Projet - Optimisation

Minimisation de coût

CECILE LIN / MINH ANH NGUYEN

Année 2021 - 2022

Contents

1 Introduction **3**

2 Résoudre le problème **3**

2.1 Trouver la répartition de la production permettant de minimiser le coût avec la contrainte. 4

2.2 Trouver la repartition de la production permettant de minimiser le coût sans cette contrainte, si elle existe. La solution obtenue est-elle realisable ? 7

2.3 En exploitant les resultats obtenus aux point précédents, que suggériez-vous au glacier ? 8

3 Ecrire un programme Python **8**

3.1 Le graphe de la surface associé a la fonction coût c et ses lignes de niveau 8

3.2 Calcul en language formel son gradient ∇c et sa matrice hessienne $\mathbb{H}c$ 12

3.3 Trouve et affiche les extrema de la fonction cout sans contrainte 12

3.4 Les lignes de niveaux et l'ensemble de contrainte(s) 14

3.5 Trouve et affiche les extrema sous contrainte de la fonction coût 15

1 Introduction

Un glacier produit deux types de glace, la glace A et la glace B. Le coût total pour produire x kilogrammes de glace de type A et y kilogrammes de glace de type B pour une semaine est

$$c(x, y) = 600 + 200x + 200y - 2x^2y$$

Sachant que le coût hebdomadaire en produisant les deux types de glace en une quantité inférieure ou égale à 300 kilogrammes.

Peut-il minimiser le coût hebdomadaire en produisant les deux types de glace en une quantité inférieure ou égale à 300 kilogrammes? Pour répondre à cette question on analyse d'abord le cas d'une utilisation à pleine capacité du laboratoire et on étudie ensuite si cette stratégie est la meilleure possible.

2 Résoudre le problème

Nous allons d'abord traduire le problème sous forme mathématiques.

Tout d'abord, on remarque qu'il existe deux contraintes implicites qui sont $x \geq 0$ et $y \geq 0$. Ainsi, nous avons choisi de définir la fonction de $R_+^2 \to R$ (et non pas de $R^2 \to R$) puisqu'un niveau de production est forcément positif ou nul.
On a alors $c : R_+^2 \to R$ et $\phi_1, \phi_2, \phi_3 : R_+^2 \to R$ définies par

$$c(x, y) = 600 + 200x + 200y - 2x^2y$$

On note qu'on a donc un ensemble de 3 contraintes:

$$\phi_1 = x + y - 300 \leq 0$$

$$\phi_2 = -x \leq 0$$

$$\phi_3 = -y \leq 0$$

on note

$$\phi(x, y) = (\phi_1(x, y), \phi_2(x, y), \phi_3(x, y))$$

Ce sont deux fonctions polynomiales qui sont de classe C^∞.
On cherche l'infimum de c sur l'ensemble des contraintes K avec

$$K = \{(x, y) \in R_+^2, \phi(x, y) \leq 0\}$$

L'ensemble K est compact puisque:
• fermé car c'est la contre-image d'intervalle fermé $]-\infty; 0]$ par une fonction continue
• borné par la boule centré en 0 et de rayon 500
Sur l'ensemble des contraintes K, la fonction c, continue, admet donc un point de maximum et un point de miminum.

2.1 Trouver la répartition de la production permettant de minimiser le coût avec la contrainte.

Nous allons trouver la répartition de la production permettant de minimiser le coût hebdomadaire en prenant en compte l'ensemble des contraintes K.

Commençons par les deux contraintes ϕ_2 et ϕ_3:

$$\phi_2(x,y) = 0 \tag{1}$$

• Si la contrainte ϕ_2 est active, i.e. si x = 0 alors $c(0,y) = 600 + 200y$. On remarque que pour tout $y \in R_+^2$, $c(0,y) \geq 600$ et que $c(0,0) = 600$ donc le point de minimum dans ce cas est $(0,0)$.
Comme $c(0,y) = 600 + 200y$ est une fonction croissante de y, si on ajoute ϕ_1 active, dans ce cas le maximum de la fonction est atteint en (0,300).

$$\phi_3(x,y) = 0 \tag{2}$$

• Si la contrainte ϕ_3 est active, i.e. si y = 0 alors $c(x,0) = 600 + 200x$. On remarque que pour tout $x \in R_+^2$, $c(x,0) \geq 600$ et que $c(0,0) = 600$ donc le point de minimum dans ce cas est de nouveau $(0,0)$.
Comme $c(x,0) = 600 + 200x$ est une fonction croissante de x, si on ajoute ϕ_1 active, dans ce cas le maximum de la fonction est atteint en (300,0).

