

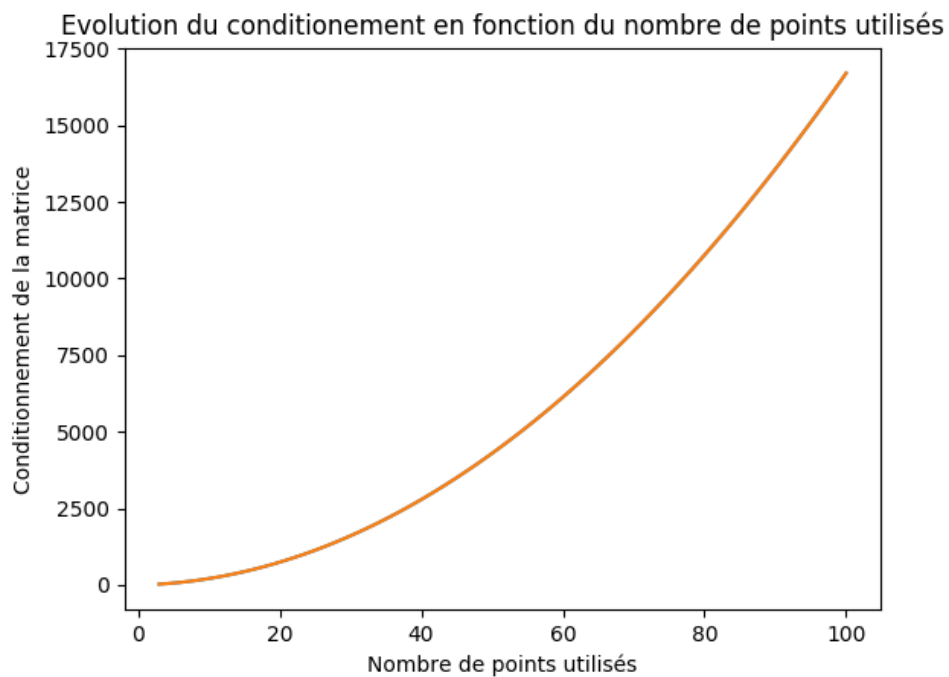
---

*Lorys Le Gohebel, Mechineau Alexandre*

## **Sommaire**

Etude du conditionnement de la matrice .....	2
Etude de l'erreur pour N fixé en fonction de la méthode et de la fonction .....	3
$f(x)=0$ , $N=9$ : .....	3
$f(x)=1$ , $N=9$ : .....	4
$f(x)=x$ , $N=9$ .....	5
$f(x)=x^2$ , $N=9$ .....	6
$f(x)=4\pi^2 \sin(2\pi x)$ , $N=9$ .....	7
Conclusion.....	8
Etude de la convergence des deux schémas.....	9
$f(x)=0$ , $N=9$ : .....	9
$f(x)=1$ , $N=9$ : .....	10
$f(x)=x$ , $N=9$ .....	11
$f(x)=x^2$ , $N=9$ .....	12
$f(x)=4\pi^2 \sin(2\pi x)$ , $N=9$ .....	13
Conclusion : Convergence .....	14
Source : .....	14

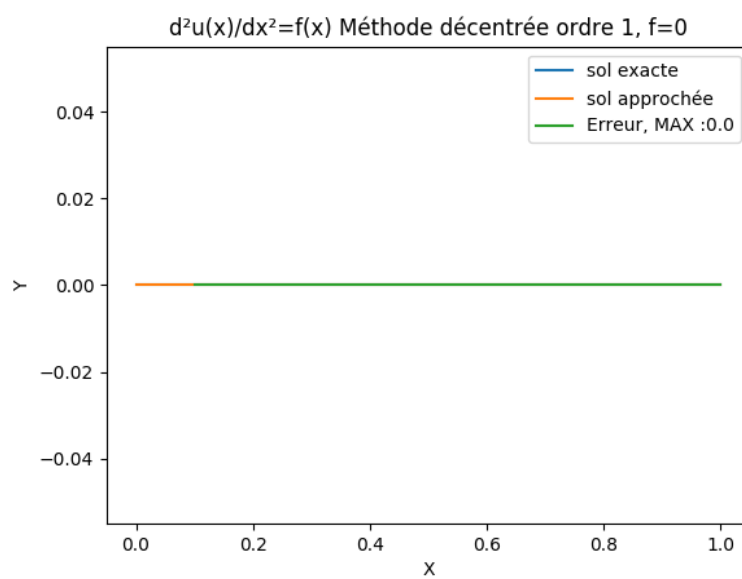
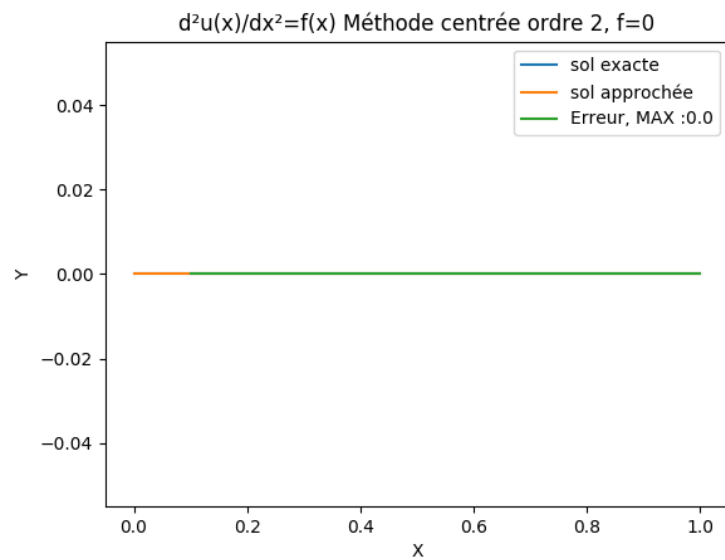
## Etude du conditionnement de la matrice



On remarque donc que la matrice A est mal conditionné. De plus, nous pouvons remarquer que l'erreur n'est pas linéaire ( $O(N)$ ).

