite de manière générale.

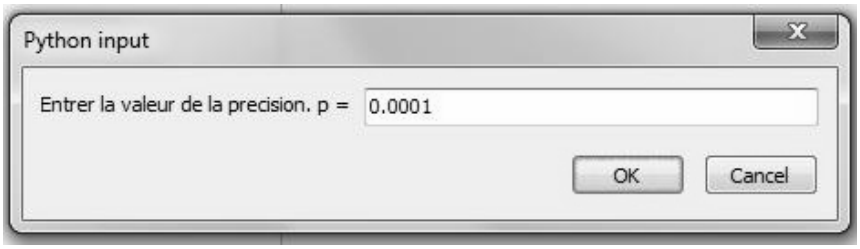
```
w = input("Entrer votre prénom : ","string");
u = input("Entrer les bornes de l''intervalle. [a,b]= ");
p = input("Entrer la valeur de la précision. p= ");
```

La variable w est une chaîne de caractère ("string") contenant le prénom saisi.
La variable u est un vecteur contenant deux nombres réels.
Si l'utilisateur a saisi [3,9], alors u(1) (ou u(1,1)) est égal à 3 et u(2) (ou u(1,2)) est égal à 9.
La variable p contient un nombre réel.

- **Lecture des données en langage Python**

On ne considère que le cas de la lecture de données par interrogation de l'utilisateur.

```
w = input ("Entrer la valeur de la précision. p = ")
```



2-1.3 L'écriture (ou sortie) des données

L'écriture des données permet d'afficher, pour l'utilisateur, les valeurs des variables après traitement.

- **Ecriture des données en pseudo-code**

```
Afficher identificateur
```

Pour écrire des informations non contenues dans une variable :

```
Afficher "message"
```

Exemple :

```
Afficher x1
Afficher "L'équation n'a aucune solution"
```

- **Ecriture des données en langage Scilab**

➔ **La fonction disp**

Affichage d'un message :

```
disp ("f doit changer de signe !")
```

- **Structure alternative en langage Python**

```
d = b**2-4*a*c

if d > 0 :
    x1 = (-b - np.sqrt(d))/(2*a)
    x2 = (-b + np.sqrt(d))/(2*a)
    print ("RESULTATS :")
    print ("Deux racines distinctes : x1 = " + str(x1) + " et x2 = " +
str(x2))

elif d == 0 :
    x1 = -b/(2*a)
    print ("RESULTATS :")
    print ("Une racine double : x1 = " + str(x1))
else :
    print ("RESULTATS :")
    print ("Aucune solution ! ")
```

2-2.2 Les structures répétitives

Les structures répétitives permettent d'exécuter plusieurs fois de suite le même traitement c'est à dire la même série d'instructions.
On utilise pour cela un compteur (par exemple une variable k) ou une condition pour contrôler le nombre de fois que les instructions sont répétées.
Dans le cas d'un compteur, pour chaque répétition, la structure répétitive incrémente ou décrémente la valeur de la variable k d'une valeur prédéfinie.
Dans le cas d'une structure répétitive avec une condition, la structure vérifie si la condition est toujours vérifiée avant chaque répétition.

- **Structures répétitives en pseudo-code**

Structure répétitive avec un compteur (structure répétitive de 1 à n)

```
Pour k de 1 jusqu'à n Faire
      {Traitement 1}
FinPour
```

Structure répétitive avec une condition

```
TantQue (b – a) > p Faire
      {Traitement 2}
FinTantQue
```

Remarque :
Le nombre de répétitions dépendra de la condition :
- Si la condition n'est pas vérifiée au début alors le "Traitement 2" ne sera pas exécuté du tout.
- Si la condition est vérifiée au début et si la condition n'est pas susceptible d'être modifiée lors du "Traitement 2", alors le "Traitement 2" sera exécuté indéfiniment et l'utilisateur sera obligé d'arrêter lui-même le programme. On dit que le programme boucle indéfiniment, ce qui est une erreur majeure de programmation. Pour que l'algorithme soit correct, il est nécessaire que la condition cesse d'être vérifiée au bout d'un nombre fini de répétitions.

➔ **Exemple 1 :** Structure répétitive de 1 à n

```
t = 0

Pour k de 1 jusqu'à n Faire
	x prend une valeur aléatoire strictement comprise entre 0 et 1.
	u prend une valeur aléatoire strictement comprise entre 0 et 1.
	y prend la valeur (1 - x) * u
	z prend la valeur 1 - x - y

	Si x < 0.5 et y < 0.5 et z < 0.5 Alors
		t prend la valeur t + 1
	FinSi
FinPour
```

➔ **Exemple 2 :** Structure répétitive avec une condition

```
TantQue (b – a) > p Faire
	m prend la valeur (b + a)/2
	Si f(a)*f(m) < 0 Alors
		b prend la valeur m
	Sinon
		a prend la valeur m
	FinSi
FinTantQue
```

- **Structures répétitives en langage Scilab**

➔ **Exemple 1 :** Structure répétitive de 1 à n

```
for k = 1:n
	x = rand();
	y = (1 - x) * rand();
	z = 1 - x - y;

	if x < 0.5 & y < 0.5 & z < 0.5 then
		t = t + 1;
	end
end
```

➔ **Exemple 2 :** Structure répétitive avec une condition

```
while (b - a) > p
	m = (a + b) / 2;
	if f(a)*f(m) < 0 then
		b = m;
	else
		a = m;
	end
end
```

- **Structures répétitives en langage Python**

➔ **Exemple 1 :** Structure répétitive de 1 à n

```
for k in range(1,n + 1) :
      x = random.random()
      y = (1 - x) * random.random()
      z = 1 - x - y
      if x < 0.5 and y < 0.5 and z < 0.5 :
            t = t + 1
```

➔ **Exemple 2 :** Structure répétitive avec une condition

```
while (b - a) > p :
      m = (a + b) / 2
      if f(a)*f(m) < 0 :
            b = m
      else :
            a = m
```

2-2.3 Indentation nécessaire en langage Python

Pour les structures de contrôles du langage Python, il n'y a pas de mot clé "end" pour signaler la fin du bloc de lignes concernées par les structures if, while, for.
Pour toute boucle, test, fonction, il faut indenter les lignes (c'est-à-dire créer des décalages à l'aide de la touche "tabulation" du clavier) afin de définir une dépendance d'un bloc de lignes par rapport à un autre.