Pour les cas suivant, calculons les gradients de c et de ϕ, et la matrice hessienne de la fonction de coût c. On a alors:

$$\nabla c(x,y) = \begin{bmatrix} \dfrac{\partial c}{\partial x}(x,y) \\ \\ \dfrac{\partial c}{\partial y}(x,y) \end{bmatrix} = \begin{bmatrix} 200 - 4xy \\ \\ 200 - 2x^2 \end{bmatrix}$$

$$\mathscr{H}c(10,5) = \begin{bmatrix} -4y & -4x \\ -4x & 0 \end{bmatrix}$$

$$\nabla \phi_1(x,y) = \begin{bmatrix} \dfrac{\partial \phi_1}{\partial x}(x,y) \\ \\ \dfrac{\partial \phi_1}{\partial y}(x,y) \end{bmatrix} = \begin{bmatrix} 1 \\ \\ 1 \end{bmatrix}$$

On distingue deux situations:

• Si $(x^*,y^*) \in K^o = \{(x,y) \in R_+^2, \phi(x,y) < 0\}$, alors l'ensemble des contraintes est inactive et on résout

$$\begin{cases} \nabla c(x,y) = 0 \\ \phi(x,y) < 0 \end{cases} \tag{3}$$

On a donc le système

$$\begin{cases} 200 - 4xy = 0 \\ 200 - 2x^2 = 0 \\ x + y < 300 \\ x > 0 \\ y > 0 \end{cases}$$

soit

$$\begin{cases} 2x^2 - 4xy = 0 \\ x^2 = 100 \\ x + y < 300 \\ x > 0 \\ y > 0 \end{cases}$$

En prenant en compte que $x > 0$, on a:

$$\begin{cases} x = 10 \\ 200 - 40y = 0 \\ x + y < 300 \end{cases} \quad \text{donc} \quad \begin{cases} x = 10 \\ y = 5 \\ x + y < 300 \end{cases}$$

On obtient (x^*, y^*) = (10, 5) qui satisfait aussi l'ensemble des contraintes K.

• Si $(x^*, y^*) \in \{(x, y) \in R_+^2, \phi_1(x, y) = 0\}$, i.e. avec la contrainte ϕ_1 active. $\nabla\phi_1(x, y) \neq 0$, on peut donc appliquer le théorème de Lagrange. Dans ce cas, il existe $\lambda \in$ R tel que

$$\begin{cases} \nabla c(x, y) = \lambda \nabla \phi_1(x, y) \\ \phi_1(x, y) = 0 \\ \phi_2(x, y) < 0 \\ \phi_3(x, y) < 0 \end{cases}$$

On peut réécrire cette condition sous la forme du système suivant:

$$\begin{cases} 200 - 4xy = \lambda \quad \text{(L1)} \\ 200 - 2x^2 = \lambda \quad \text{(L2)} \\ x + y = 300 \\ x > 0 \\ y > 0 \end{cases}$$

Par (L1) - (L2), on obtient:

$$\begin{cases} 2x^2 - 4xy = 0 \\ y = 300 - x \\ x > 0 \\ y > 0 \end{cases}$$

équivalent aux systèmes

$$\begin{cases} 2x^2 - 4(300 - x)x = 0 \\ y = 300 - x \\ x > 0 \\ y > 0 \end{cases}$$

$$\begin{cases} x[2x - 4(300 - x)] = 0 \\ y = 300 - x \\ x > 0 \\ y > 0 \end{cases}$$

$$\begin{cases} x(6x - 1200) = 0 \quad \text{(A)} \\ y = 300 - x \\ x > 0 \\ y > 0 \end{cases}$$

En résolvant l'équation (A), on obtient: $\begin{cases} \text{x = 0} \\ \text{y = 300} \\ \lambda = 200 > 0 \end{cases}$ ou $\begin{cases} \text{x = 200} \\ \text{y = 100} \\ \lambda = -79600 < 0 \end{cases}$

Comme on doit avoir $x > 0$ alors on ne retient que la solution $(200, 100)$. On obtient alors $(x^*, y^*) = (200, 100)$ et $\lambda < 0$. Il s'agit donc d'un point de minimum.