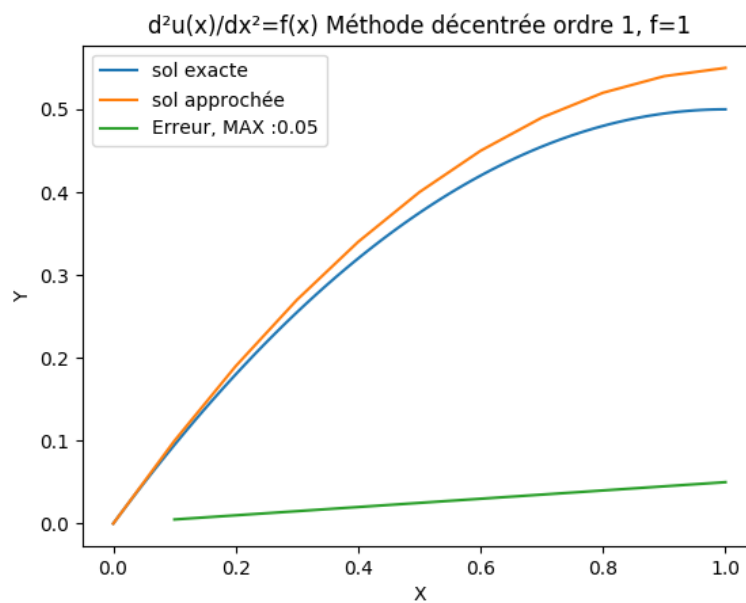
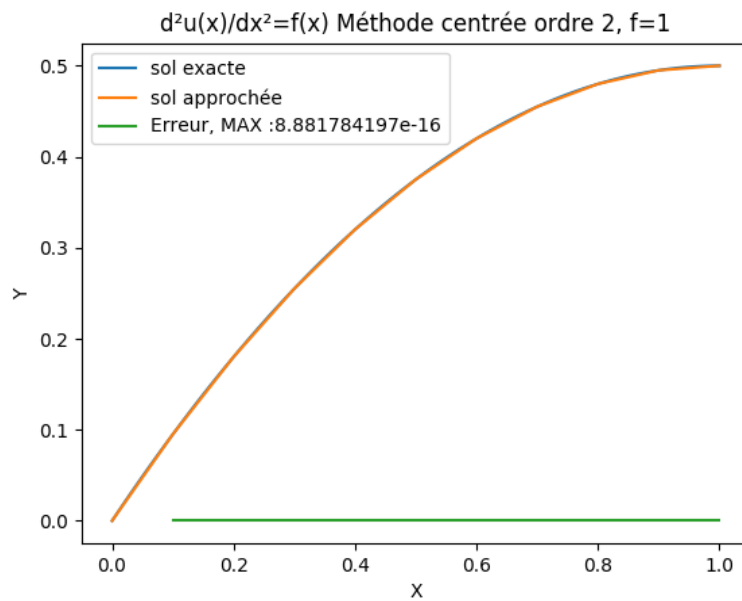
## Etude de l'erreur pour N fixé en fonction de la méthode et de la fonction

$f(x)=0$ ,  $N=9$  :



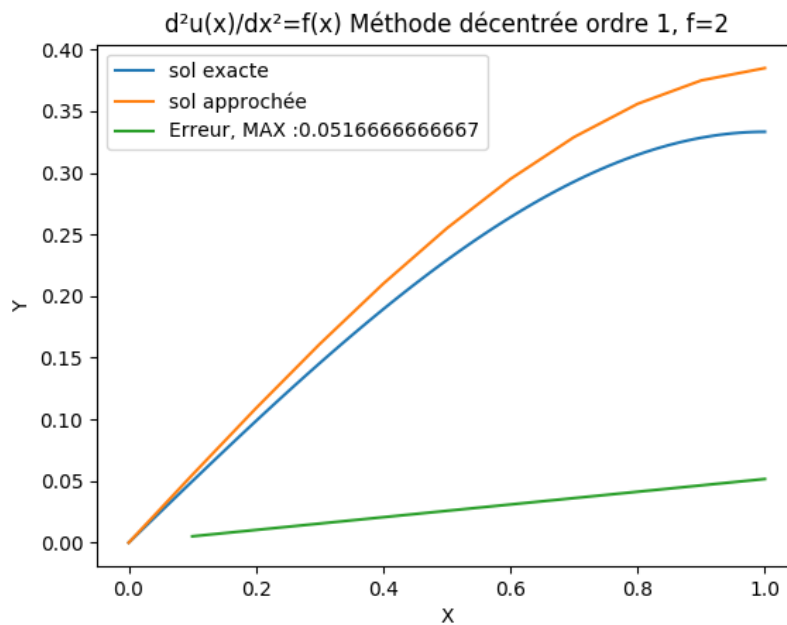
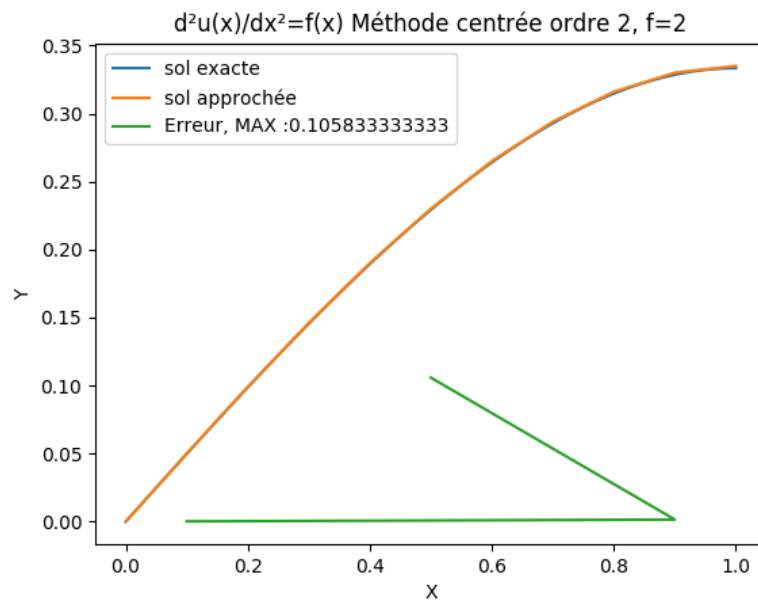
Nous pouvons remarquer que l'erreur est nulle pour les deux méthodes.