Exemple :
➔ Programme N° 1

```
k = 0
p = 0

for i in range(1,5) :
      k = k + 1
p = p + 1

print("k = ", k)
print("p = " ,p)
```

La console affiche :

```
k = 4
p = 1
```

➔ Programme N° 2

```
k = 0
p = 0

for i in range(1,5) :
      k = k + 1
      p = p + 1

print("k = ", k)
print("p = ", p)
```

La console affiche :

```
k = 4
p = 4
```

Les travaux pratiques

Tri par sélection et tri à bulles

2-1 Enoncé

Il s'agit de développer un algorithme, puis un programme, permettant de trier une liste de valeurs numériques donnée afin de ranger les éléments dans l'ordre croissant. On considère deux méthodes de tri : le tri par sélection (ou tri par extraction) et le tri à bulles (ou tri par propagation).

2-2 Travail demandé

1] Ecrire un algorithme de tri par sélection (ou tri par extraction).
2] Ecrire le programme correspondant.
3] Trier la liste suivante : [9,9,9,10.5,12,15,242,1,0,5,1,2,78].
4] Ecrire un algorithme de tri à bulles (ou tri par propagation).
5] Ecrire le programme correspondant.
6] Trier la liste suivante : [9,9,9,10.5,12,15,242,1,0,5,1,2,78].

2-3 Méthodes et fonctions utilisées

2-3.1 Présentation des méthodes

Tri par sélection

On considère une liste de n éléments $a(1),a(2),a(3),...a(n)$ donnée. Le principe du tri par sélection est le suivant :

➔ On cherche le plus petit élément de la liste de n éléments $a(1),a(2),a(3),...a(n)$ puis on l'échange avec l'élément d'indice 1 : $a(1)$.

➔ On recherche ensuite le second plus petit élément de la liste de $n-1$ éléments $a(2),a(3),...a(n)$ puis on l'échange avec l'élément d'indice 2 : $a(2)$.

➔ On recherche ensuite le troisième plus petit élément de la liste de $n-2$ éléments $a(3),a(4),...a(n)$ puis on l'échange avec l'élément d'indice 3 : $a(3)$.

➔ On continue de cette façon jusqu'à ce que la liste soit entièrement triée.

Tri à bulles
Le tri à bulles est un algorithme de tri qui consiste à faire remonter progressivement les plus grands éléments d'un tableau, un peu comme des bulles d'air qui remonteraient progressivement à la surface d'un liquide.
On considère une liste de n éléments $a(1), a(2), a3), ... a(n)$ donnée. Le principe du tri à bulles est le suivant :
➔ On parcourt la liste en comparant les couples d'éléments successifs $a(i)$ et $a(i+1)$. Lorsque deux éléments successifs ne sont pas dans l'ordre croissant, ils sont échangés.
➔ Après chaque parcours complet de la liste, on recommence l'opération.
➔ On continue jusqu'à ce qu'aucun échange n'ait eu lieu pendant un parcours.

2-3.2 Les fonctions à utiliser

Fonctions Scilab
length() ; disp() ; input()

Variable booléenne :
echange = %t donne la valeur "VRAI" à la variable echange.
echange = %f donne la valeur "FAUX" à la variable echange.

Fonctions Python
len() ; input() ; print() ; range() ; eval()

Variable booléenne :
echange = True donne la valeur "VRAI" à la variable echange.
echange = False donne la valeur "FAUX" à la variable echange.

Solution

2-4 L'algorithme

➔ Tri par sélection

Entrées

```
Saisir la liste M()
```

Traitement

```
N prend la valeur "nombre d'éléments de M()"

Pour i de 1 jusqu'à (N - 1) Faire
    Pour j de (i + 1) jusqu'à N Faire
        Si M(j) <  M(i) Alors
           val_temp prend la valeur M(i)
           M(i) prend la valeur M(j)
           M(j) prend la valeur val_temp
        FinSi
    FinPour
FinPour
```

Sorties

```
Afficher M()
```

➔ Tri à bulles

Entrées

```
Saisir la liste M()
```

Traitement

```
N prend la valeur "nombre d'éléments de M()"
echange prend la valeur VRAI

TantQue echange ==  VRAI
    echange prend la valeur FAUX
    Pour i de 1 jusqu'à (N - 1) Faire
        Si M(i) >  M(i + 1) Alors
           val_temp prend la valeur M(i)
           M(i) prend la valeur M(i + 1)
           M(i + 1) prend la valeur val_temp
           echange prend la valeur VRAI
        FinSi
    FinPour
FinTantQue
```

Sorties

```
Afficher M()
```

2-5 Le programme Scilab

➔ Tri par sélection

```
// Tri par sélection

// Saisie des valeurs à trier
M = input("Entrer la liste des n valeurs à trier : [a1,a2,.....,an]=
");
N = length(M)
// Programme

for i = 1:(N-1)
    for j = (i+1):N
        if M(j) < M(i) then
           val_temp = M(i)
           M(i) = M(j)
           M(j) = val_temp
        end
    end
end

disp("Liste triée :")
disp(M)
```

➔ Tri à bulles

```
// Tri à bulles

// Saisie des valeurs à trier
M = input("Entrer la liste des n valeurs à trier : [a1,a2,.....,an]=
");
N = length(M)
// Programme

echange = %t;

while echange == %t
    echange = %f
    for i = 1:(N-1)
        if M(i) > M(i+1) then
           val_temp = M(i)
           M(i) = M(i+1)
           M(i+1) = val_temp
           echange = %t
        end
    end
end

disp("Liste triée :")
disp(M)
```

2-6 Résultats numériques Scilab

Entrer la liste des n valeurs à trier : [a1,a2,.....,an]=
[9,9,9,10.5,12,15,242,1,0,5,1,2,78]

Liste triée :
0. 1. 1. 2. 5. 9. 9. 9. 10.5 12. 15. 78. 242.

2-7 Le programme Python

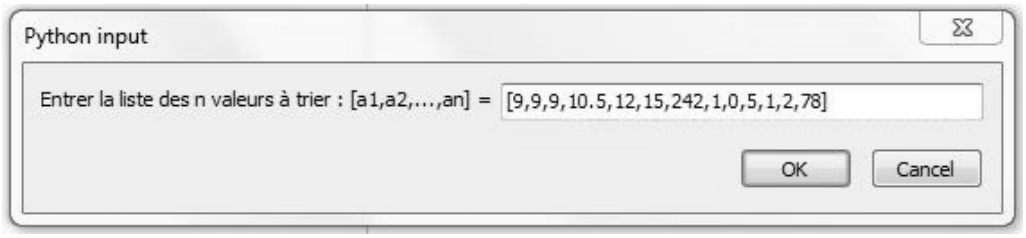
➔ Tri par sélection

```
# Tri par sélection
# Données initiales
Liste = input("Entrer la liste des n valeurs à trier : [a1,a2,...,an]
=")
M = eval(Liste)
N = len(M)

# Programme
for i in range (0,N-1):
    for j in range(i + 1,N):
        if M[j] < M[i]:
           val_temp = M[i]
           M[i] = M[j]
           M[j] = val_temp

print("Liste triée :")
print(M)
```

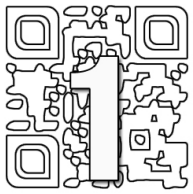
➔ Tri à bulles

```
# Tri à bulles
# Données initiales
Liste = input("Entrer la liste des n valeurs à trier : [a1,a2,...,an]
=")
M = eval(Liste)
N = len(M)

# Programme
echange = True

while echange == True:
    echange = False
    for i in range (0,N - 1):
        if M[i] > M[i+1]:
            val_temp = M[i]
            M[i] = M[i+1]
            M[i+1] = val_temp
            echange = True

print("Liste triée :")
print(M)
```

2-8 Résultats numériques Python




```
>>>
Liste triée :
[0, 1, 1, 2, 5, 9, 9, 9, 10.5, 12, 15, 78, 242]
```



Ensembles de Mandelbrot et de Julia

1-1 Enoncé

➔ **Ensemble de Mandelbrot**

L'ensemble de Mandelbrot est une fractale définie comme l'ensemble des points du plan complexe d'affixe c pour lesquels la suite (z_n) est convergente. Cette suite est définie par la formule de récurrence suivante :

$$\begin{cases} z_0 = 0 \\ z_{n+1} = z_n^2 + c \end{cases}$$; pour $n \geq 0$, $z_0 = 0$ et c variable.

Une fractale est une figure géométrique constituée d'un motif qui se répète à l'infini (si on effectue un zoom sur une partie de la figure, on retrouve le même motif).

➔ **Ensemble de Julia**

Si on fixe la valeur de c, l'ensemble de Julia est l'ensemble des points du plan complexe d'affixe z_0 pour lesquels la suite (z_n) est convergente. Cette suite est définie par la formule de récurrence suivante :

$$\begin{cases} z_0 = a'+ib' \\ z_{n+1} = z_n^2 + c \end{cases}$$; pour $n \geq 0$, $z_0 = a'+ib'$ et c fixé.

Il s'agit de développer un algorithme, puis un programme, permettant de représenter ces deux ensembles.

1-2 Travail demandé

1] Ecrire les algorithmes permettant de représenter ces deux ensembles.
2] Ecrire les programmes correspondants.
3] Représenter l'ensemble de Mandelbrot pour $c = a + ib$ avec $a \in [-2.1;0.6]$ et $b \in [-1.2;1.2]$.
4] Représenter l'ensemble de Julia pour $c = 0.285 + 0.010i$ et $z_0 = a'+ib'$ avec $a' \in [-1;1]$ et $b' \in [-1.2;1.2]$.

1-3 Méthode et fonctions utilisées

1-3.1 Présentation de la méthode

(1) Convergence de la suite (z_n) :

On admet que s'il existe n tel que $|z_n| > 2$ alors la suite (z_n) est divergente.

(2) Itérations successives :

On considère $taille\_x$ valeurs pour a et $taille\_y$ valeurs pour b dans leurs intervalles respectifs (on fait de même pour les valeurs a' et b'). Pour chaque couple de valeurs, on teste si la suite (z_n) est convergente ou non. On effectue pour cela k itérations.

(3) Langage Scilab : On construit une matrice M de taille $taille\_y \times taille\_x$ (c'est-à-dire une matrice comportant $taille\_y$ lignes et $taille\_x$ colonnes). On note $m_{i,j}$ le terme de la matrice M situé à l'intersection de la ligne i et de la colonne j.

On attribue alors les valeurs suivantes :

➔ $m_{N+1-j,i} = 8$ si la suite (z_n) est divergente pour le complexe $c = a_i + ib_j$ (ensemble de Mandelbrot) ou pour $z_0 = a'_i + ib'_j$ (ensemble de Julia).

➔ $m_{N+1-j,i} = 1$ si la suite (z_n) est convergente pour le complexe $c = a_i + ib_j$ (ensemble de Mandelbrot) ou pour $z_0 = a'_i + ib'_j$ (ensemble de Julia).

(4) Langage Scilab : On utilise la fonction `Matplot(M)` qui attribue la couleur noire aux valeurs 1 et la couleur blanche aux valeurs 8.

Langage Python : On utilise la fonction `create_line(i, j, i, j + 1)` pour tracer les segments de droites $[M_{i,j}N_{i,j+1}]$ avec $M_{i,j}(i;j)$ et $N_{i,j+1}(i;j+1)$.

1-3.2 Les fonctions à créer

Ecriture d'une valeur complexe avec Scilab :

Le nombre complexe $c = 3 + 2i$ s'écrit : `c = 3 + %i*2`

Ecriture d'une valeur complexe avec Python :

Le nombre complexe $c = 3 + 2i$ s'écrit : `c = complex(3,2)`

1-3.3 Les fonctions à utiliser

Fonctions Scilab

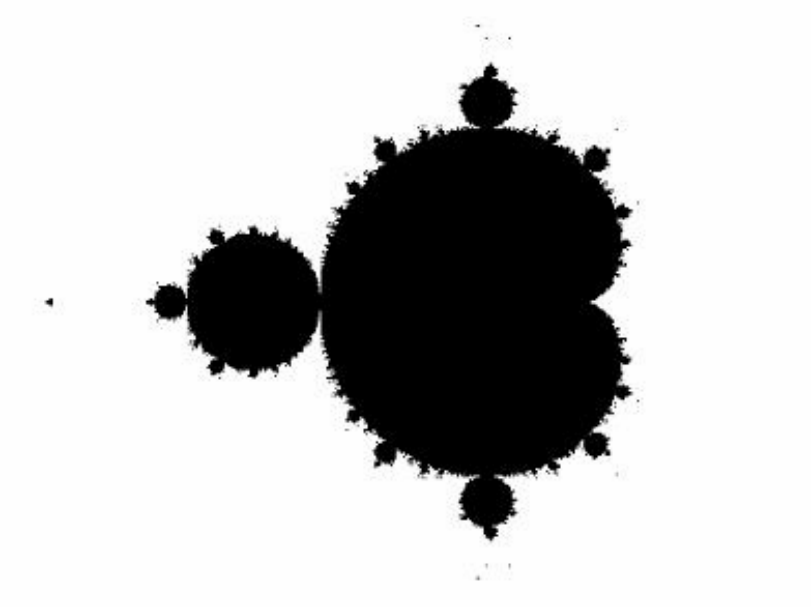
Matplot() ; linspace() ; zeros() ; abs() ; int()

Fonctions Python

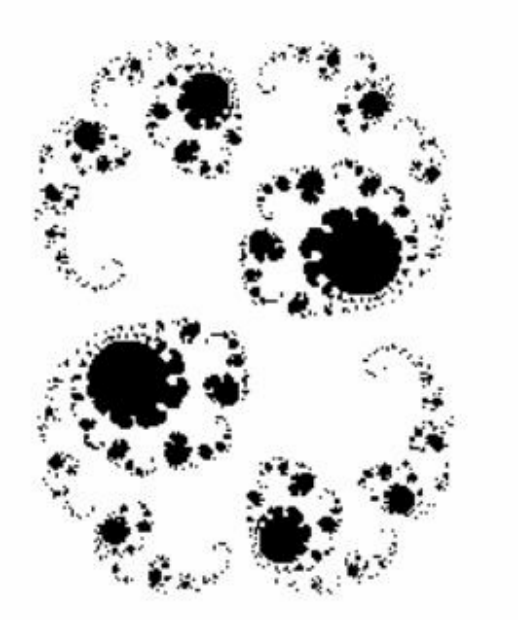
range() ; complex() ; abs() ; create_line() ; int()

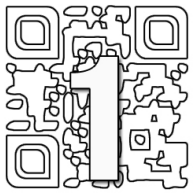
1-6 Résultats numériques Scilab

Ensemble de Mandelbrot de la question 3]



Ensemble de Julia de la question 4]





Approximation de la fonction exponentielle par la méthode d'Euler

1-1 Enoncé

Il s'agit de développer un algorithme, puis un programme, de résolution numérique de l'équation différentielle suivante :

$$\begin{cases} u'(t) = u(t) \\ u(0) = 1 \end{cases} \quad ; \; t \in [0;5]$$

1-2 Travail demandé

1] Ecrire l'algorithme permettant de résoudre numériquement l'équation différentielle proposée.
2] Ecrire le programme correspondant et tracer le nuage de points de l'approximation de la solution.
3] Représenter sur le même graphique la courbe représentative de la fonction s suivante :

$$s : x \mapsto e^x$$

1-3 Méthode et fonctions utilisées

1-3.1 Présentation de la méthode

On considère l'équation différentielle de forme générale suivante :

$$\begin{cases} u'(t) = f\big(t, u(t)\big) \\ u(a) = u_0 \end{cases} \quad ; \; t \in [a;b]$$

(1) On découpe l'intervalle $[a;b]$ en N parties et on pose : $h = \dfrac{b-a}{N}$.

On obtient ainsi $N+1$ points $t_i = a + i\dfrac{b-a}{N} = a + ih$ avec $i \in [0;N]$.