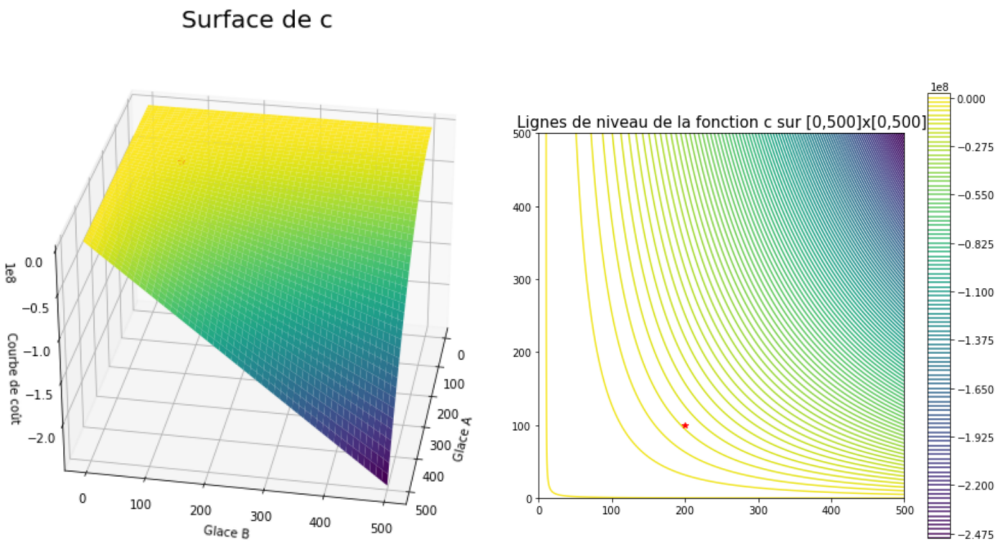
Dans les cas:
• Si les contraintes ϕ_2 et ϕ_3 sont toutes deux actives on a $x = 0$ et $y = 0$ donc (0,0) est le point cherché, on obtient le point de minimum des cas 1 et 2.
• Si les contraintes ϕ_1 et ϕ_3 sont toutes deux actives, on obtient le point de maximum du cas 2.
• Si les contraintes ϕ_1 et ϕ_2 sont toutes deux actives, on obtient le point de maximum du cas 1.
On retrouve donc les points précédémment trouvés.

Or on a
• c(0,0) = 600
• c(10,5) = 2 600
• c(0,300) = 60 600

• c(300,0) = 60 600
• c(200,100) = -7 939 400.

Ainsi $\underset{K}{\operatorname{argmin}}\, c(x,y) = (200,100)$

Sur les graphes suivant on peut voir le point de minimum cherché.



En conclusion, en prenant en compte la contrainte K, si le glacier souhaite minimiser son coût de production hebdomadaire, il devra produire 200 kilogrammes de glace de type A et 100 kilogrammes de glace de type B chaque semaine.

2.2 Trouver la repartition de la production permettant de minimiser le coût sans cette contrainte, si elle existe. La solution obtenue est-elle realisable ?

On va maintenant observer le cas d'une production sans contrainte.

Pour optimiser la fonction de coût c, il faut trouver les éventuelles points critiques de la fonction, i.e. résoudre l'équation suivante: $\nabla c(x,y) = 0$.
On a

$$\begin{cases} 200 - 4xy = 0 \\ 200 - 2x^2 = 0 \end{cases}$$

On remarque qu'on retrouve le cas précédent, avec contrainte, où les contraintes sont inactives 3. C'est pourquoi, on a de nouveau:

$$\begin{cases} x = 10 \\ y = 5 \end{cases}$$

On obtient alors (10,5) comme unique point critique.

En utilisant le critère de la matrice Hessienne, on a:

$$\mathscr{H}c(10,5) = \begin{bmatrix} -20 & -40 \\ -40 & 0 \end{bmatrix}$$

On a $\det(\mathscr{H}c(10,5)) = -40^2 < 0$ et $\text{tr}(\mathscr{H}c(10,5)) = -20$. Donc dans R_+^2, (10,5) est point selle de la fonction c.