$f(x)=1$ ,  $N=9$  :



Pour  $f(x) = 1$ , On remarque la méthode centrée est exacte. Cependant, ce n'est plus le cas pour la méthode décentrée.

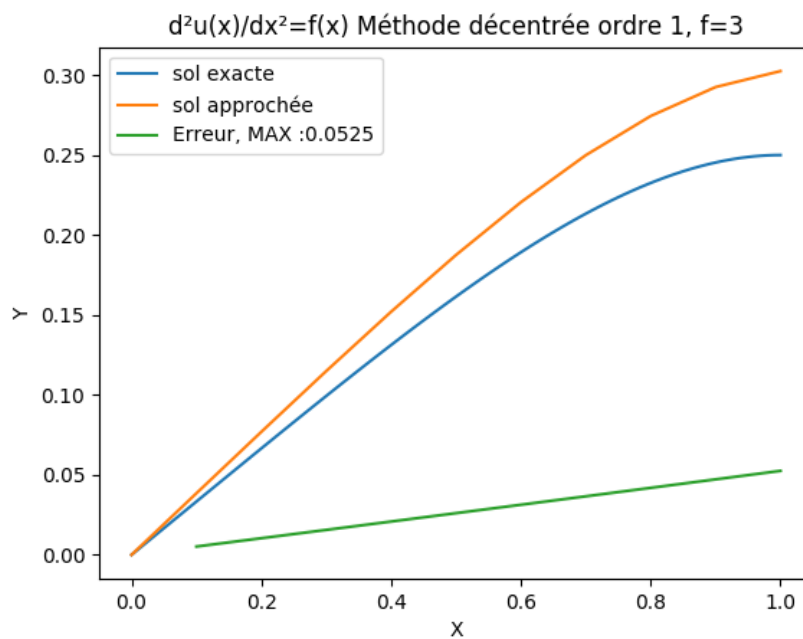
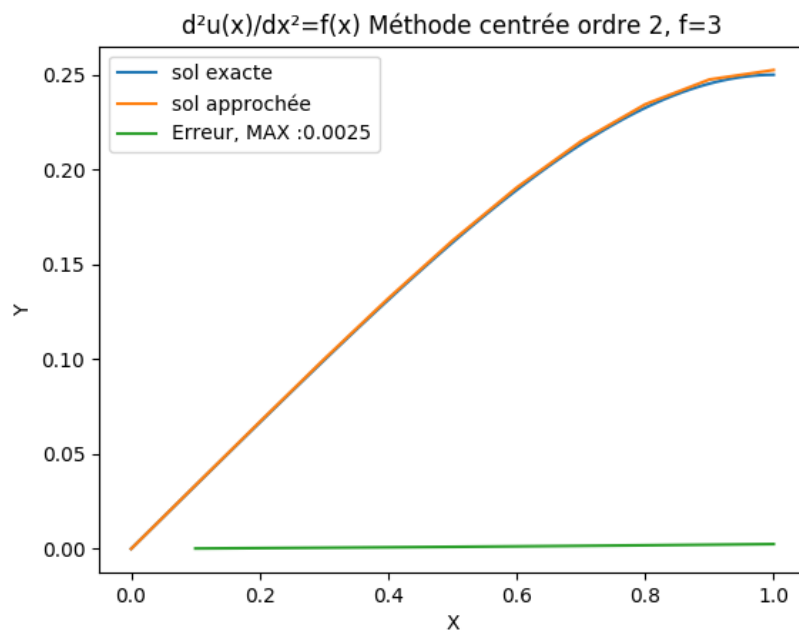
$$f(x)=x, N=9$$



Ici, les deux méthodes présentent des erreurs. La méthode centrée est quand même plus précise que celle décentrée. La première méthode semble avoir une erreur sur le bord.

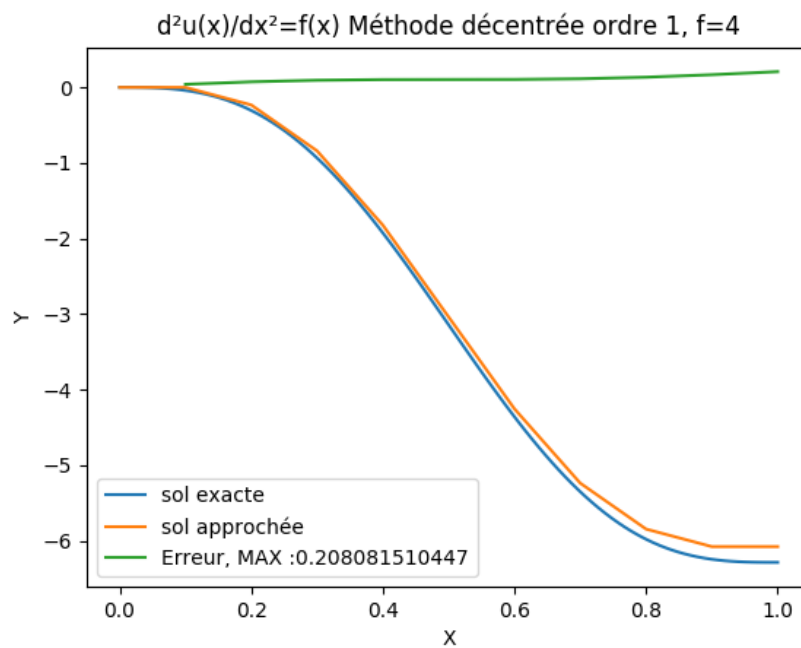
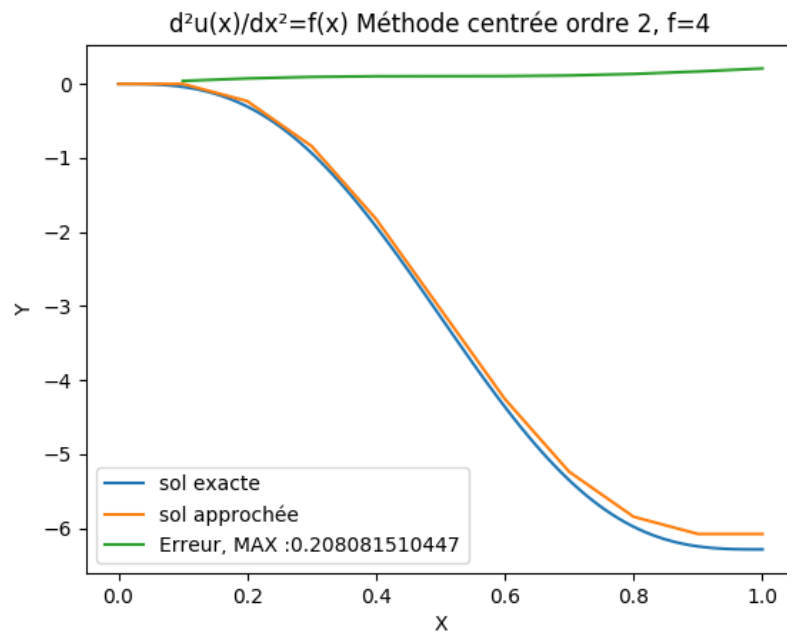
(L'erreur du premier graph a été calculée correctement. Bug python ?)

$$f(x)=x^2, N=9$$



On a le même constat que précédemment, sauf que ici la méthode 1 a une bonne précision partout.

$$f(x)=4\pi^2\sin(2\pi x), N=9$$



Pour cette dernière fonction les deux méthodes ne sont pas assez précises avec  $N=9$  pour avoir un résultat précis.

## Conclusion

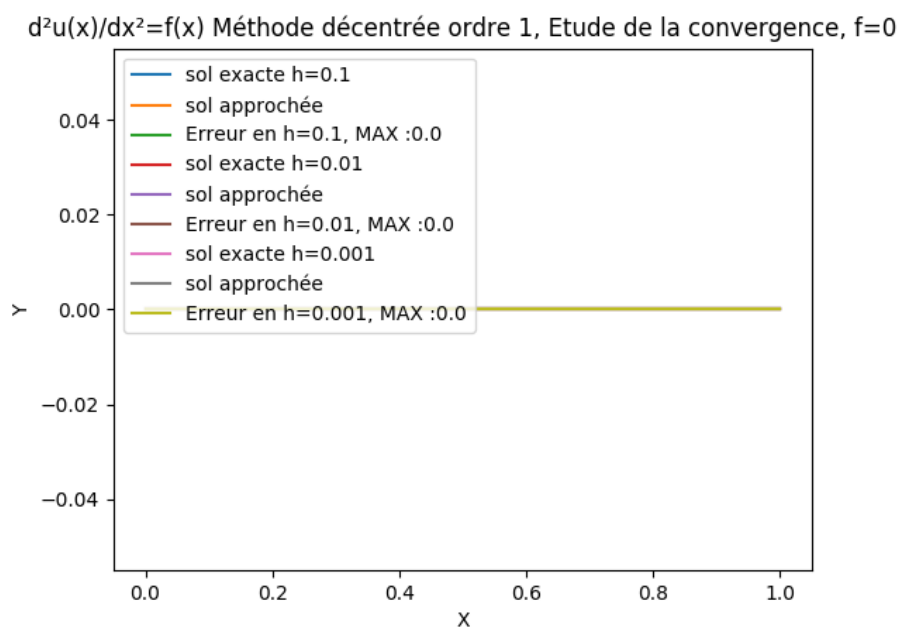
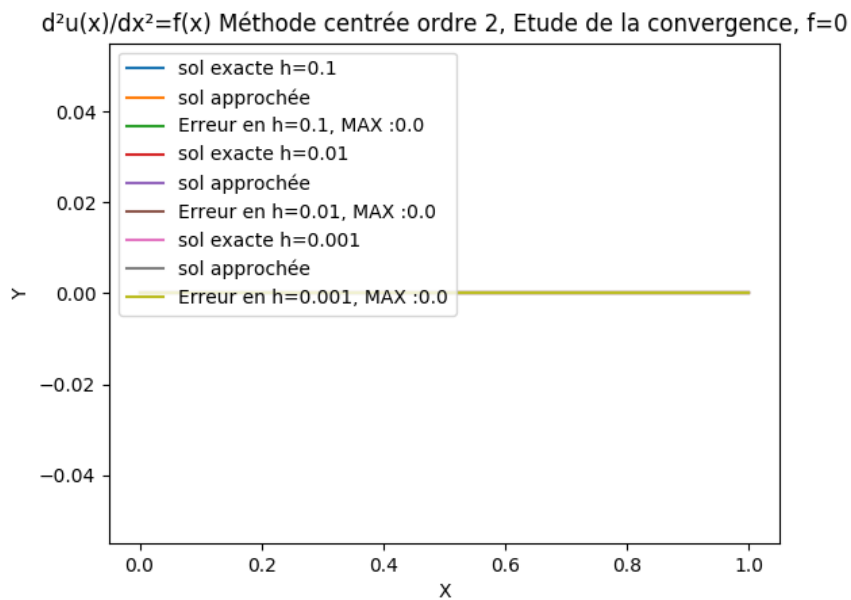
La méthode décentrée ne permet d'être exacte qu'avec  $f(x)=0$ . Alors que la méthode centrée est exacte jusqu'à  $f(x)=1$ .

On remarque tout de même pour la dernière fonction les deux méthodes sont dans l'impossibilité d'obtenir un résultat convenable avec si peu de points.



## Etude de la convergence des deux schémas

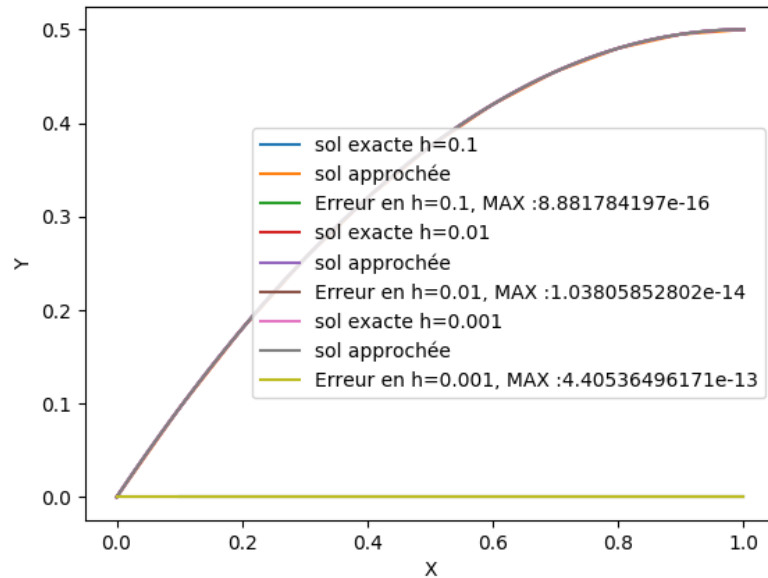
$f(x)=0$ ,  $N=9$  :