(2) Le nombre dérivé $u'(t_i)$ en t_i est approché par le quotient suivant :

$$u'(t_i) \approx \frac{u(t_{i+1}) - u(t_i)}{t_{i+1} - t_i}$$

Si on note y_i l'approximation de la valeur $u(t_i)$ en t_i, on obtient :

$$u'(t_i) \approx \frac{y_{i+1} - y_i}{t_{i+1} - t_i} \text{ et } u'(t_i) = f\left(t_i, u(t_i)\right)$$

Finalement l'approximation s'écrit :

$$\frac{y_{i+1} - y_i}{t_{i+1} - t_i} = f\left(t_i, y_i\right)$$

(3) On en déduit la relation de récurrence suivante :

$$y_{i+1} - y_i = \left(t_{i+1} - t_i\right) \times f\left(t_i, y_i\right)$$
$$y_{i+1} = y_i + \left(t_{i+1} - t_i\right) \times f\left(t_i, y_i\right)$$

$$\begin{cases} y_{i+1} = y_i + h \times f\left(t_i, y_i\right) \\ y_0 = u_0 \end{cases} \quad ; \quad i \in [0; N]$$

1-3.2 Les fonctions à créer

On crée la fonction f(t,y) suivante :

➔ Function z = f(t,y)
Description : Cette fonction permet de calculer la valeur $f\left(t_i, y_i\right)$.

1-3.3 Les fonctions à utiliser

Fonctions Scilab
plot() ; zeros()

Fonctions Python
zeros() ; range() ; plot() ; show()

Solution

1-4 L'algorithme

Entrées

```
Saisir a, b, u0, N
```

Traitement

```
Définir la fonction f

h prend la valeur (b-a)/N

t(1,1) prend la valeur a
y(1,1) prend la valeur u0

Pour i de 1 jusqu'à N Faire
    t(1,i+1) prend la valeur t(1,1) + h*i
    y(1,i+1) prend la valeur y(1,i) + h*f(t(1,i),y(1,i))
FinPour
```

Sorties

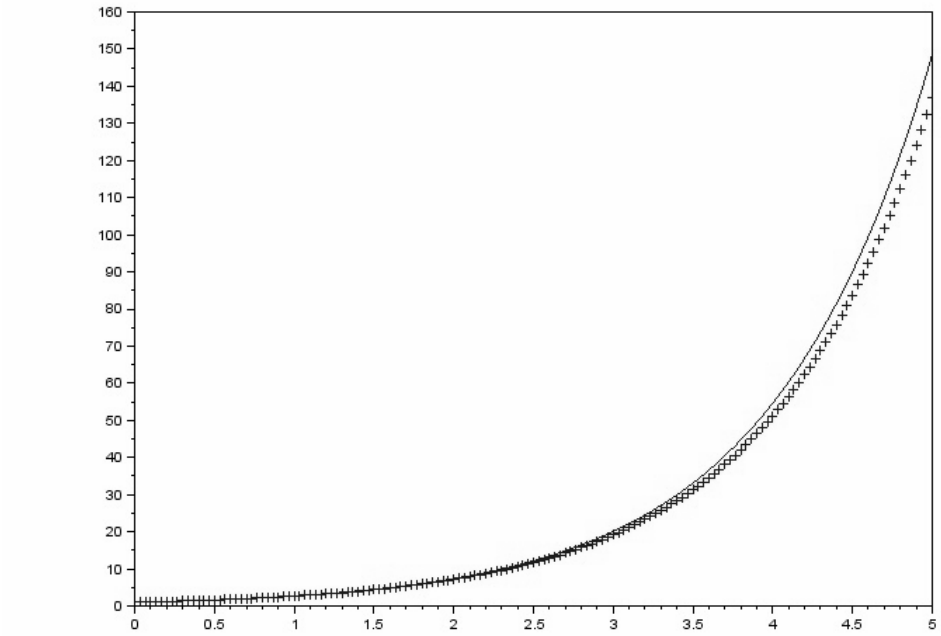
```
Tracer le nuage de points (t(1,i) ; y(1,i))
Tracer la courbe représentative de la fonction exponentielle
```

1-5 Le programme Scilab

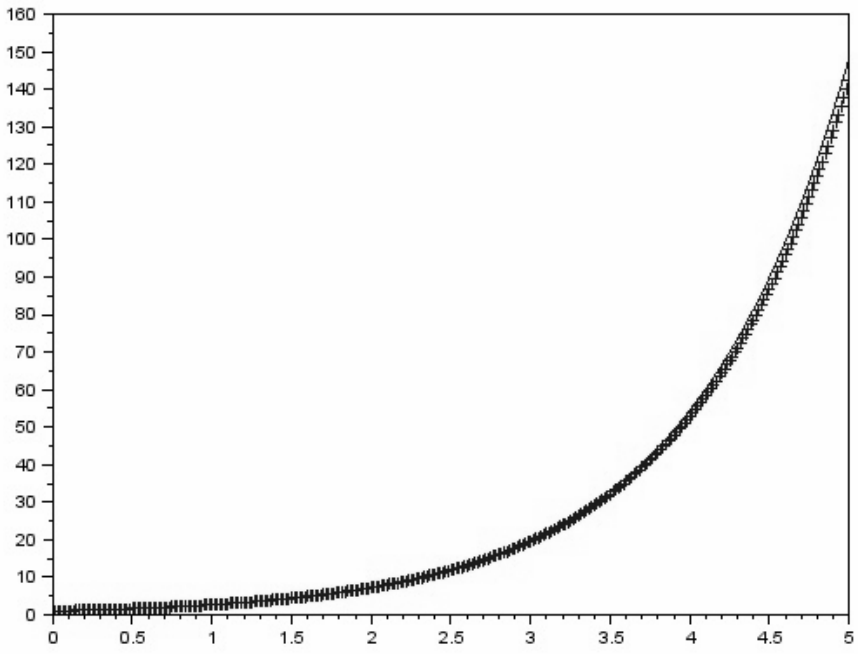
```
// Approximation de la fonction exponentielle
//----
funcprot(0);

// Fonction f
function z = f(t, y)
    z = y;
endfunction

// Données initiales
a = 0;
b = 5;
u0 = 1;
N = 150;
h = (b-a)/N;

// Programme
t = zeros(1,N+1);
y = zeros(1,N+1);

t(1,1) = a;
y(1,1) = u0;

for i = 1:N
   t(1,i+1) = t(1,1) + h*i;
   y(1,i+1) = y(1,i) + h*f(t(1,i),y(1,i)) ;
end

// Représentations graphiques
plot(t,y,"+b");
plot(t,exp(t))
```

1-6 Résultats numériques Scilab

N = 150



N = 300

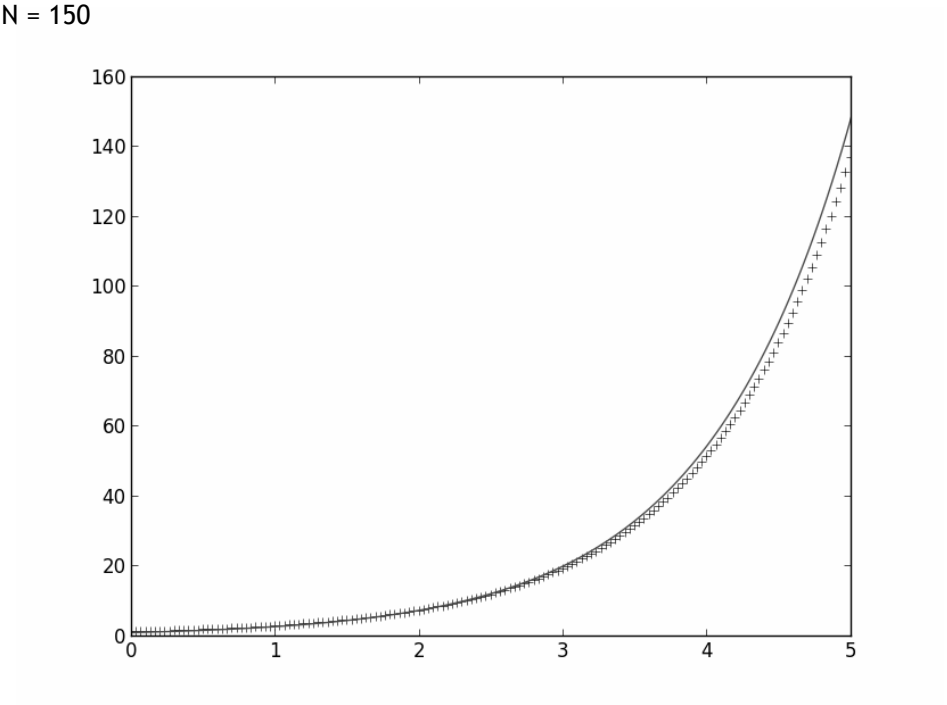


1-7 Le programme Python

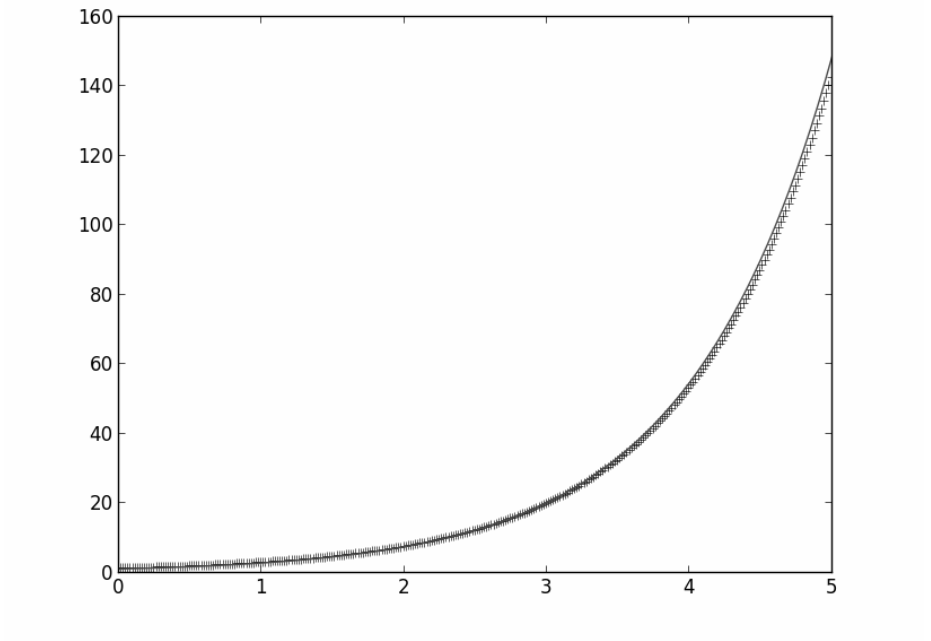
```
# Approximation de la fonction exponentielle

import matplotlib.pyplot as plt
import numpy as np

def f(t,y):
    return y

# Données initiales

a = float(0)
b = float(5)
u0 = 1
N = 300
h = (b-a)/N

# Programme

t = np.zeros(N+1)
y = np.zeros(N+1)
sol = np.zeros(N+1)

t[0] = a
y[0] = u0

for i in range(0,N) :

    t[i+1] = t[0] + h*(i+1)
    y[i+1] = y[i] + h*f(t[i],y[i])


# Représentations graphiques

plt.plot(t,y,"+b",t,np.exp(t),"r")
plt.show()
```

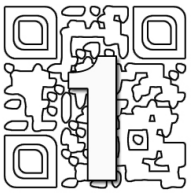
1-8 Résultats numériques Python

N = 150



N = 300





Méthode d'élimination de Gauss

1-1 Enoncé

Il s'agit de développer un algorithme, puis un programme, permettant de transformer un système matriciel $A.X = B$ en un système équivalent $\widetilde{A}.X = \widetilde{B}$ où $\widetilde{A}$ est une matrice triangulaire supérieure déterminée à partir de la matrice A. Les formules de récurrences nécessaires sont données dans la partie relative à la méthode utilisée.