La solution (10,5) est tout à fait réalisable. Cependant elle ne minimise pas le coût.

À partir de ce résultat, on conclut que sans aucune contrainte, on ne peut pas obtenir une répartition de la production qui permettrait de minimiser le coût.

2.3 En exploitant les resultats obtenus aux point précédents, que suggériez-vous au glacier ?

Enfin, en comparant les deux cas et les resultats obtenus, on conseille donc le glacier de produire 200 kilogrammes de glace de type A et 100 kilogrammes de type B, afin de minimiser son coût de production.

3 Ecrire un programme Python

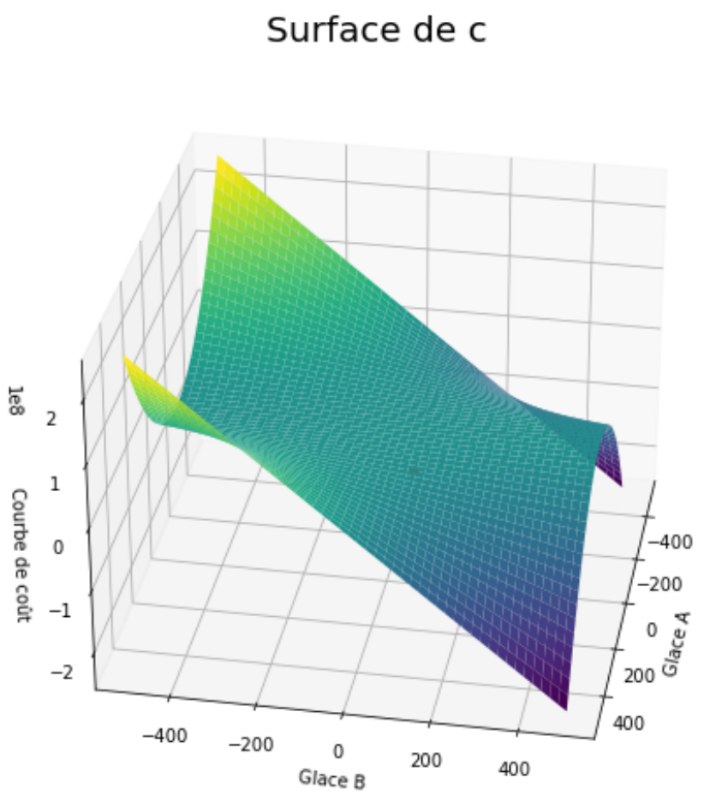
Nous avons utiliser différents packages pour réaliser ces visualisations.

[12]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from sympy import *

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import scipy
from scipy import optimize as opt
```

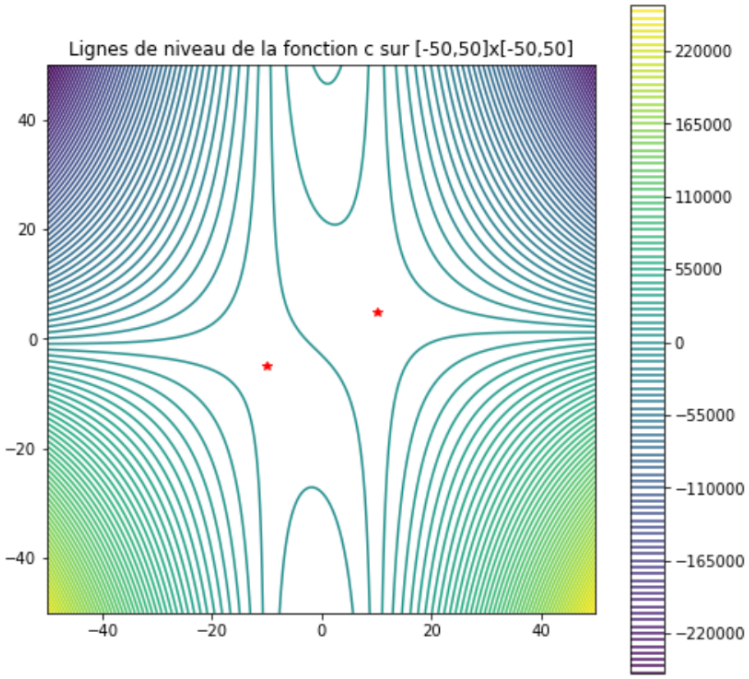
3.1 Le graphe de la surface associé a la fonction coût c et ses lignes de niveau

Comme c'est une fonction de coût, qui prend en entrée une quantité de production on va définir la fonction sur R_+^2 dans R.

Fonction c de $R^2 \to R$: on voit que la fonction est convexe puis concave. Elle est non bornée et n'admet donc pas de maximum ni de minimum sans contrainte.

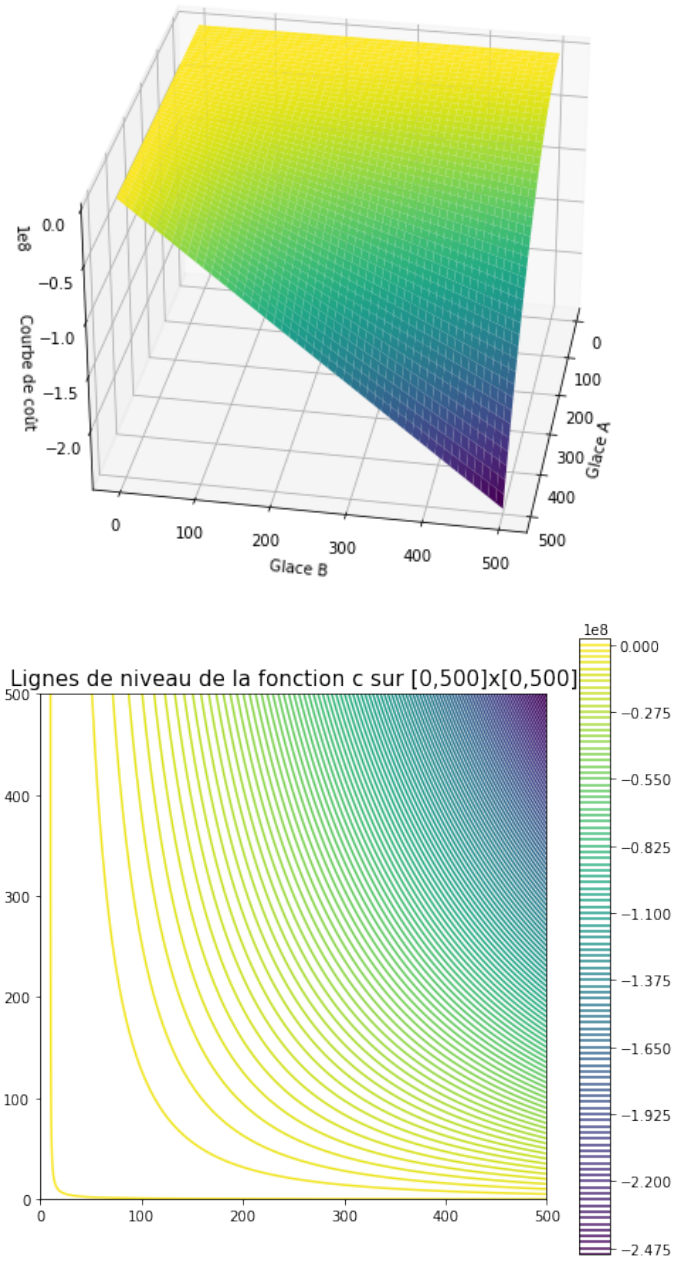


Ligne de niveau de c: on voit qu'il n'y a pas de point de minimum ni de maximum, uniquement des points selles.



[13]:
```
#définition de c(X)
#on définit X = (x,y) = (X[0],X[1]) comme un vecteur à deux coordonnées

def c(X):
    return (600 + 200*X[0] + 200*X[1] - 2*(X[0]**2)*X[1])

# on rappelle que x >= 0 et y >= 0
# on va coder la fonction sur [0,500]x[0,500]
x = np.linspace(0,500,1000)
y = np.linspace(0,500,1000)
X,Y = np.meshgrid(x,y)

Cout = c([X,Y])