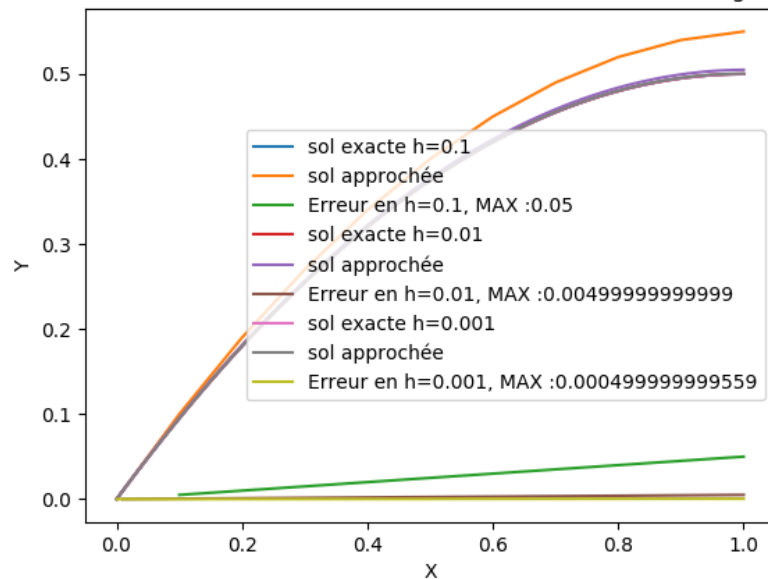
Les deux méthodes sont exactes peu importe le nombre de points.

$f(x)=1$ ,  $N=9$  :

$d^2u(x)/dx^2=f(x)$  Méthode centrée ordre 2, Etude de la convergence,  $f=1$



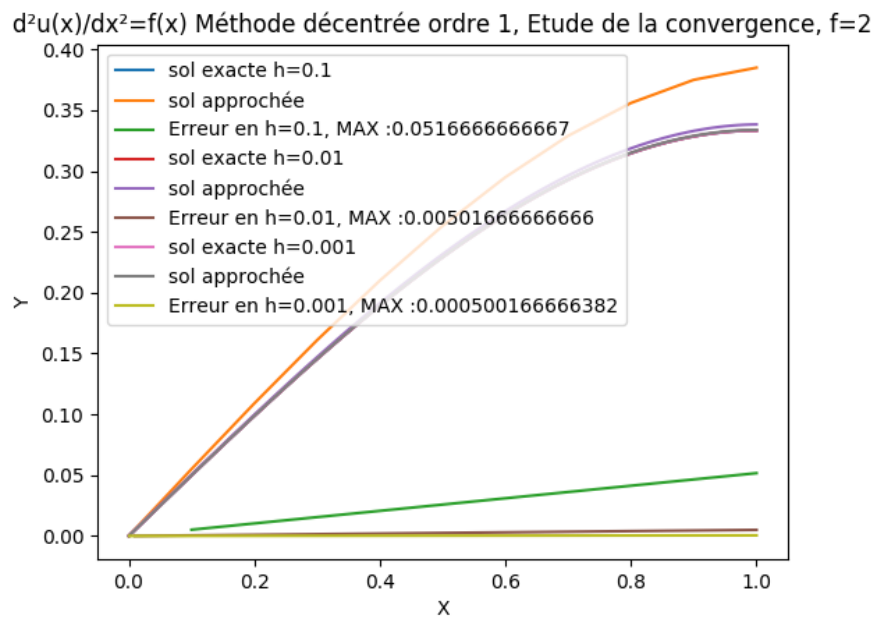
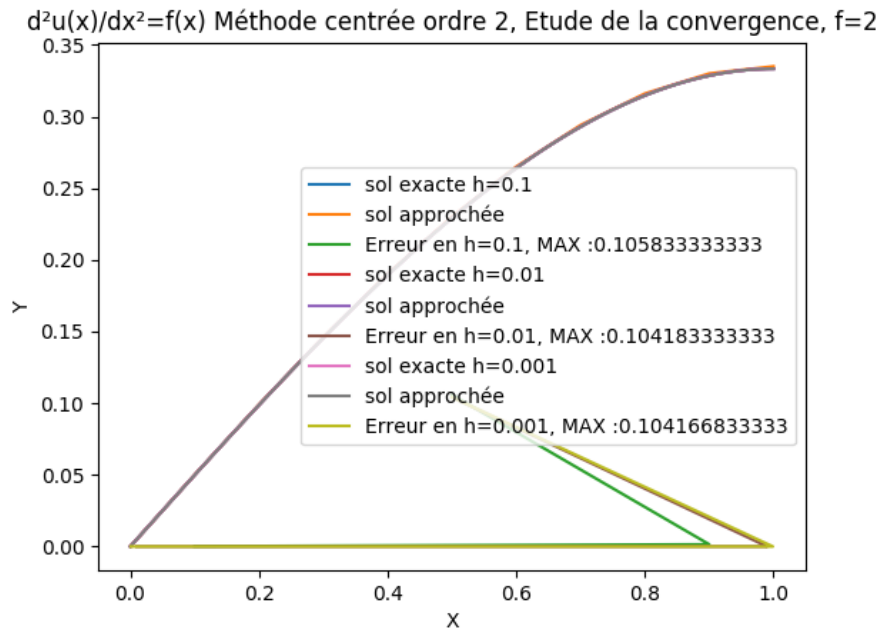
$d^2u(x)/dx^2=f(x)$  Méthode décentrée ordre 1, Etude de la convergence,  $f=1$



La première méthode atteint la précision machine mais si le nombre de points augmente la précision se dégrade.

