Projet

April 7, 2024

[1]:
```
from math import *
import matplotlib.pyplot as plt
import numpy as np
import random as rand
```

1 A. Introduction

A ajouter une petite intro pour le rendu

2 B. Optimisation d'une fonction d'une variable réelle

Considérons ici des fonctions continues définies sur $[a, b]$.

Question 1

Méthode par balayage à pas constant :

[2]:
```
def BalConstant (f,a,b,N):
    dx=(b-a)/N
    min = f(a)
    for i in range (1,N+1):
        if f(a+i*dx)<min:
            min = f(a+i*dx)
    return min
```

Méthode par balayage aléatoire :

[3]:
```
def BalAlea (f,a,b,N):
    X=[rand.uniform(a,b) for i in range (N+1)]
    min = f(X[0])
    for x in X:
        if f(x)<min:
            min = f(x)
    return min
```

Question 2

[4]:
```
#Fonction 1
def pol(x):
```

```python
    return x**3 - 3*x**2+2*x+5

# Autre polynome pour test : Fonction 2
def pol2(x):
    return x**4-2*x**3+x**2-5*x+2

#Ces fonctions rentrent bien dans les hypothèse.
```

[5]:
```python
a=0
b=3
N=100
print("1) Méthode par balayage à pas constant:",BalConstant(pol,a,b,N))
print("1) Méthode par balayage aléatoire :",BalAlea(pol,a,b,N))
print("2) Méthode par balayage à pas constant:",BalConstant(pol2,a,b,N))
print("2) Méthode par balayage aléatoire :",BalAlea(pol2,a,b,N))
```

```
1) Méthode par balayage à pas constant: 4.615379000000001
1) Méthode par balayage aléatoire : 4.615645251526421
2) Méthode par balayage à pas constant: -5.099743750000002
2) Méthode par balayage aléatoire : -5.099496635112019
```

Ces valeurs sont cohérentes avec les résultats réels.

