Benjamin Rogé / Edouard Poupa

Compte-rendu du TP1 Math-Info 2 : Optimisation pour l'ingénieur

Exercice 1 : Optimisation de la fonction de Rosenbrock

Etude théorique

(1)

Fonction de Rosenbrock : $f(x,y) = (x-1)^2 + 10.(x^2-y)^2$

Afin de trouver les points critiques de la fonction de Rosenbrock, tout d'abord, on va calculer les dérivées partielles par rapport à x et par rapport à y (nos deux variables).

$$\frac{\partial f}{\partial x}(x,y) = 2x - 2 + 40x^3 - 40x.y$$
$$\frac{\partial f}{\partial y}(x,y) = -20x^2 + 20y$$

Trouver les points critiques de f revient à résoudre le système : $\begin{cases} 2x - 2 + 40x^3 - 40x.y = 0 \\ -20x^2 + 20y = 0 \end{cases}$

Après résolution on obtient donc : $\begin{cases} x = 1 \\ y = 1 \end{cases}$

Par conséquent, f admet un unique point critique : (1,1).

Or, les extrema locaux d'une fonction différentiable ne pouvant être atteints qu'en un point critique, il suffit d'étudier si le point (1,1) est un extremum local.

Or : $(x-1)^2 \geq 0$ et $(x^2-y)^2 \geq 0$

Et : $f(1,1) = 0$

Donc : $\boldsymbol{f(x,y) = (x-1)^2 + 10(x^2-y)^2 \geq 0 = f(1,1)}$

Ainsi, le point (1,1) est un extremum local et même global de f.

(2)

$$Hessf(x,y) = \begin{pmatrix} 2 + 120x^2 - 40y & -40x \\ -40x & 20 \end{pmatrix}$$

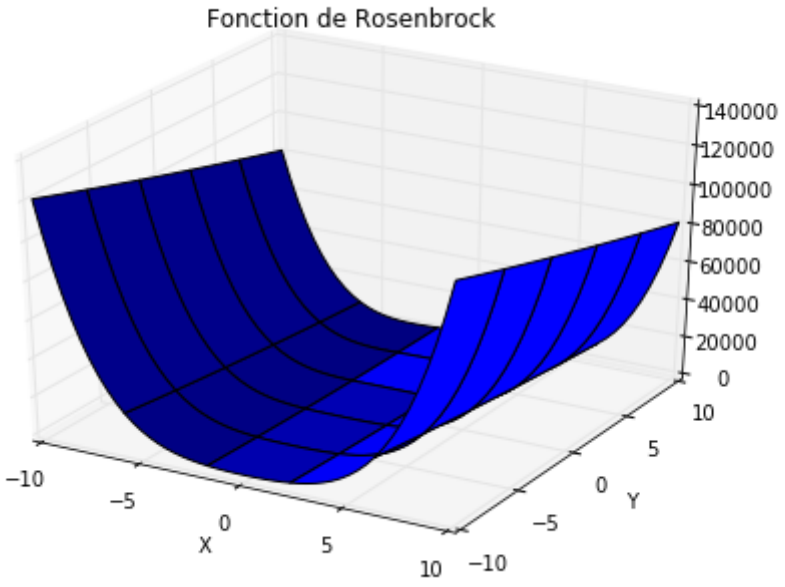
Par conséquent on obtient : $\boldsymbol{Hessf(x*, y*) = Hessf(1,1) = \begin{pmatrix} 82 & -40 \\ -40 & 20 \end{pmatrix}}$

Etude numérique

(1)

```
 8 import math as mp
 9 import numpy as np
10 import matplotlib.pyplot as plt
11 from mpl_toolkits.mplot3d import Axes3D
12
13 #Etude numerique
14
15 #Question 1
16
17 def Rosenbrock(X):
18     return ((X[0]-1)**2 + 10*(X[0]**2-X[1])**2)
19
20 #Creation du graphique en 3D de la fonction de Rosenbrock
21
22 def graph_Rosenbrock():
23
24     ax = Axes3D(plt.figure())
25     X = np.linspace(-10,10,50)
26     Y = np.linspace(-10,10,50)
27     X, Y = np.meshgrid(X, Y)
28     Z = Rosenbrock([X,Y])
29     ax.plot_surface(X, Y, Z)
30     plt.title("Fonction de Rosenbrock")
31     plt.xlabel("X")
32     plt.ylabel("Y")
33     plt.legend()
34     plt.show()
```

Après avoir écrit ce code, on lance la fonction *graph_Rosenbrock()* et on obtient le résultat suivant :



(2)

On passe maintenant à la comparaison de deux méthodes de minimisation : la méthode du gradient à pas fixe puis celle du gradient à pas optimal.
Ces deux méthodes seront toutes deux initialisées avec le point (0,1).