La deuxième méthode semble présenter une erreur d'ordre  $O(1/N)$ .

$$f(x)=x, N=9$$

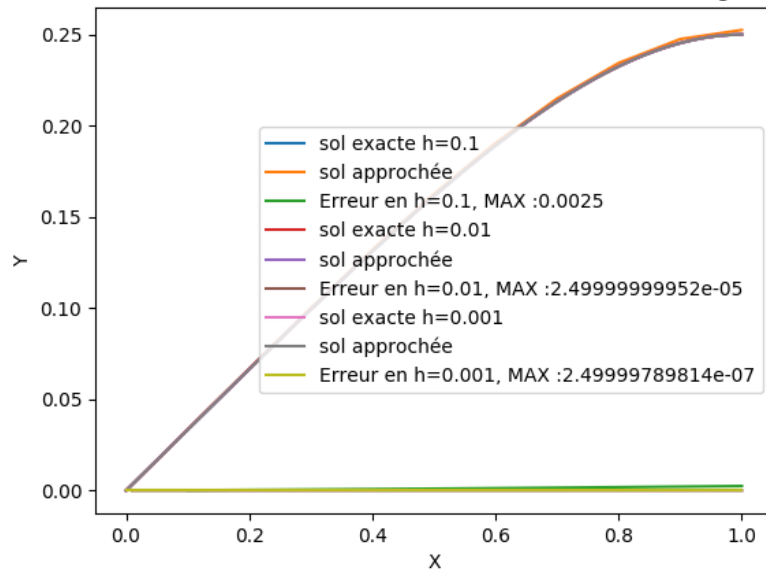


Même problème précédemment pour la première méthode.

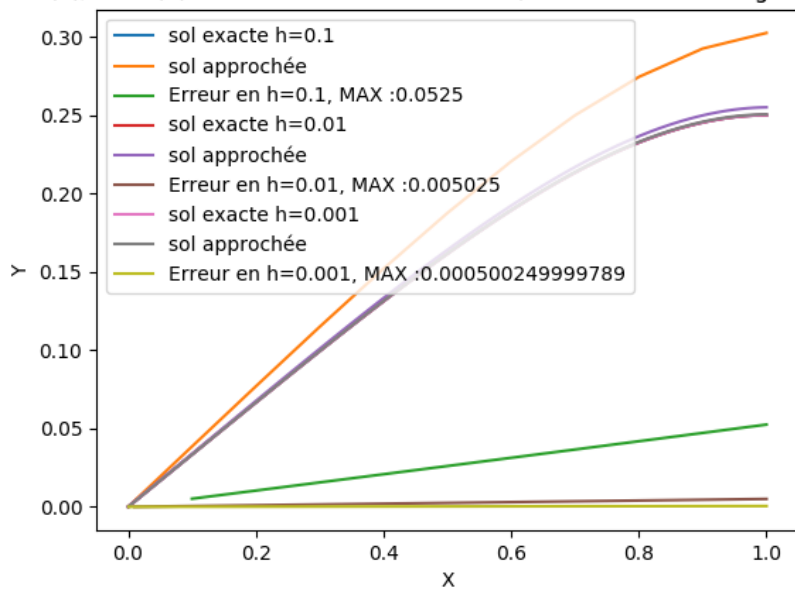
On remarque encore le même comportement pour la seconde méthode.

$$f(x)=x^2, N=9$$

$d^2u(x)/dx^2=f(x)$  Méthode centrée ordre 2, Etude de la convergence,  $f=3$



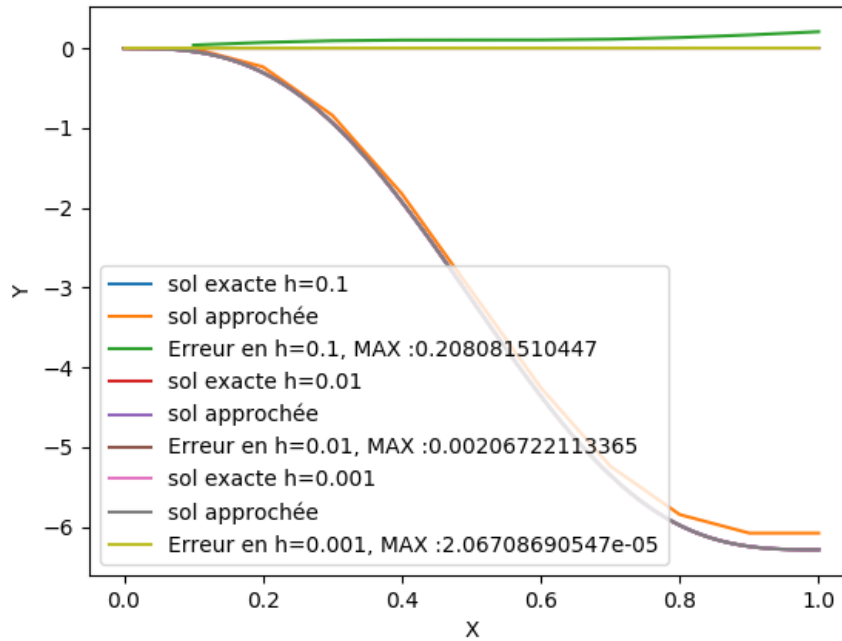
$d^2u(x)/dx^2=f(x)$  Méthode décentrée ordre 1, Etude de la convergence,  $f=3$



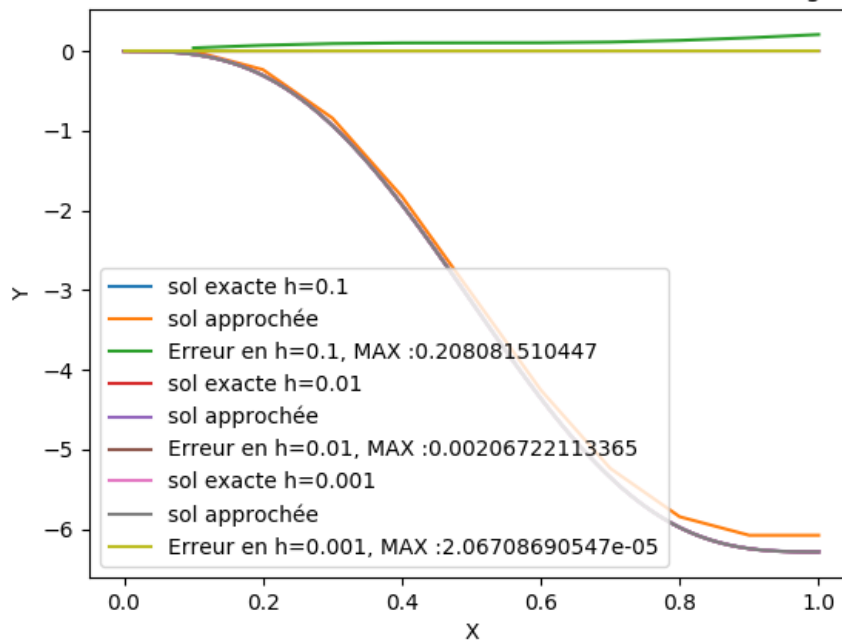
On remarque que première méthode semble avoir une erreur quadratique.

