CI 2 : ALGORITHMIQUE & PROGRAMMATION

CHAPITRE 2 – INTRODUCTION À L'ALGORITHMIQUE

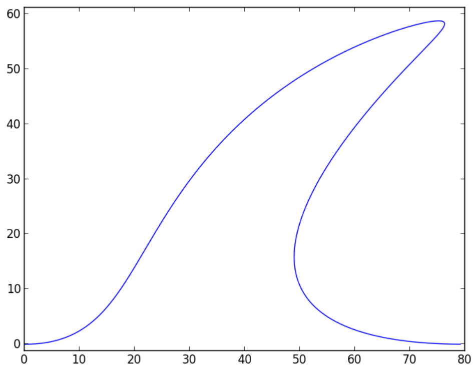
Les courbes de Bézier sont des courbes paramétriques définies par n points P_i de coordonnées (x_i, y_i) appelés pôles. Ainsi, les coordonnées d'un point $M(u) = (x(u), y(u))$ appartenant à la courbe sont définies par :

$$\forall u \in [0,1] \quad \begin{cases} x(u) = \sum_{i=0}^{n} B_i^n(u) x_i \\ y(u) = \sum_{i=0}^{n} B_i^n(u) y_i \end{cases}$$

Avec :

$$B_i^n(u) = \begin{pmatrix} n \\ i \end{pmatrix} u^i (1-u)^{n-i}, \begin{pmatrix} n \\ i \end{pmatrix} = \frac{n!}{i!(n-i)!}$$

$$\forall n \in \mathbb{N}^*, n! = \prod_{i=1}^{n} i \text{ et } 0! = 1$$



Tracé d'une courbe de Bézier

Le tracé de ces courbes fait appel à la définition de fonctions, de boucles, d'instructions conditionnelles qui sont au cœur du développement de programmes informatiques.

Le but de ce cours est de définir les instructions de base qui doivent permettre la réalisation d'algorithmes.

Savoir

SAVOIRS :
- Instructions conditionnelles
- Instructions itératives
- Fonctions

Ce document évolue. Merci de signaler toutes erreurs ou coquilles.

1 Syntaxe

1.1 Sémantique

Définition

Lors de l'exécution d'un programme, les instructions s'exécutent les unes après les autres dans leur ordre d'écriture.

Une séquence d'instructions s'appelle un bloc d'instruction.

Exemple

Le pseudo-code (ou pseudo langage) est une représentation non normalisée d'un algorithme ou d'une fonction. Il permet d'écrire un algorithme en utilisant le langage naturel.

Pseudo Code

Début Fonction
Instruction 1
...
Instruction 2
Fin

En python, les instructions peuvent être séparées par des retours à la ligne, des points-virgules. Dans le cas des blocs d'instructions, les instructions sont **indentées** (4 espaces).

python

```
Instruction1
Instruction2
Instruction1 ; Instruction2
Bloc :
    Instruction 1
    Instruction 2
```

En Scilab : une virgule, un point virgule, un retour à la ligne peuvent séparer des instructions. (Lorsque les instructions sont terminées par un point virgule, les résultats des instructions ne sont pas affichées à l'écran.)

Les blocs sont terminés par end.

Scilab

```
Instruction1
Instruction2
Instruction1 , Instruction2
Instruction1 ; Instruction2 ;
Debut Bloc
Instruction 1
Instruction 2
end
```

1.2 Définition de fonctions

1.2.1 Les fonctions

Lors de l'exécution d'un programme, il est très courant qu'une même séquence d'instruction soit répété un grand nombre de fois. Ainsi, il est courant de décomposer un problème sous forme de plusieurs sous programmes élémentaires.