(a)

Tout d'abord, la méthode du gradient à pas fixe.

On introduit une fonction nommée *gradient_f* qui renvoie le gradient de la fonction de Rosenbrock en un point donné ainsi que la fonction *norme* qui renvoie tout simplement la norme d'un vecteur.

Dans la fonction *PasFixe*, nous utilisons comme l'énoncé le suggère un double critère d'arrêt : à la fois lorsque la norme du gradient est inférieure à une tolérance donnée que nous avons nommée epsilon dans le programme et que nous avons fixé à 0.001. Ainsi qu'une deuxième condition d'arrêt en ce qui concerne le nombre total d'itération : cela évite les boucles infinies et nous donne une information sur l'efficience du programme.

```
#Question 2
#Question 2a) : Methode du gradient a pas fixe


def gradient_f(X):
    return([2*X[0]-2+40*X[0]**3-40*X[0]*X[1],-20*X[0]**2+20*X[1]])

def norme(u):
    x=u[0]
    y=u[1]
    return mp.sqrt(x**2+y**2)

# l enonce propose la fonction PasFixe avec seulement pour argument x (l initialisation) et beta le pas
#Par consequent kmax et epsilon (l erreur acceptee) seront donnes dans le programme


def PasFixe(x,beta):
    k=0
    x= [x]
    kmax=10000 #tres eleve juste pour ne pas bloquer la resolution mais eviter une boucle infinie ou tres longue
    epsilon=10**(-3)

    while norme(gradient_f(x[k]))>=epsilon and k<=kmax:
        a=x[k][0]+beta*(-gradient_f(x[k])[0])
        b=x[k][1]+beta*(-gradient_f(x[k])[1])
        A=(a,b)
        x.append(A)
        k+=1

    return(k,x[-1],norme(gradient_f(x[-1])),x)

PF = PasFixe([0,1],0.01)
print ("PasFixe : [x,y] = ",PF[0],' et k = ',PF[1], ' et Epsilon = ',PF[2])

```

Lorsqu'on exécute le programme avec la condition initiale donnée dans l'énoncé : le point (0,1) ainsi qu'un pas qu'on choisit à 0.01 on obtient alors :

```
PasFixe : [x,y] =  1407  et k =  (0.9988852403538895, 0.9977269260563946)  et
Epsilon =  0.0009979926086203 66
```

(b)

Nous sommes maintenant amenés à programmer la méthode du gradient à pas optimal. Pour cela, nous allons introduire une fonction *Omega* qui représente la fonction que l'on doit optimiser et qui fait référence au calcul des $x^{(k)}$du cours.

```
#Question 2 b : Methode du gradient a pas optimal

#Fonction a minimiser

def Omega (X,alpha):
    return ([X[0]-alpha*gradient_f(X)[0],X[1]-alpha*gradient_f(X)[1]])

```

Ensuite, nous programmons la fonction *SectionDoree* qui utilise la méthode de la section dorée afin de déterminer ici le minimum de la fonction *Omega* et qui renvoie donc le pas optimal. Le pas optimal sera ensuite rentré dans la fonction *PasOptimal* comme le suggère l'énoncé (cela correspond au α de l'énoncé).

```
78
79 def SectionDoree(X0,a,b,tolerance): #Recherche du pas optimal
80     tau = (1+np.sqrt(5))/2
81     it = 0
82     err = b-a
83     while np.abs(err)>tolerance :
84         aprime = a + (b-a)/(tau*tau)
85         bprime = a + (b-a)/tau
86         c , d = Omega(X0,aprime) , Omega(X0,bprime)
87         if c > d :
88             a = aprime
89         elif c < d :
90             b = bprime
91         else :
92             a , b = aprime , bprime
93         err = b - a
94         it = it + 1
95     return (a + b)/2
96
97 alpha = SectionDoree([0,1],0,1,0.01)
```