$$f(x)=4\pi^2\sin(2\pi x), N=9$$

$d^2u(x)/dx^2=f(x)$  Méthode centrée ordre 2, Etude de la convergence,  $f=4$



$d^2u(x)/dx^2=f(x)$  Méthode décentrée ordre 1, Etude de la convergence,  $f=4$



Les deux méthodes semblent présenter la même erreur.

## Conclusion : Convergence

La méthode centrée se comporte comme une méthode d'erreur quadratique et la méthode décentrée se comporte comme une erreur linéaire.

Elles se comportent toute fois de la même façon pour la dernière fonction.

Le même probleme se produit lorsque  $f(x)=x$  pour la première méthode.

## Source :

[https://github.com/Alexsaphir/TP\\_EDP\\_Python](https://github.com/Alexsaphir/TP_EDP_Python)

```

1  # -*- coding: utf-8 -*-
2  from numpy import *
3  from numpy.linalg import *
4  from numpy.random import *
5  from matplotlib.pyplot import *
6
7  print('Valeur de pi :', pi)
8  print(finfo(float).eps)
9
10 #Create a Vector
11 X1 = array([1, 2, 3, 4, 5])
12 print('Simple vecteur :', X1)
13
14 X2 = arange(0,1,.25)
15 print('Subdvision uniforme :', X2)
16
17 X3 = linspace(0,10,3)
18 print('Vecteur n-pts :', X3)
19
20 X4 = zeros(5)
21 print('Vecteur zeros :', X4)
22
23 #Matrice
24 M1=array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
25 print('Simple matrice', M1)
26
27 M2=zeros((2,3))
28 print('Matrice zeros', M2)
29
30 #Exercice 1
31 A=-1*(diag(ones(9),1)+diag(ones(9),-1))+2.*eye(10);
32 A[9,9] = 1
33 print(A)
34
35 #Fonctions
36 def fonc(x, y, m):
37     z = x**2 + y**2
38     t = x - y + m
39     return z, t
40
41 #Test
42 if(1<2) :
43     print('1<2')
44 elif (1<3) :
45     print('1<3')
46 else :
47     print('Aussi non !')
48
49 #####
50 ## Exercice 3 ##
51 #####
52 #####
53 def f(x, m) :
54     if (m == 0) :
55         y = zeros(size(x))
56     elif (m == 1) :
57         y = ones(size(x))
58     elif (m == 2) :
59         y = x
60     elif (m == 3) :
61         y = x**2
62     elif (m == 4) :
63         y = 4.*pi*pi*sin(2.*pi*x)
64     else :
65         print('valeur de m indéfinie')
66     return y
67
68 def solex(x, ug, m) :
69     if (m == 0):
70         y = ug*ones(size(x))
71     elif (m == 1) :
72         y = -0.5*x**2+x+ug

```

```

73     elif (m == 2) :
74         y = -(1/6)*x**3+(1/2)*x+ug
75     elif (m == 3) :
76         y = -(1/12)*x**4+(1/3)*x+ug
77     elif (m == 4) :
78         y = sin(2*pi*x)-2*pi*x+ug
79     else :
80         print('valeur de m indéfinie')
81     return y
82
83
84 #####
85 #####
86 ## Exercice 2 ##
87 #####
88 #####
89
90
91 print('Pour le second membre, choix de la fonction f')
92 print('Pour m=0 : f=0')
93 print('Pour m=1 : f=1')
94 print('Pour m=2 : f=x')
95 print('Pour m=3 : f=x^2')
96 print('Pour m=4 : f=4*pi^2*sin(2*pi*x)')
97 m = int(input("Choix de m = "))
98
99 print('Choix de la condition a gauche')
100 ug = float(input('ug = u(0) ='))
101
102 print("Methode pour l'approximation de u'(1) : ")
103 print("1- decentre d'ordre 1")
104 print("2- centre d'ordre 2")
105 meth = int(input('Choix = '))
106
107 print('Choix du nombre Ns de points interieurs du maillage')
108 Ns = int(input('Ns = '))
109
110 # Maillage
111 h=1./(Ns+1)
112 X=linspace(0, 1., Ns+2)
113 Xh=linspace(h,1.,Ns+1)
114
115 # Matrice du systeme lineaire :
116 A=-1*(diag(ones(Ns),1)+diag(ones(Ns),-1))+2.*eye(Ns+1);
117 A[Ns, Ns] = 1
118 A=1./h/h*A
119 #Conditionnement de la matrice
120 cond_A=cond(A)
121 print('Conditionnement de la matrice A :', cond_A)
122
123
124
125 # Second membre
126 # b = ... (plus loin, exercice 3)
127 b = f(Xh, m)
128 b[0] = b[0] + (Ns + 1)**2*ug
129
130
131 # Transformation de b[Ns] pour prendre en compte u'(1) = 0 (cf TD)
132 if (meth == 2):
133     b[Ns] = b[Ns]/2
134
135
136 # Resolution du syteme lineaire
137 Uh = solve(A, b) # ceci calcule Uh solution du systeme AU=b
138
139 # Calcul de la solution exacte aux points d'approximation
140
141 Uex = solex(Xh, ug, m)
142
143 # Calcul de l'erreur en norme infini
144 Uerr = abs(Uex - Uh)