Exemple

Courbes de Bézier d'ordre 2

Prenons le cas où nous souhaitons connaître les coordonnées d'un point appartenant à une courbe de Bézier définie par 3 points. Dans ce cas,

$$\forall u \in [0,1] \quad \begin{cases} x(u) = (1-u)^2 x_0 + 2u(1-u)x_1 + u^2 x_2 \\ y(u) = (1-u)^2 y_0 + 2u(1-u)y_1 + u^2 y_2 \end{cases}$$

Exemple

Écrivons la fonction permettant d'évaluer une coordonnée en un paramètre donné. Autrement dit :

$$\forall u \in [0,1] f(u) = (1-u)^2 x_0 + 2u(1-u)x_1 + u^2 x_2$$

Pseudo Code

Données : u, x_0, x_1, x_2
Début Fonction
  Fonction $f(u, x_0, x_1, x_2)$:
  $val \leftarrow (1-u)^2 x_0 + 2u(1-u)x_1 + u^2 x_2$
  retourner val
Fin
print $f(0.5, 0, 1, 2)$

python

```python
def f(u,x0,x1,x2):
    val = (1-u)**2*x0 + 2*u*(1-u)*x1 + u**2*x2
    return val

print(f(0.5,0,1,2))
```

Scilab

```
function [val]=f(u,x0,x1,x2)
  val=(1-u)**2*x0+2*u*(1-u)*x1+u**2*x2;
endfunction

printf("%f",f(0.5,0,1,2))
```

Dans ce cas, rien ne permet de contrôler que u appartient bien à l'intervalle $[0,1]$ et que les arguments $x0$, $x1$, et $x2$, sont bien des nombres réels.

1.2.2 Variables locales – Variables globales

Définition

Visibilité : Une variable globale est définie en dehors de toute fonction, une variable locale est définie dans une fonction et masque toute autre variable portant le même nom.

Durée de vie : Une variable globale existe durant l'exécution du programme, une variable locale existe durant l'exécution de la fonction.

Par défaut, dans un langage interprété, les variables sont locales à un bloc.

Exemple

Comportement de Python lors de l'utilisation de variables

python

```python
def f(x):
    a=3
    b=2
    res = a*x+b
    return res
a=4
b=f(a)
```

Quel est le but de la fonction f ? Que se passe-t-il à l'écran si elle est exécutée ? Après exécution du programme, que valent a, b et x ?

Exemple

python

```python
def f(x):
    x=x+2
    return x
x=4
y=f(x)
```

Quel est le but de la fonction f ? Que se passe-t-il à l'écran si elle est exécutée ? Après exécution du programme, que valent x et y ?

python

```python
def f(x):
    x[0]=0
    return x
x=[1,2,3]
y=f(x)
```

Quel est le but de la fonction f ? Que se passe-t-il à l'écran si elle est exécutée ? Après exécution du programme, que valent a, b et x ?

Exemple

Exemples de variables globales - Courbes de Bézier d'ordre 2

On reprend le cas précédent. On se place dans le cas où le programme n'utilise que des courbes Bézier d'ordre 2 et où les pôles restent inchangés. On souhaite alors définir la fonction f sans avoir à rappeler les coordonnées de chacun des pôles.

Pseudo Code

Données : u
Données : Global : $x_0 \leftarrow 0, x_1 \leftarrow 1, x_2 \leftarrow 2$
Début Fonction
  Fonction $f(u)$:
  $val \leftarrow (1-u)^2 x_0 + 2u(1-u)x_1 + u^2 x_2$
  retourner *val*
Fin
print $f(0.5)$

python

```python
def fx(u):
    val = (1-u)**2*x0 + 2*u*(1-u)*x1 + u**2*x2
    return val

global x0,x1,x2
x0,x1,x2=0,1,2

print(fx(0.5))
```

Scilab

```scilab
function [val]=f(u)
  val=(1-u)**2*x0+2*u*(1-u)*x1+u**2*x2;
endfunction

global x0 x1 x2
x0=0;x1=1;x2=2;

 printf("%f",f(0.5))
```

Exemple

Remarque

De manière générale, on essaiera d'utiliser le moins possible les variables globales.