# Affichage du graphe de c
fig = plt.figure(figsize = (8,8))
ax = plt.axes(projection = '3d')
ax.set_xlabel("Glace A")
ax.set_ylabel("Glace B")
ax.set_zlabel("Courbe de coût")
ax.view_init(30,10)
ax.plot_surface(X,Y,Cout,cmap = 'viridis')
#ax.plot_surface(X,Y,300-X,cmap = 'viridis')
#plt.plot(200,100,'r*')
#plt.plot(0,300,'g*')
#plt.plot(300,0,'b*')
plt.title("Surface de c",fontsize=20)
plt.show()

# Affichage des lignes de niveau de la fonction c
plt.figure(figsize = (8,8))
plt.contour(X,Y,Cout,100)
plt.colorbar()
plt.title("Lignes de niveau de la fonction c sur [0,500]x[0,500]", fontsize = 15)
plt.axis("square")
#plt.plot(200,100,'r*')
#plt.plot(0,300,'g*')
#plt.plot(300,0,'b*')
plt.show()


# On a utilisé ce code pour obtenir un graphe sur R² -> R même si nous avons␣
 ↪défini la fonction c sur R^2 -> R

# Affichage des lignes de niveau de la fonction c sur [-500,500]x[-500,500]
# plt.figure(figsize = (8,8))
# plt.contour(X,Y,Cout,100)
```

```
# plt.colorbar()
# plt.title("Lignes de niveau de la fonction c sur [-500,500]x[-500,500]",␣
 ↪fontsize = 10)
# plt.axis("square")
# plt.plot(200,100,'r*')
# plt.show()
```



3.2 Calcul en language formel son gradient ∇c et sa matrice hessienne $\mathbb{H}c$

[14]:
```
# nous sommes en dimension 2, donc on pose X=(x,y)
x, y = symbols('x y')
c = symbols('c', cls=Function)
X = Matrix([x,y])

c = Matrix([600 + 200*x + 200*y - 2*(x**2)*y])
#print ("c(x,y) ="); c

#calcul de la jacobienne et du gradient
#print ("La dérivée partielle de c en x est : "); diff(c,x)
#print ("La dérivée partielle de c en y est : "); diff(c,y)

#affichage de la jacobienne et du gradient
JacC = c.jacobian(X)
#print ("La jacobienne de c est : \n"); JacC
#print("Le gradient de c est donc : \n"); JacC.T

#calcul et affichage de la matrice hessienne
HessC = simplify(hessian(c,X))
#print ("La hessienne de c est : \n"); HessC
```

3.3 Trouve et affiche les extrema de la fonction cout sans contrainte

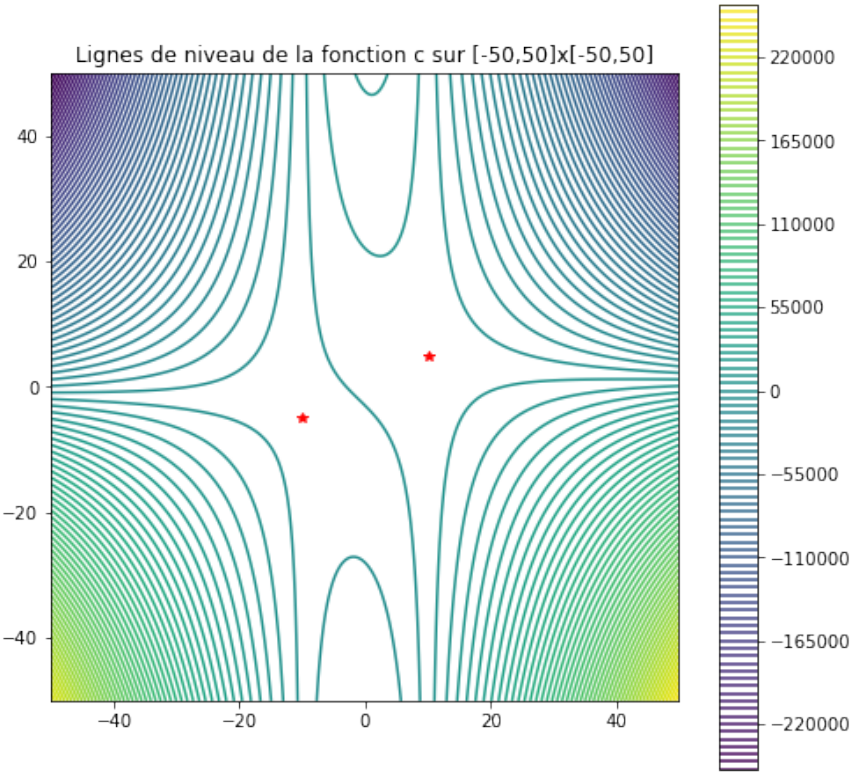
Sans la contrainte $x + y \leq 300$, on a remarqué avec les questions précédentes qu'il y a pas de point de minimum ou de maximum sans contrainte. On a donc préféré ne pas utiliser la fonction opt.minimize (puisque cela ne fonctionnerait pas) mais de trouver puis classifier les points critiques grâce au critère de la hessienne.

Nous utiliserons la fonction opt.minimize à la question suivante, avec la contrainte.

[15]:
```
def c(X):
    return (600 + 200*X[0] + 200*X[1] - 2*(X[0]**2)*X[1])