```



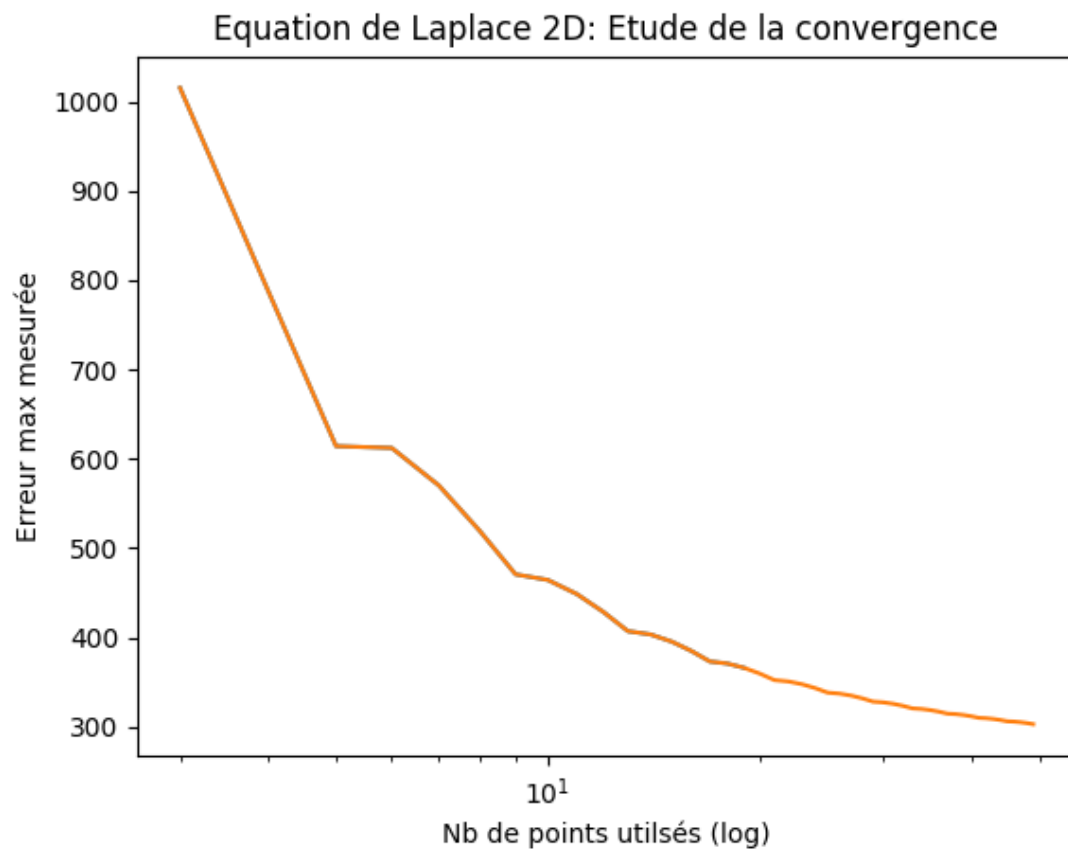
```

145 disp(max(Uerr))
146
147 #Graphes
148 Uh = concatenate([ug],Uh))
149 # on complete le vecteur solution avec la valeur ug en 0
150 # On trace le graphe de la fonction solex sur un maillage fin de 100 points
151 plot(linspace(0,1,100),solex(linspace(0,1,100), ug, m), label = 'sol ex')
152
153 # et le graphe de la solution approchée obtenue
154 plot(X, Uh, label = 'sol approchee')
155
156
157 plot(Xh, Uerr, label = 'Erreur')
158 # On ajoute un titre
159 title('d²u(x)/dx²=f(x)')
160
161 # On ajoute les labels sur les axes
162 xlabel('X')
163 ylabel('Y')
164
165 # Pour faire afficher les labels
166 legend()
167 show() # Pour afficher la figure
168

```

# Etude de la convergence du schéma pour l'équation de Laplace 2D

---



On peut remarquer une convergence d'ordre 2

Code Source : [https://github.com/Alexsaphir/TP\\_EDP\\_Python](https://github.com/Alexsaphir/TP_EDP_Python)

```

1  # -*- coding: utf-8 -*-
2
3  from numpy import * # importation du module numpy
4  from numpy.linalg import * # importation du module numpy.linalg
5  from numpy.random import *
6  from matplotlib.pyplot import *
7  from mpl_toolkits.mplot3d import Axes3D
8
9
10 Ns = 20
11
12 # Maillage
13
14 h = 1./(Ns + 1)
15 X = linspace(0,1,Ns+2)
16 Xh = X[1:Ns+1]
17
18 # Matrice du système linéaire
19
20 A = -1*(diag(ones(Ns*Ns-3),3) + diag(ones(Ns*Ns-3),-3))
21
22 B = 4*eye(Ns) -1*(diag(ones(Ns-1),1) + diag(ones(Ns-1),-1))
23
24 for i in arange(0,Ns):
25     A[Ns*i:Ns*i+Ns,Ns*i:Ns*i+Ns] = B
26
27 def Ud(x):
28     y = sin(2*pi*x)*sinh(2*pi)
29     return y
30
31 # Fonction définissant la solution exacte de l'équation
32
33 def solex(x, y):
34     z = sin(2*pi*x)*sinh(2*pi*y)
35     return z
36
37 # Second membre
38
39 b = zeros(Ns*Ns)
40 b[Ns*(Ns-1):Ns*Ns] = Ud(Xh)
41
42 # Resolution du systeme lineaire
43
44 A_inv = linalg.inv(A)
45 Uh = solve(A_inv, b)
46
47 #Mise en forme de la matrice Zh
48 Zh = array( 0*Ud(X))
49 for i in arange(0, Ns, 1):
50     newrow = Uh[ i*(Ns):i*Ns+Ns]
51     newrow = concatenate([[0], newrow, [0]])
52     Zh = vstack([newrow, Zh])
53 Zh = vstack([Ud(X), Zh])
54
55
56 #Calcul du maillage
57 coordX, coordY= np.meshgrid(X, flip(X,0))
58
59 #Calcul de la solution exacte sur le maillage
60 U = solex(coordX,coordY)
61
62 #Calcul de l'erreur
63 Err = amax(absolute(U-Zh))
64
65 fig = figure()
66 ax = Axes3D(fig, azimuth = 30, elev = 30)
67 ax.plot_surface(coordX, coordY, Zh, cmap = cm.jet)
68 #ax.plot_surface(coordX, coordY, U, cmap = cm.jet)
69 fig.show()
70

```