1.2.3 Documentation des fonctions

En programmation, il est indispensable de documenter les fonctions. En effet, il n'est pas toujours facile de se replonger dans un algorithme qu'on a écrit et il est indispensable de le ponctuer de commentaires pour pouvoir bien comprendre le but d'une fonction, d'une boucle *etc.*

En python un commentaire court commence par le signe #. Les commentaires longs sont encadrés par trois guillemets ".

```python
# ======= Debut de la definition des fonctions =======
def f(u,x0,x1,x2):
    """
    Retourne la coordonnee d'un point pour une courbe de Bezier d'ordre 2
    Keyword arguments:
    u -- parametre de la courbe parametree (doit etre compris entre 0 et 1)
    x0 -- coordonnee du pole 0 (sur x, y ou z)
    x1 -- coordonnee du pole 1 (sur x, y ou z)
    x2 -- coordonnee du pole 2 (sur x, y ou z)
    """
    val = (1-u)**2*x0 + 2*u*(1-u)*x1 + u**2*x2
    return val
# ======= Fin de la definition des fonctions =======
```

Lorsqu'une fonction est commentée comme dans l'exemple ci-dessus, on peut accéder à de la documentation sur la fonction en procédant ainsi :

```python
>>> help(f)
>>> f.__doc__
```

Enfin, sous linux, il est possible de générer automatiquement de la documentation au format html :

```
pydoc -w ./ExempleCours.py
```

En scilab un commentaire commence par un double slash : //.

```scilab
// La fonction f etourne la coordonnee d'un point pour une courbe de Bezier d'ordre 2
//     * u : parametre
//     * x0, x1, x2 coordonnees des poles 0 1 et 2
function [val]=f(u,x0,x1,x2)
  val=(1-u)**2*x0+2*u*(1-u)*x1+u**2*x2;
endfunction
```

1.3 Import de méthodes et de fonctions

Par défaut, Python ne permet que de réaliser des opérations élémentaires (opérations mathématiques élémentaires, comparaisons, boucles *etc.*).

Il existe par ailleurs un grand nombre de bibliothèques permettant par exemple de manipuler des fonctions mathématiques (sin, cos, $\sqrt{}$ *etc.*), des bibliothèques permettant de tracer des courbes, des bibliothèques permettant d'interroger des bases de données *etc.*

Pour utiliser les méthodes liées à ces bibliothèques, on procèdera ainsi :

python

```python
import math # Import de toutes les methodes de la bibliotheque math
math.sqrt(2) # Permet d'utiliser la methode sqrt de la bibliotheque math
from math import sqrt # Import de la methode sqrt de la bibliotheque math

import os # Import de la bibliotheque os permettant de realiser des operations systemes
```

Attention

Il est déconseillé d'utiliser la méthode d'import suivante :

```python
from os import *
```

En effet, si des méthodes ont le même nom, seules les méthodes de la dernière bibliothèque appelée ou de la dernière fonction nommée sont utilisables.

Exemple

Fonction pow

2 Instructions conditionnelles

2.1 Expressions booléennes

Définition

Une expression booléenne est une instruction qui renvoie la valeur "vrai" ou "faux".

Définition

Opérateurs de comparaison

Pseudo Code

$2 = 8$
$2 \neq 8$
$2 \geq 8$
$2 > 8$
$2 \leq 8$
$2 < 8$

python

```python
>>> 2==8
    False
>>> 2!=8
    True
>>> 2>=8
    False
>>> 2>8
    False
>>> 2<=8
    True
>>> 2<8
    True
```

Scilab

```
-->2==8
    F
-->2!=8
    T
-->2>=8
    F
-->2>8
    F
-->2<=8
    T
-->2<8
    T
```

2.2 Boucle *Tant que*

Définition

La boucle Tant que appelée aussi boucle while permet de répéter une instruction tant qu'une condition reste vraie.

Remarque

Les boucles ont la plupart du temps besoin d'être incrémentées. Pour cela plusieurs solutions sont possibles.