Pour une initialisation avec le point (0,1) on trouve alors un pas optimal :
alpha = 0.0040653093778916733

```
 99 def PasOptimal (X0,alpha):
100
101     x=X0[0]
102     y=X0[1]
103     listeXY = [[x,y]]
104     k=0
105     kmax=10000
106
107     gradx,grady = gradient_f([x,y])
108
109     while (norme(gradient_f([x,y]))>0.01) and (k<=kmax):
110
111         X = [x,y]
112         x,y = Omega(X,alpha)
113         k+=1
114         gradx,grady = gradient_f([x,y])
115         listeXY.append([x,y])
116     X = [x,y]
117
118     a=X0[0]-alpha*gradient_f(X0)[0]
119     b= X0[1]-alpha*gradient_f(X0)[1]
120
121     Resp=Rosenbrock([a,b])
122
123
124     return (X,k,norme(gradient_f([x,y])),listeXY,Resp)
125
126
127 PO = PasOptimal([0,1],alpha)
128 print ("PasOptimal : [x,y] = ",PO[0],' et k = ',PO[1], ' et Epsilon = ',PO[2])
129
```

Lorsqu'on applique la fonction du *PasOptimal* à X0= (0,1) et

alpha = 0.0040653093778916733 on obtient alors :

```
PasOptimal : [x,y] =  [0.98893432972137441, 0.97754369670930741]  et k =  2035  et Epsilon =
0.009986069012976969
```

Si on renvoie ce que demande l'énoncé c'est-à-dire : $f(x - \alpha\Delta f(x))$ cela correspond dans notre programme à : *PasOptimal([0,1],alpha)[4]* qui est égal à : 9.4225734822192493

(c)

On en vient alors à comparer les deux méthodes d'optimisation.

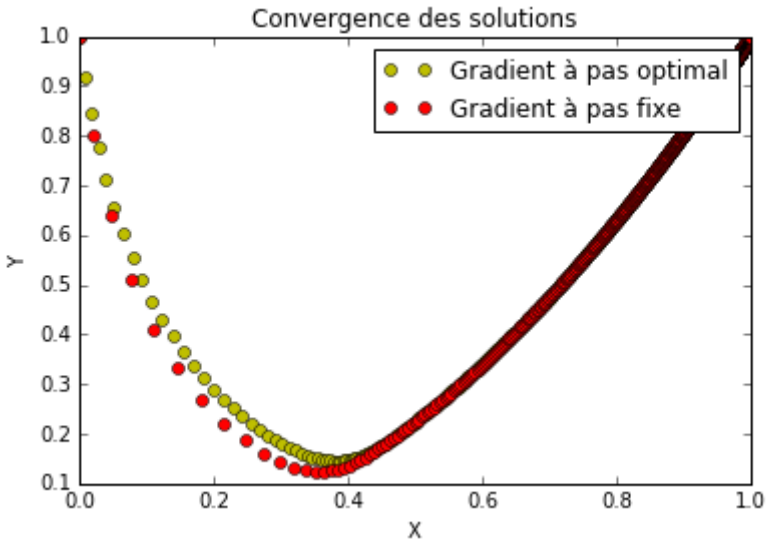
(i)

On affiche sur une même figure les itérés obtenus par les deux méthodes. Pour cela on réalise le code suivant :

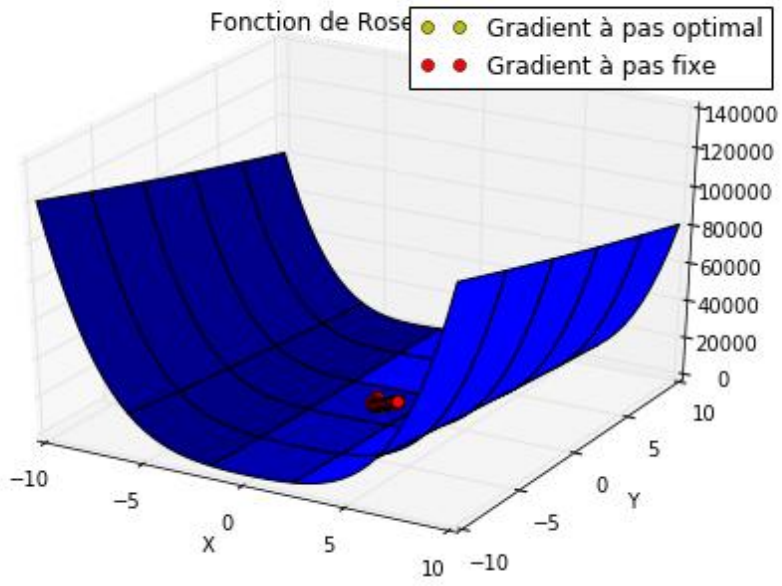
```
131 #Question 2c) Comparaison des methodes
132 #i. Affichage des courbes des iteres pour la methode du pas fixe et du pas optimal
133
134 def CreerlistepointsX(liste):
135     A = []
136     for i in range (len(liste)):
137         A.append(liste[i][0])
138
139     return (A)
140
141 def CreerlistepointsY(liste):
142     B= []
143     for i in range (len(liste)):
144         B.append(liste[i][1])
145     return (B)
146
147
148 A = CreerlistepointsX(PasFixe([0,1],0.01)[3]) # Création de la liste contenant les valeurs de X
149 B = CreerlistepointsY(PasFixe([0,1],0.01)[3]) # Création de la liste contenant les valeurs de Y
150
151
152 C = CreerlistepointsX(PasOptimal([0,1],alpha)[3]) # Création de la liste contenant les valeurs de X
153 D = CreerlistepointsY(PasOptimal([0,1],alpha)[3]) # Création de la liste contenant les valeurs de Y
154
```

```
157
158 plt.plot(C,D,'o',color='y', label = "Gradient à pas optimal")
159 plt.plot(A,B,'o',color='r', label = "Gradient à pas fixe")
160 plt.legend()
161 plt.title("Convergence des solutions")
162 plt.xlabel("X")
163 plt.ylabel("Y")
164
165 ax = Axes3D(plt.figure())
166 X = np.linspace(-10,10,50)
167 Y = np.linspace(-10,10,50)
168 X, Y = np.meshgrid(X, Y)
169 Z = Rosenbrock([X,Y])
170 ax.plot_surface(X, Y, Z)
171 plt.title("Fonction de Rosenbrock")
172 plt.xlabel("X")
173 plt.ylabel("Y")
174 ax.plot(C,D,'o',color='y', label = "Gradient à pas optimal")
175 ax.plot(A,B,'o',color='r', label = "Gradient à pas fixe")
176 plt.legend()
177 plt.show()  # affichage de la nappe
178
```

On obtient alors :



Et :



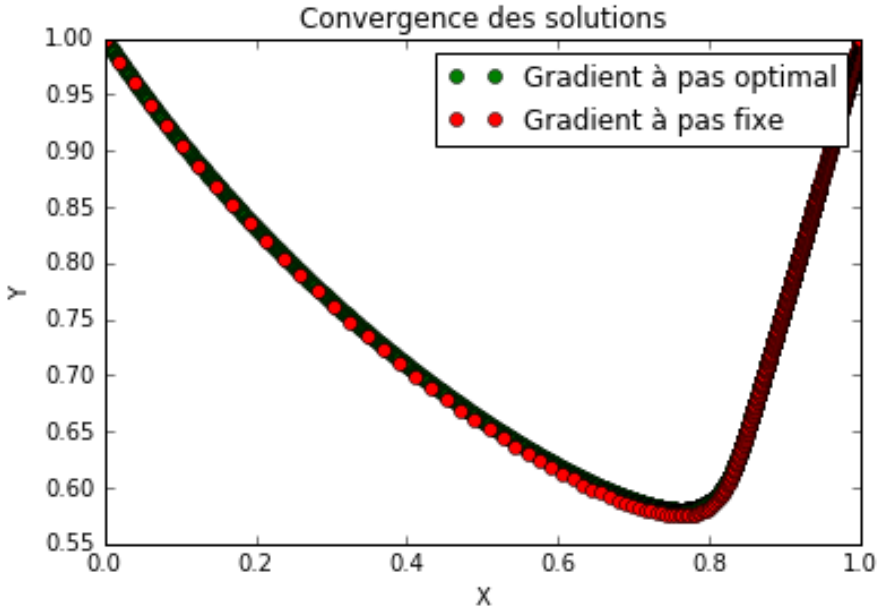
(ii)