On utilise les notations suivantes :

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1j} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2j} & \dots & a_{2n} \\ \dots & \dots & & & & \\ a_{i1} & a_{i2} & \dots & a_{ij} & \dots & a_{in} \\ \dots & \dots & & & & \\ a_{n1} & a_{n2} & \dots & a_{nj} & \dots & a_{nn} \end{pmatrix} ; \quad X = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_i \\ \dots \\ x_n \end{pmatrix} ; \quad B = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_i \\ \dots \\ b_n \end{pmatrix}$$

1-2 Travail demandé

1] Ecrire l'algorithme permettant de déterminer les matrices $\widetilde{A}$ et $\widetilde{B}$.
2] Ecrire le programme correspondant.
3] Application numérique :

$$A = \begin{pmatrix} 2 & -5 & 1 & 3 \\ 4 & 7 & 8 & 2 \\ 3 & 1 & 1 & 6 \\ 4 & 1 & 7 & 9 \end{pmatrix} \quad ; \quad X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} ; \quad B = \begin{pmatrix} 5 \\ 10 \\ 2 \\ 6 \end{pmatrix}$$

1-3 Méthode et fonctions utilisées

1-3.1 Présentation de la méthode

L'algorithme d'élimination de Gauss comporte n étapes. On note $a_{i,j}^{(k)}$ l'élément de la $i^{ème}$ ligne et de la $j^{ème}$ colonne de la matrice A à l'étape k, notée $A^{(k)}$. On obtient la matrice $A^{(n)} = \widetilde{A}$ au bout des n étapes.

On initialise l'algorithme avec $A^{(1)} = A$ puis on calcule les étapes $k = 1, k = 2, k = 3, ..., k = n-1$ en utilisant les formules de récurrences suivantes définies pour $i \in \{k+1, k+2, ..., n\}$:

$$\begin{cases} a_{i,j}^{(k+1)} = a_{i,j}^{(k)} - \dfrac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}} \times a_{k,j}^{(k)} \;\; ; \;\; j \in \{k+1, k+2, ..., n\} \\ b_i^{(k+1)} = b_i^{(k)} - \dfrac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}} \times b_k^{(k)} \end{cases}$$

1-3.2 Les fonctions à utiliser

Fonctions Scilab

zeros() ; evstr() ; disp()

Fonctions Python

range() ; print()

Solution

1-4 L'algorithme

Entrées
```
Saisir les matrices A et B
```

Traitement
```
AT prend la valeur A
BT prend la valeur B

Pour k de 1 jusqu'à (N-1) Faire

       Pour i de (k+1) jusqu'à N Faire

              q prend la valeur AT(i,k)/AT(k,k)
              BT(i,1) prend la valeur BT(i,1) - q*BT(k,1)
              AT(i,k) prend la valeur 0

              Pour j de (k + 1) jusqu'à N Faire
                     AT(i,j) prend la valeur AT(i,j) - q*AT(k,j)
              FinPour

       FinPour

FinPour
```

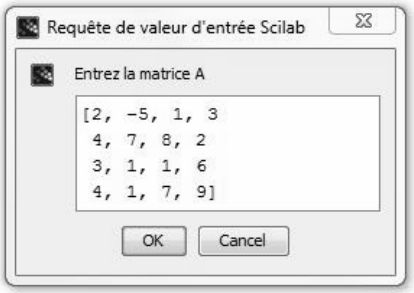
Sorties
```
Afficher les matrices AT et BT
```

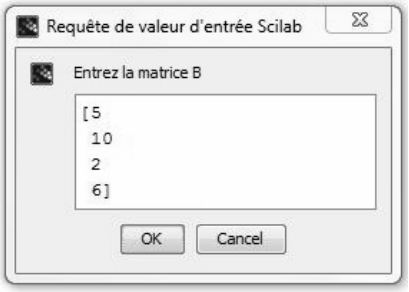
1-5 Le programme Scilab

```
// Méthode d'élimination de Gauss

// Données
N = 4;

AT = zeros(N,N);
BT = zeros(N,1);
matA = zeros(N,N);
matB = zeros(N,1);
A = evstr(x_matrix('Entrez la matrice A',matA));
B = evstr(x_matrix('Entrez la matrice B',matB));
AT = A;
BT = B;

// Programme
for k = 1:(N-1)
    for i = (k+1):N
        q = AT(i,k)/AT(k,k)
        BT(i,1) = BT(i,1) - q*BT(k,1)
        AT(i,k) = 0
        for j = (k + 1):N
             AT(i,j) = AT(i,j) - q*AT(k,j)
        end
    end
end

disp("Matrice AT = ")
disp(AT)
disp("Matrice BT = ")
disp(BT)
```

1-6 Résultats numériques Scilab

```
 Matrice AT =

    2.  - 5.     1.     3.
    0.    17.    6.   - 4.
    0.    0.   - 3.5    3.5
    0.    0.     0.     6.7058824

 Matrice BT =

    5.
    0.
  - 5.5
  - 5.7563025
```





1-7 Le programme Python

```
# Méthode d'élimination de Gauss

import numpy as np

N = 4
A = np.array([  [2,  -5,  1,  3],
                [4,  7,  8,  2],
                [3,  1,  1,  6],
                [4,  1,  7,  9]], float)
B = np.array([[ 5],
               [10],
                [2],
                [6]],float)

AT = A.copy()
BT = B.copy()

for k in range(0,(N-1)):
    for i in range((k + 1),(N)):
        q = AT[i,k]/AT[k,k]
        BT[i,0] = BT[i,0] - q*BT[k,0]
        AT[i,k] = 0
        for j in range((k+1), (N)):
            AT[i,j] = AT[i,j] - q*AT[k,j]

print("Matrice AT : ")
print(AT)
print("Matrice BT : ")
print(BT)
```