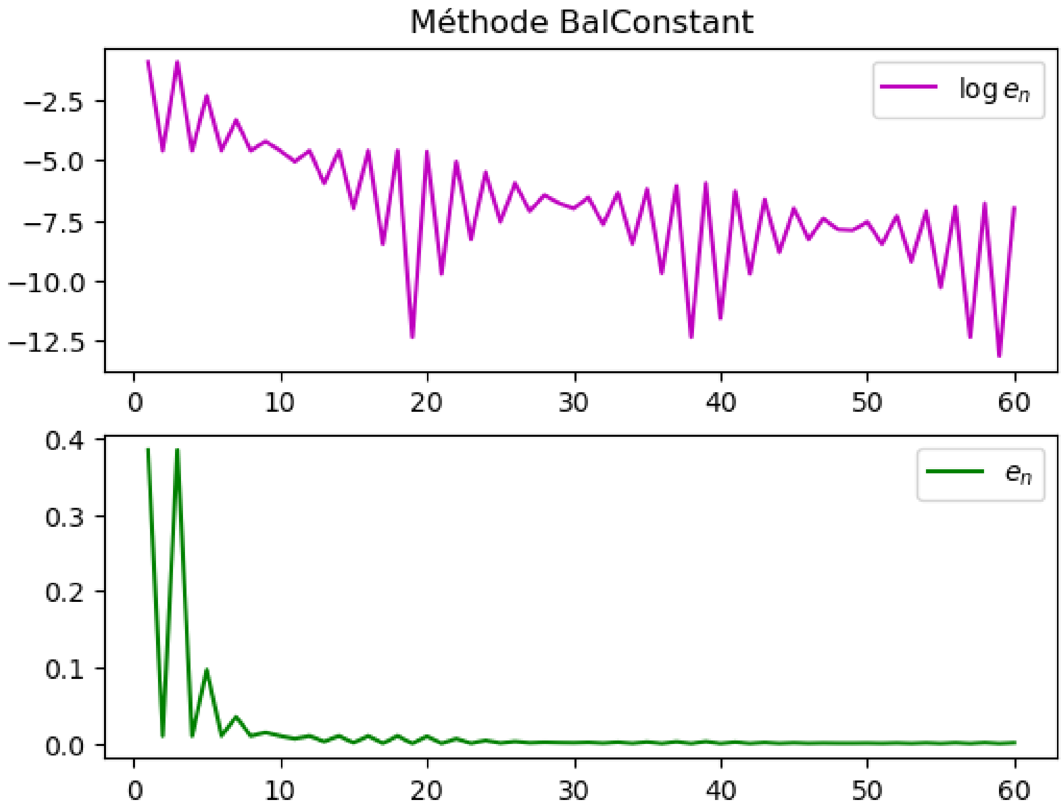
Question 3

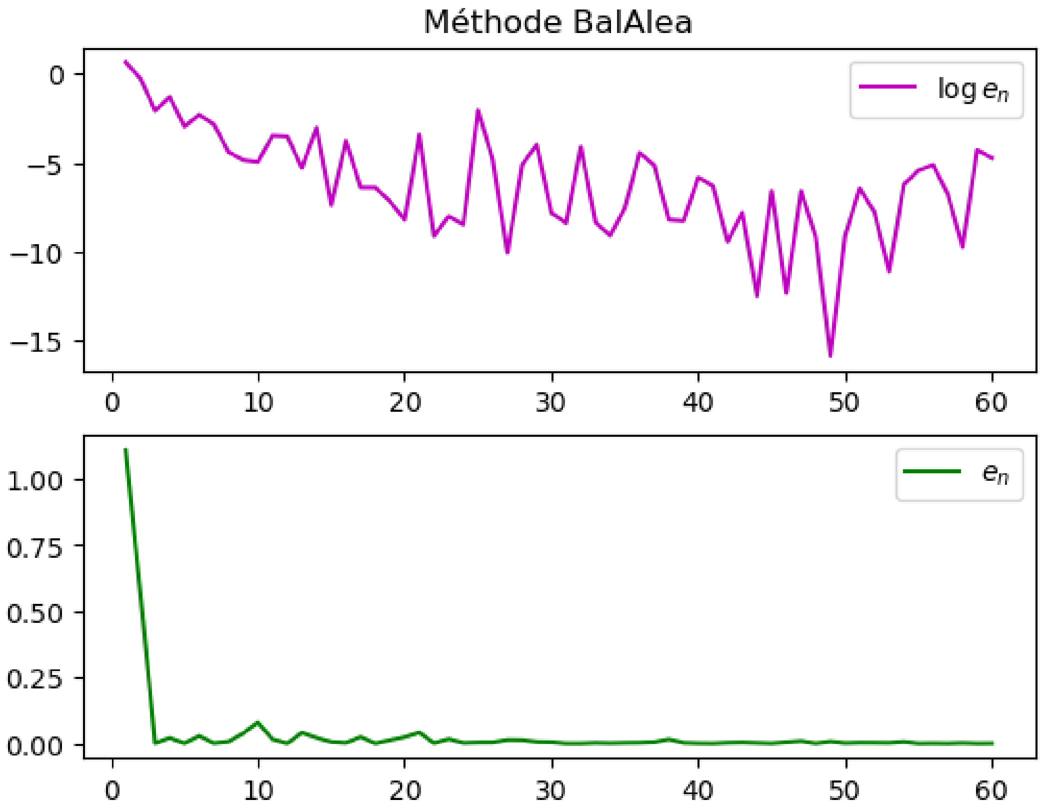
Soit A_N l'approximation du minimum de la fonction et a_r et la valeur réelle du minimum.

Déterminons $e_n = |A_N - a_r|$ pour la fonction $f(x) = x^3 - 3x^2 + 2x + 5$.

[6]:
```python
def erreur(methBal,minReel,Nmax,a,b):
    X=[x for x in range (1,Nmax+1)]
    Y=[]
    Z=[]
    for i in range (1,Nmax+1):
        Z.append(np.abs(methBal(pol,a,b,i)-minReel))
        Y.append(log(np.abs(methBal(pol,a,b,i)-minReel)))
    plt.figure()
    plt.subplot(211)
    plt.title(f"Méthode {methBal.__name__}")
    plt.plot(X,Y,color='m',label="$\log e_n$")
    plt.legend()
    plt.subplot(212)
    plt.plot(X,Z,color="g",label="$e_n$")
    plt.legend()
```

[7]:
```python
minReel= 5 - 2/(3*sqrt(3))
erreur(BalConstant,minReel,60,a,b)
erreur(BalAlea,minReel,60,a,b)
```


Méthode BalConstant
$\log e_n$
−2.5
−5.0
−7.5
−10.0
−12.5
0
10
20
30
40
50
60
e_n
0.4
0.3
0.2
0.1
0.0
0
10
20
30
40
50
60



On constate que la méthode de balayage à pas constant semble plus efficace et plus **fiable**.