On remarque alors que pour atteindre une précision aussi fine que celle obtenue par la méthode du gradient à pas fixe, il faut beaucoup plus d'itérations avec la méthode du gradient à pas optimal. Ce phénomène peut s'expliquer par le fait que nous rentrons le pas optimal comme argument de la fonction *PasOptimal*, ce dernier est donc fixé par rapport au point initial (ici (0,1)). Il faudrait sûrement, pour éviter cela recalculer le pas optimal à chaque itération.

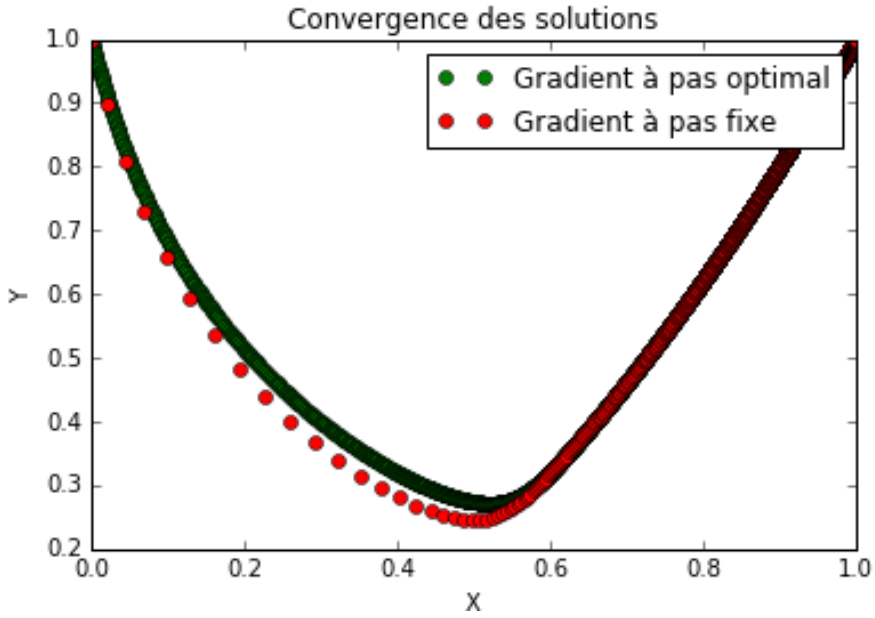
(3)

Quand le paramètre P augmente, le minimum du graphe de convergence diminue.

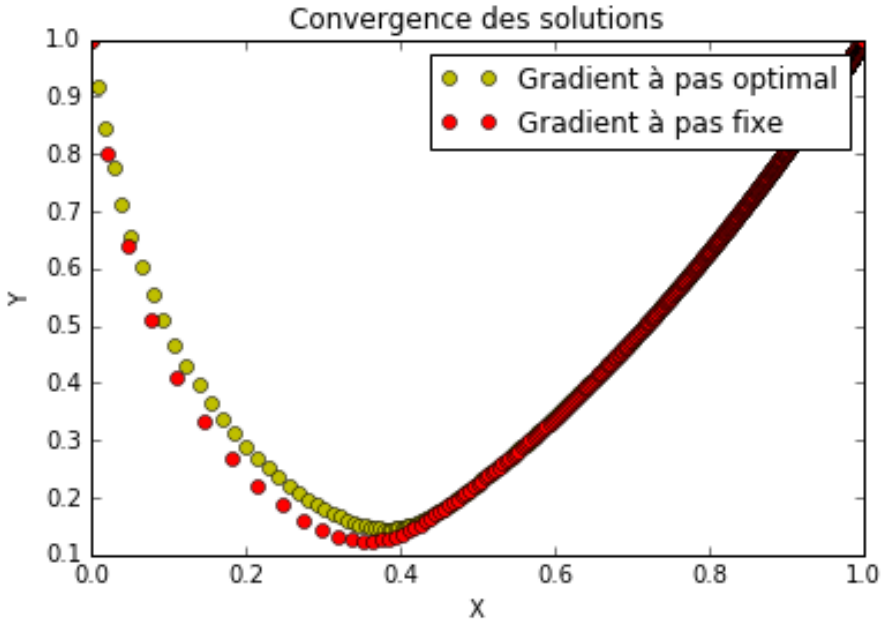
En effet, pour k=1 on obtient :