1-8 Résultats numériques Python

```
 >>>

Matrice AT :

[[  2.          -5.           1.           3.        ]
 [  0.          17.           6.          -4.        ]
 [  0.           0.          -3.5          3.5       ]
 [  0.           0.           0.           6.70588235]]

Matrice BT :

[[ 5.        ]
 [ 0.        ]
 [-5.5       ]
 [-5.75630252]]
```

Les instructions et fonctions du langage Scilab

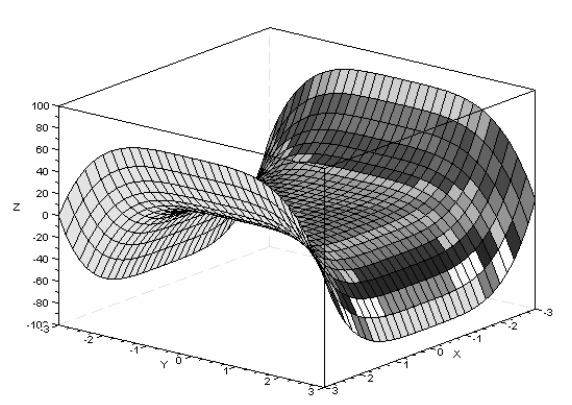
FPLOT3D1

La fonction fplot3d1() permet de tracer la représentation graphique d'une surface définie par une fonction.

```
Editeur :
deff('z = f(x,y)','z = x^4-y^4')
x = -3:0.2:3;
y = x;
clf()
fplot3d1(x,y,f)
```

Remarques :
x = -3:0.2:3 : retourne un vecteur x comportant des nombres de -3 à 3 avec un pas de 0.2, c'est-à-dire : x = [-3 , -2.8 , -2.6 , ……. , 2.6, 2.8, 3]

Affichage de la surface :



FUNCTION

L'instruction function permet de définir une fonction utilisateur.

```
function [arguments_sortie] = nom_de_la_fonction(arguments_entrée)
{instructions}
endfunction
```

```
Editeur :
function z=f(x,y)
z = x^4-y^4;
endfunction

disp(f(3,2))
```

```
Console :
 65
```

Les instructions et fonctions du langage Python

LEN

La fonction len() renvoie la longueur d'une chaîne de caractères.
Exemple :

```
Console :
>>> len("bonjour")
7
>>> len("bonjour à tous")
14
```

LINSPACE

La fonction linspace() permet de créer un vecteur de valeurs équidistantes.

```
x = linspace(valeur_initiale , valeur_finale , nombre_de_valeurs)
```

Exemple :

```
import numpy as np
x = np.linspace(-2,3,5)
print(x)

Console :
>>>
[-2.   -0.75  0.5   1.75  3. ]
```

ONES

La fonction ones() permet de générer une matrice dont les coefficients sont tous égaux à 1.
Exemple :

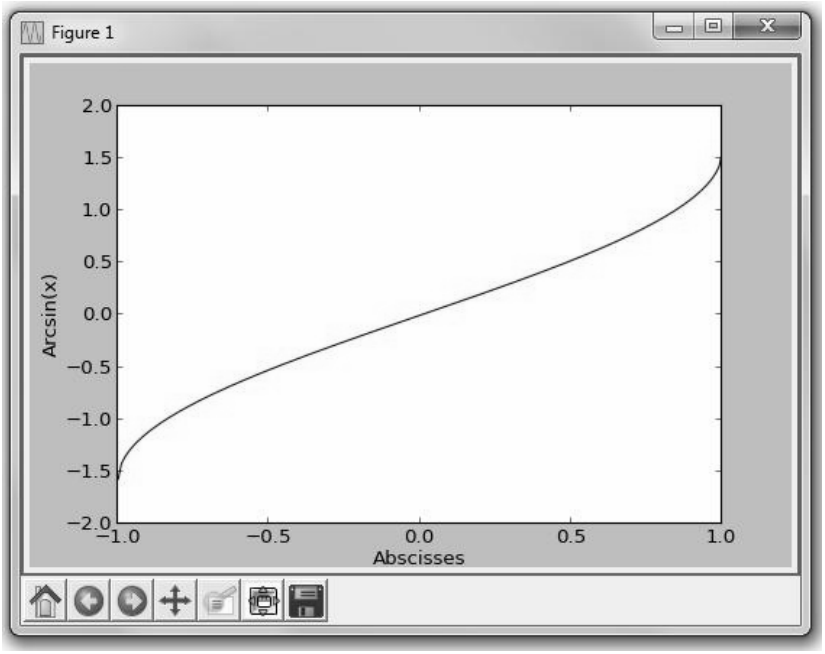
```
import numpy as np
Mat = np.ones((4,4))
print(Mat)

Console :
>>>
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]
```

PLOT

La fonction plot() permet de tracer la courbe représentative d'une fonction donnée ou de représenter un ensemble de points.
Exemple :

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-1,1,200)

plt.plot(x,np.arcsin(x))
plt.ylabel('Arcsin(x)')
plt.xlabel("Abscisses")
plt.show()
```

Remarque : Il est possible d'exporter la figure sous plusieurs formats (png, eps, pdf, etc.)



Graphique généré par la fonction plot()

PRINT

La fonction print() affiche un résultat dans la console.
Exemple :

```
a = 14
print("Valeur de a : ",a)

Console :
>>>
Valeur de a :  14
```

RANGE

La fonction range() permet de créer une liste de valeurs.
Exemple :

```
for k in range(5,10):
    print(k)

Console
>>>
5
6
7
8
9
```