Pseudo Code

$i \leftarrow 1$
$i \leftarrow i+1$

python

```
>>> i=1
>>> i=i+1;print(i)
    2
>>> i+=1;print(i)
    3
>>> i+=2;print(i)
    5
```

Scilab

```
--> i=1;
--> i=i+1;
```

Exemple

Implémentation de la fonction "factorielle"

On peut définir la fonction factorielle ainsi :

$$\forall n \in \mathbb{N} \begin{cases} \text{si } n=0 \quad n!=1 \\ \text{sinon } n!=\prod_{i=1}^{n} i \end{cases}$$

On s'intéresse à la programmation du cas où n est supérieur ou égal à 1.

Pseudo Code

Début Fonction
  factorielle(n) :
  $i \leftarrow 1$
  $res \leftarrow 1$
  tant que $i \leq n$
  faire
    $res \leftarrow res \cdot i$
    $i \leftarrow i+1$
  fin
  retourner *res*
Fin

python

```
def factorielle(n):
    i=1
    res=1
    while i<=n:
        res=res*i
        i+=1
    return res
```

Scilab

```
function [res]= factorielle (n)
  i=1;
  res=1;
  while i<=n
    res = res*i
    i=i+1
  end
endfunction
```

Attention

Lors de la réalisation d'une boucle *while* il faut veiller à ce que l'instruction conditionnelle change d'état afin de sortir de la boucle et de ne pas provoquer une boucle sans fin...

2.3 Instruction *Si, Sinon*

Définition

La boucle Si appelée aussi boucle if permet d'exécuter une instruction si une condition est vraie.

Exemple

Implémentation de la fonction "factorielle"

On souhaite maintenant gérer le cas où $n = 0$. Dans ce cas, le calcul de $n!$ est différent du cas où $n > 0$.

Pseudo Code

Début Fonction
  factorielle(n) :
  si n=0 **alors**
    retourner *1*
  sinon
    $i \leftarrow 1$
    $res \leftarrow 1$
    tant que $i \leq n$ **faire**
      $res \leftarrow res \cdot i$
      $i \leftarrow i+1$
    fin
    retourner *res*
  fin
Fin

python

```python
def factorielle(n):
    if n==0:
        return 1
    else :
        i=1
        res=1
        while i<=n:
            res=res*i
            i+=1
        return res
```

Scilab

```scilab
function [res]= factorielle(n)
  if n==0
    res =1;
  else
    i=1;
    res=1;
    while i<=n
      res = res*i
      i=i+1
    end
  end
endfunction
```

Remarque : Il faudrait vérifier que n est bien un **entier positif ou nul**.

Exemple

Implémentation de la fonction "factorielle"

On souhaite maintenant s'assurer que n est bien un **entier positif ou nul**.

python

```python
def calc_factorielle(n):
    if (type(n)==int) & (n>=0):
        return factorielle(n)
    else :
        print("Oooops... il faut saisir un nombre entier POSITIF ou nul")
```

Remarque : Il faudrait vérifier que n est du bon type.

La gestion des erreurs

Exemple

Implémentation de la fonction "factorielle"

On souhaite maintenant s'assurer que n est du bon type.

Exemple

```python
def calc_factorielle(n):
    try:
        if (type(n)==int) & (n>=0):
            return( factorielle(n))
        else :
            print("Oooops... il faut saisir un nombre entier POSITIF ou nul")
    except TypeError:
        print("Oooops... le type de la variable n'est pas le bon")
```

Remarque

Pour aller plus loin : on peut définir ses propres exceptions et gérer les messages d'erreur.

```python
class MonException(Exception):
    def __init__(self,raison):
        self.raison = raison

    def __str__(self):
        return self.raison

def calc_factorielle(n):
    if n > 20:
        raise MonException("Il faut saisir un entier positif ou nul")
    else:
        return factorielle(n)
```

3 Instructions itératives

Définition

Une instruction itérative permet de répéter une suite d'instructions un nombre déterminé de fois. On parle aussi de boucle for.

Exemple

Courbes de Bézier d'ordre 2

Nous avons précédemment étudié la fonction permettant de calculer l'abscisse ou l'ordonnée d'un point d'une courbe de Bézier.

Pour afficher une telle courbe une solution consiste en calculer les coordonnées d'un nombre n de points et de relier ces points par des segments de droite. Plus le nombre de points sera élevé plus la courbe paraîtra lisse, mais le temps de calcul sera d'autant plus élevé. On rappelle qu'une courbe de Bézier est une courbe paramétrique définie pour $u \in [0;1]$ et que la fonction f est définie par $f(u) = (1-u)^2 x_0 + 2u(1-u)x_1 + u^2 x_2$.

Il va falloir **discrétiser** l'intervalle $[1,0]$.

Exemple

Pseudo Code

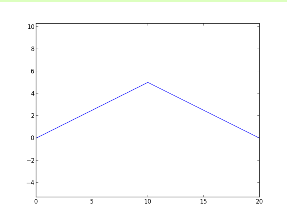
Tableau x ;
Tableau y ;
$x_0 \leftarrow 0$; $y_0 \leftarrow 0$
$x_1 \leftarrow 10$; $y_1 \leftarrow 10$
$x_2 \leftarrow 20$; $y_2 \leftarrow 0$
$i \leftarrow 0$; $n \leftarrow 50$
pour *i de 0 à n* **faire**
  $u \leftarrow i/(n-1)$
  $x[i] \leftarrow f(u, x_0, x_1, x_2)$
  $y[i] \leftarrow f(u, y_0, y_1, y_2)$
  $i \leftarrow i+1$
fin
Afficher(x, y)

python

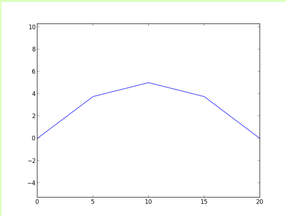
```python
import numpy as np
import pylab as pl

x0=0;y0=0
x1=10;y1=10
x2=20;y2=0
n=50
x,y =[],[]
for i in range(0,n):
    u=i/(n-1)
    x.append(f(u,x0,x1,x2))
    y.append(f(u,y0,y1,y2))
pl.plot(x,y)
pl.axis('equal')
pl.show()
```

Scilab

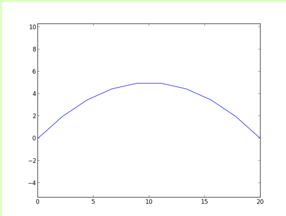
```
x0=0;y0=0;
x1=10;y1=10;
x2=20;y2=0;
n=50;
for i=1:n
  u=i/n;
  x(i)=f(u,x0,x1,x2);
  y(i)=f(u,y0,y1,y2);
end
plot2d(x,y) // a  verifier
```



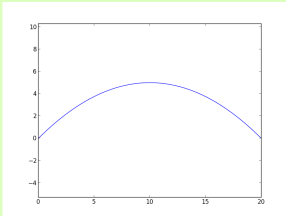
$n = 3$



$n = 5$



$n = 10$



$n = 50$

Remarque

En python, range permet de définir la liste des valeurs qui vont être utilisées lors du parcours de la boucle for. Cette fonction peut prendre jusqu'à 3 arguments : le premier argument désigne la valeur de départ, le second la valeur de fin (exclue), la troisième la valeur de l'incrément.

Remarque

La boucle while définie précédemment est aussi une instruction itérative.

Exemple

Courbes de Bézier d'ordre 2

python

```python
inc = 0.1
while u<=1 :
    x.append(f(u,x0,x1,x2))
    y.append(f(u,y0,y1,y2))
    u += inc
```

Exemple

```
inc =0.1; i =1;
u=0;
x =[]; y =[];
while u<=1
    x(i)=f(u,x0,x1,x2);
    y(i)=f(u,y0,y1,y2);
    u=u+inc;
    i=i+1;
end
```