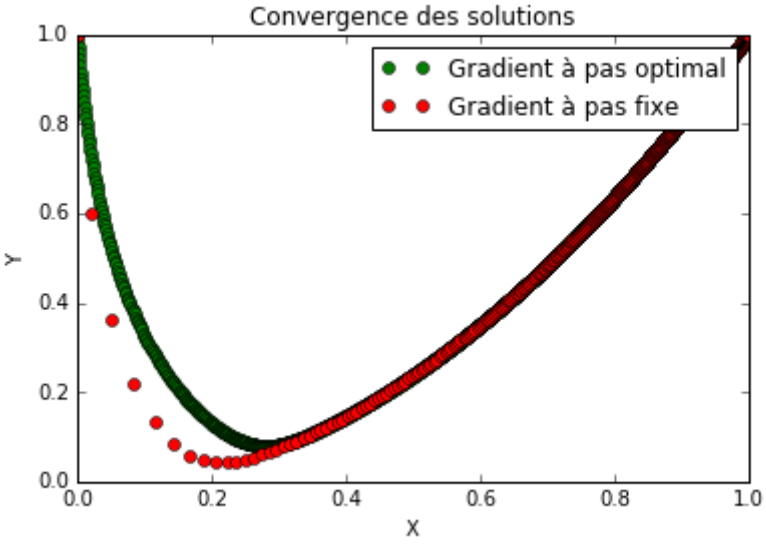
Pour k=5 :



Pour k=10 :



Et pour k=20 :

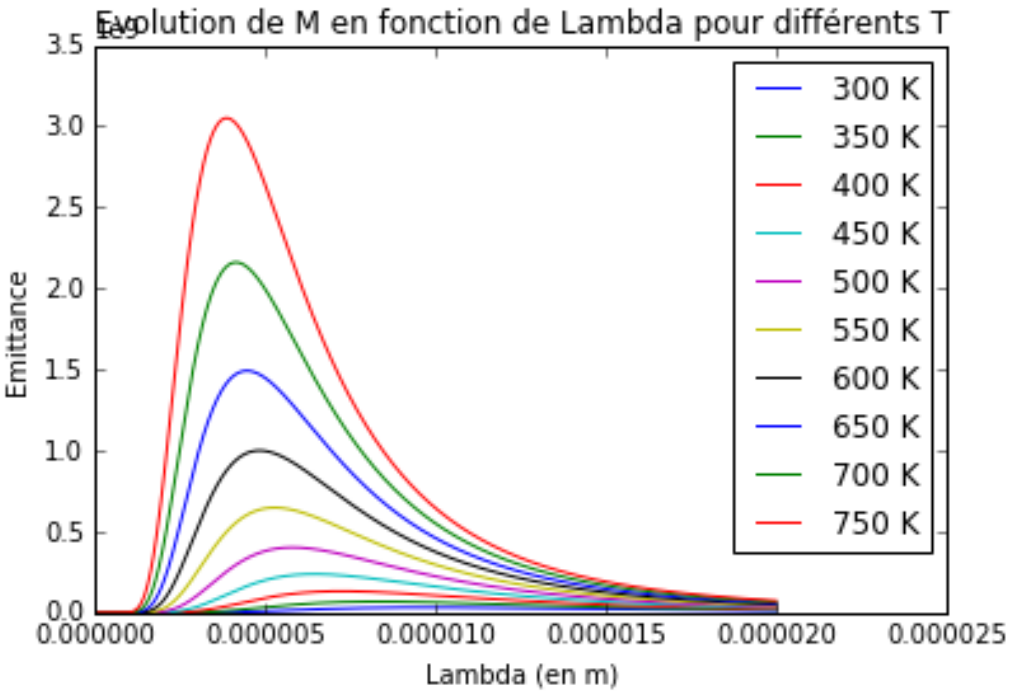


Exercice 2 : Emittance

(1)

```python
#Exercice 2

#Question 1

def emittance(L,T) :   # On définie la définition de la fonction émittance
    C = 2.997*10**8
    h = 6.625 * 10**-34
    k = 1.38*10**-23
    n = 1
    return (((2 * np.pi* h * C**2)/(n**2 * L**5))/(np.exp((h*C)/(n*k*T*L))-1))

ListeLambda = np.linspace(10**-7,2*10**-5,1000) #Permet de créer une liste de lambda de 10**-7 à 2*10**-5

def Liste_emittance(Liste,T): #Création des listes de valeur de la fonction emittance pour différents lambda
    liste = np.zeros(len(Liste))
    for i in range (len(liste)):
        liste[i]=emittance(Liste[i],T)
    return (liste)

def TracerCourbes(Tinit,Tfin): #Permet de tracer les courbes pour tous T
    T = Tinit
    while T !=Tfin:
        plt.plot(ListeLambda,Liste_emittance(ListeLambda,T),label = str(T) + " K" )
        T+=50
    plt.legend()
    plt.xlabel("Lambda (en m)")
    plt.ylabel("Emittance")
    plt.title("Evolution de M en fonction de Lambda pour différents T")
    plt.show()



TracerCourbes(300,800) #Lance le tracer des courbes pour différents T

```

On obtient alors :



(2)

Si on utilise la méthode de la section dorée, il faut connaitre un intervalle sur lequel l'émittance admet un unique maximum. D'après l'affichage de l'émittance pour différentes températures, on prendra comme intervalle [10**-7,2*10**-5]

La méthode de la section dorée cherche le minimum d'une fonction or dans notre cas, nous recherchons le maximum.
Par conséquent, pour utiliser la section dorée nous avons changé le signe de l'inégalité.

On a alors le code suivant :

```
231 def SectionDoree_Max(a,b,tolerance,T): #Section dorée pour la recherche d'un maximum
232     tau = (1+np.sqrt(5))/2
233     c = b-(b-a)/tau
234     d = a+(b-a)/tau
235     k=0
236     kmax = 10000
237 
238     while np.abs(c-d)>tolerance and k<=kmax:
239         if emittance(c,T)>emittance(d,T):
240             b=d
241         else:
242             a=c
243 
244         c = b-(b-a)/tau
245         d = a+(b-a)/tau
246     return ((b+a)/2)
247 
248 def RechercheMax(T,tolerance): #Cette fonction cherche l'abscisse correspondant au maximum de la fonction M pour un T donné.
249     intervalle = [10**-7,2*10**-5]
250     min = SectionDoree_Max(intervalle[0],intervalle[1],tolerance,T)
251     return (min)
252 
253 print ("Lambda* (T = 300 K) = ",RechercheMax(300,10**-8))
254 print ("Lambda* (T = 400 K) = ",RechercheMax(400,10**-8))
255 print ("Lambda* (T = 450 K) = ",RechercheMax(450,10**-8))
256 print ("Lambda* (T = 500 K) = ",RechercheMax(500,10**-8))
257 print ("Lambda* (T = 600 K) = ",RechercheMax(600,10**-8))
258 print ("Lambda* (T = 800 K) = ",RechercheMax(800,10**-8))
```

Et on obtient les résultats suivants en ce qui concerne les longueurs d'onde maximisant l'émittance pour quelques températures données :

Avec lambda en mètre.

```
Lambda* (T = 300 K) =  9.65279643439e-06
Lambda* (T = 400 K) =  7.23482350248e-06
Lambda* (T = 450 K) =  6.43484370297e-06
Lambda* (T = 500 K) =  5.80223588498e-06
Lambda* (T = 600 K) =  4.84045677223e-06
Lambda* (T = 800 K) =  3.63147030627e-06
```