En effet, l'erreur semble décroitre et tendre vers 0 de manière *relativement* régulière pour cette méthode tandis que la méthode de balayage aléatoire elle semble plus imprévisible. Dans les faits, on observe aussi une décroissance et une tendance à tendre vers 0 mais avec des pics réguliers, pics qui ne sont pas présent dans l'autre méthode. D'où la conclusion.

Question 4

Modifions légèrement le code de la méthode de balayage à pas constant précédent pour cette fois çi obtenir le maximum de la fonction:

[8]:
```
def BalConstantMax(f,a,b,N):
    dx=(b-a)/N
    max = f(a)
    for i in range (1,N+1):
        if f(a+i*dx)>max:
            max= f(a+i*dx)
    return max
```

[9]:
```
print("1)",BalConstantMax(pol,a,b,N))
print("2)",BalConstantMax(pol2,a,b,N))
```

```
1) 11.0
2) 23.0
```

Ces valeurs sont cohérentes avec les résultats réels.

Question 5

Soit le point d'abscisse x_n. Considérons d'abord le cas où ce dernier est un minimum, alors $x_{n+1} = x_n$ car dans ce cas on aurait $f'(x_n) = 0$.

Maintenant si ce n'est pas le cas, alors en choisissant u assez petit et **négatif** (car sinon on reculerait quand la fonction est décroissante et n'atteindrons donc pas le minimum, mais un potentielle maximum), on avance de manière précise et proportionnel à la pente de la courbe vers le minimum, cette pente étant donc non nulle. Ainsi, en général x_{n+1} est plus proche du minimum que x_n car nous avons avancé proportionellement à la pente et c'est celle-çi qui est indicatrice et qui caractérise un minimimum par le fait qu'elle soit nulle en ce dernier. C'est le cas en général du fait que si le pas u est trop élevée ou que si nous somme extrêment proche du minimum mais pas exactement à ce dernier, il est possible de le dépasser (car $f'(x_n) \neq 0$) et donc de s'en éloigner potentiellement le temps d'une itération et donc d'avoir un contre-exemple.

Question à poser: Possible que si 1 minimum local seulement dans l'intervalle. ??

Question 6

[10]:
```python
#Définition des dérivées des fonctions 1 et 2 précédentes
def df(x):
    return 3*x**2 - 6*x + 2

def df2(x):
    return 4*x**3 - 6*x**2 + 2*x - 5
```

[34]:
```python
#Méthode avec le premier terme choisie manuellement
def Grad1D(f,df,x0,u,N):
    xn=x0
    for i in range(N-1):
        xn1=xn+u*df(xn)
        xn=xn1
    return f(xn)
#Méthode avec premier terme choisie aléatoirement dans l'intervalle
def Grad1DVariante(f,df,a,b,u,N):
    xn=rand.uniform(a,b)
    for i in range(N-1):
        xn1=xn+u*df(xn)
        xn=xn1
    return f(xn)
```

[36]:
```python
print("1)x_0 imposée:",Grad1D(pol,df,1,-0.001,10000))
print("1)x_0 aléatoire:",Grad1DVariante(pol,df,0,3,-0.001,10000))
print("2)x_0 imposée:",Grad1D(pol2,df2,1,-0.001,10000))
print("2)x_0 aléatoire:",Grad1DVariante(pol2,df2,0,3,-0.001,10000))
```

```
1)x_0 imposée: 4.61509982054025
1)x_0 aléatoire: 4.61509982054025
2)x_0 imposée: -5.099892037763601
2)x_0 aléatoire: -5.099892037763601
```

Ces résultats son cohérent.

Question 7

a.

Soit:

$$\varphi(t) = f(x_n + tf'(x_n))$$

On a donc:

$$\varphi'(t) = f'(x_n).f'(x_n + tf'(x_n))$$

Et:

$$\varphi''(t) = (f'(x_n))^2.f''(x_n + tf'(x_n))$$

D'où:

$$\varphi'(0) = (f'(x_n))^2,\ \varphi''(0) = (f'(x_n))^2.f''(x_n)$$

b.

On a donc:

$$\varphi'(t) = (f'(x_n))^2 + (f'(x_n))^2.f''(x_n))t + o_0(t)$$

Il ne reste maintenant plus qu'à chercher t_{opt} tq. $\varphi'(t_{opt}) = 0$ pour trouver $u = t_{opt}$.

D'où:

$$u = -\frac{1}{f''(x_n)}$$

On a donc au final:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

En posant $g(x) = f'(x)$, on retrouve la **méthode de Newton**:

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)}$$

3 C. Optimisation d'une fonction de deux variables réelles

[ ]: