# Chapitre 9

# ALGORITHMES EXIGIBLES

## A. Algorithmes du chapitre 1 (suites)

### ALGO 1 : Recherche de seuil

■ **Programme 1 (suite croissante explicite de limite $+\infty$)**

Le programme détermine à partir de quel rang n, la suite croissante $(u_n)$ définie par $u_n = 2n+1$ de limite $+\infty$ vérifie $u_n > \text{Seuil}$.

```
def u(n):
    return 2*n+1

def seuil(M):
    i=0
    while u(i)<=M:
        i=i+1
    print("u(n)>",M,"dès que n>=",i)
```

■ **Programme 2 (suite décroissante explicite de limite $-\infty$)**

Le programme détermine à partir de quel rang n, la suite décroissante $(u_n)$ définie par $u_n = -2n+8$ de limite $-\infty$ vérifie $u_n < \text{Seuil}$.

```
def u(n):
    return -2*n+8

def seuil(M):
    i=0
    while u(i)>=M:
        i=i+1
    print("u(n)<",M,"dès que n>=",i)
```

■ **Programme 3 (suite croissante définie par récurrence de limite $+\infty$)**

Le programme détermine à partir de quel rang n, la suite croissante $(u_n)$ définie par la relation de récurrence $u_{n+1} = u_n + 4$ et $u_0 = 7{,}5$ , de limite $+\infty$, vérifie $u_n > \text{Seuil}$.

```
File Edit Format Run Options Window Help
def u(n):
    if n==0:
        return 7.5
    return u(n-1)+4

def seuil(M):
    i=0
    while u(i)<=M:
        i=i+1
    print("u(n)>",M,"dès que n>=",i)
```

■ **Programme 4 (suite décroissante définie par récurrence de limite $-\infty$ )**
Le programme détermine à partir de quel rang n, la suite décroissante $(u_n)$ définie par la relation de récurrence $u_{n+1} = u_n - 2$ et $u_0 = 10{,}6$, de limite $-\infty$, vérifie $u_n <$ Seuil .

```
File Edit Format Run Options Window Help
def u(n):
    if n==0:
        return 10.6
    return u(n-1)-2

def seuil(M):
    i=0
    while u(i)>=M:
        i=i+1
    print("u(n)<",M,"dès que n>=",i)
```

■ **Programme 5 (suite croissante explicite de limite finie)**
Le programme détermine à partir de quel rang n, la suite croissante $(u_n)$ définie par $U_n = 4 - \dfrac{2}{(n+1)^2}$ de limite 4 dépasse un certain seuil (comme 3,99 ou 3,999, etc.).

```
File Edit Format Run Options Window Help
Seuil=float(input("Seuil ?"))
n=0
def u(n):
    if n==0:
        return 4-2/1
    else:
        return(4-2/(n+1)**2)
while u(n)<=Seuil:
    n=n+1
print("Un>",Seuil,"dès que n>=",n)
```

■ **Programme 6 (suite décroissante explicite de limite finie)**

Le programme détermine à partir de quel rang n, la suite décroissante $(u_n)$ définie par $u_n = 1 + \frac{3}{n+1}$ de limite 1 passe sous un certain seuil (comme 1,01 ou 1,001, etc.)

```
File Edit Format Run Options Window Help
Seuil=float(input("Seuil ?"))
n=0
def u(n):
    if n==0:
        return 1+3/1
    else:
        return(1+3/(n+1))
while u(n)>=Seuil:
    n=n+1
print("Un<",Seuil,"dès que n>=",n)
```

■ **Programme 7 (suite croissante définie par récurrence de limite finie)**

Le programme détermine à partir de quel rang n, la suite croissante $(u_n)$ définie par $\begin{cases} U_{n+1} = 0,5 \times U_n + 3 \\ U_0 = 3 \end{cases}$ de limite 6 dépasse un certain seuil (comme 5,99 ou 5,999, etc.)

```
File Edit Format Run Options Window Help
Seuil=float(input("Seuil ?"))
n=0
def u(n):
    if n==0:
        return 3
    else:
        return(0.5*u(n-1)+3)
while u(n)<=Seuil:
    n=n+1
print("Un>",Seuil,"dès que n>=",n)
```

■ **Programme 8 (suite décroissante définie par récurrence de limite finie)**

Le programme détermine à partir de quel rang n, la suite décroissante $(u_n)$ définie par $\begin{cases} U_{n+1} = 0,5 \times U_n - 3 \\ U_0 = 2 \end{cases}$ de limite $-6$ passe sous un certain (comme $-5,99$ ou $-5,999$, etc.).

```
Seuil=float(input("Seuil ?"))
n=0
def u(n):
    if n==0:
        return 2
    else:
        return(0.5*u(n-1)-3)
while u(n)>=Seuil:
    n=n+1
print("Un<",Seuil,"dès que n>=",n)
```

## ALGO 2 : Pour une suite récurrente $u_{n+1} = f(u_n)$, calcul des termes successifs

### ■ Principe

Le programme est capable de calculer les termes successifs de la suite $(u_n)$ définie par $u_{n+1} = f(u_n)$ avec $f(x) = 0{,}5x + 3$ et $u_0 = 10$.

```
def f(x):
    return 0.5*x+3

def u(n):
    if n==0:
        return 10
    else:
        return f(u(n-1))

def termes_successifs(n):
    for i in range(0,n+1):
        print(u(i))
```

## ALGO 3 : Recherche des valeurs approchées de constantes mathématiques, par exemple $\pi$, $\ln(2)$, $\sqrt{2}$

### ■ Programme 1 (suite convergente vers $\pi$)

Le programme donne une valeur approchée de $\pi$ suivant la méthode d'Al-Kashi (1380-1429) et des suites $(c_n)$ et $(\pi_n)$ définies par $c_{n+1} = \sqrt{2 - \sqrt{4 - c_n^2}}$ et $\pi_n = 3 \times 2^n \times c_n$.

```
def c(n):
    if n==0:
        return 1
    else:
        return (2-(4-c(n-1)**2)**0.5)**0.5

def pi(n):
    return 3*2**n*c(n)
```

Par exemple, pi(10) donne l'approximation $\pi \simeq 3{,}1415925165\ldots$

### ■ Programme 2 (suite convergente vers $\ln(2)$)

Le programme donne une valeur approchée de $\ln(2)$ à l'aide de la suite $(u_n)$ définie par $u_{n+1} = u_n + \frac{1}{2n+2} + \frac{1}{2n+1} - \frac{1}{n+1}$ et $u_1 = 0{,}5$.

```
def u(n):
    if n==1:
        return 0.5
    else:
        return u(n-1)+1/(2*n)+1/(2*n-1)-1/n
```

Par exemple, u(500) donne l'approximation $\ln(2) \simeq 0{,}69264743055\ldots$

### ■ Programme 3 (suite convergente vers $\sqrt{2}$)

Le programme donne une valeur approchée de $\sqrt{2}$ par la suite de Bombelli (1526-1572) définie par $u_{n+1} = 1 + \frac{1}{1+u_n}$ et $u_0 = 1$.

```
def Bombelli(n):
    if n==0:
        return 1
    else:
        return 1+1/(1+Bombelli(n-1))
```

Par exemple, Bombelli(5) donne $\sqrt{2} \simeq 1{,}414285\ldots$

## B. Algorithmes du chapitre 2 (fonctions)

### ALGO 4 : Recherche de valeurs approchées d'une solution d'équation du type $f(x)=k$ par balayage

■ **Principe**

Le programme est capable de résoudre (suivant la méthode du balayage) de manière approchée l'équation $f(x)=k$ avec une précision donnée.

```
def f(x):
    return x**2
def balayage(f,k,a,b,précision):
    x=a
    while f(x)<=k:
        x=x+précision
    return x
```

### ALGO 5 : Recherche de valeurs approchées d'une solution d'équation du type $f(x)=k$ par dichotomie

■ **Principe**

Le programme est capable de résoudre (suivant la méthode de la dichotomie) de manière approchée l'équation $f(x)=k$ avec une précision donnée.

```
def f(x):
    return x**2
def dichotomie(f,k,a,b,précision):
    while b-a>précision:
        m=(a+b)/2
        if (f(a)-k)*(f(m)-k)<0:
            b=m
        else:
            a=m
    return b
```

## ALGO 6 : Recherche de valeurs approchées d'une solution d'équation du type $f(x)=k$ par la méthode de Newton

■ **Principe**

Le programme est capable de résoudre (suivant la méthode de Newton) de manière approchée l'équation $f(x)=k$ avec une précision donnée.

```
File  Edit  Format  Run  Options  Window  Help
def f(x):
    return x**2
def df(x):
    return 2*x
def Newton(k,précision):
    x=3
    y=x-(f(x)-k)/df(x)
    while abs(y-x)>précision:
        x=y
        y=y-(f(x)-k)/df(y)
    return y
```

# C. Algorithme du chapitre 3 (primitives et équations différentielles)

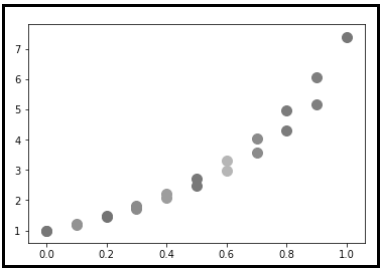
## ALGO 7 : Résolution approchée d'une équation différentielle par la méthode d'Euler

■ **Principe**

Le programme ci-dessous donne une résolution approchée de l'équation $\begin{cases} y' = 2y \\ y(0) = 1 \end{cases}$ à l'aide de la suite $\begin{cases} x_{i+1} = x_i + 0{,}1 \\ y_{i+1} = y_i + 0{,}1 \times 2y_i \end{cases}$ et la compare (graphiquement) avec la solution théorique $x \to y(x) = e^{2x}$.

```
from math import exp
import matplotlib.pyplot as plt
def x(n):
    if n==0:
        return 0
    else:
        return x(n-1)+0.1
def y(n):
    if n==0:
        return 1
    else:
        return y(n-1)+0.1*(2*y(n-1))
for n in range(0,10):
    plt.scatter(x(n),y(n),s=100)
a=0
while a<=1:
    plt.scatter(a,exp(2*a),s=100)
    a=a+0.1
```

Une fois le programme lancé, on obtient :




## D. Algorithmes du chapitre 5 (intégration)

### ALGO 8 : Méthode des rectangles

■ **Principe**

Le programme donne une valeur approchée de l'intégrale $\int_a^b f(x)dx$ (avec $f(x) = x^2$) suivant la méthode des rectangles.

```
def f(x):
    return x**2
def Rectangles(f,a,b,N):
    Pas=(b-a)/N
    Intégrale=0
    for i in range(1,N+1):
        Intégrale=Intégrale+f(a)*Pas
        a=a+Pas
    return Intégrale
```