#calcul du point critique de c, sans contrainte:
pointC = solve(JacC, X)
# on trouve deux points critiques sur R^2, qui sont (-10,-5) et (10,5)
print('Les points critiques de c sur R^2 sont : {', pointC[0], ',',␣
 ↪pointC[1],'}')

#classification avec le critère de la Hessienne :
def classification(Hess):
    vp, VP = np.linalg.eig(Hess) # trouve valeur propre
    if(np.all(np.greater(vp,np.zeros(2)))): # teste si toutes les valeurs␣
 ↪propres sont strictement postives
        return('est un minimum local')
```

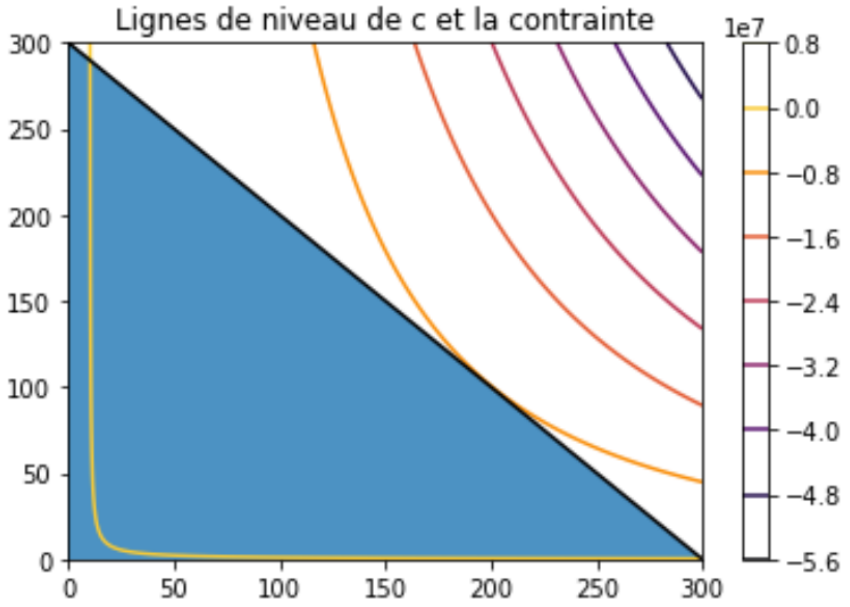
```
    elif(np.all(np.less(vp,np.zeros(2)))): # teste si toutes les valeurs propres␣
 ↪sont strictement négative
        return( 'est un maximum local')
    elif(np.any(np.equal(vp,np.zeros(2)))): # teste si au moins une des valeurs␣
 ↪propres est nulle
        return('on ne sait pas, le critère de la hessienne est non concluant')
    else:
        return('est un point selle')

print()
print('Classification des points critiques de la fonction c sur R^2:')

for pt in pointC:
    Hess = np.array(HessC.subs(dict(zip(X, pt)))).astype('float')
    print(pt, classification(Hess))

# on rappelle que pour la fonction de coût sur (R^2)_{+}
# l'unique point critique est alors (10,5)
print()
print("Il n'y a donc pas d'extrema sans contrainte, ce qui est cohérent")
print("puisque sans contrainte la fonction est convexe et concave.")

# On va tout de même afficher les points selles
x = np.linspace(-50,50,1000)
y = np.linspace(-50,50,1000)
X,Y = np.meshgrid(x,y)
Cout = c([X,Y])

# Affichage des lignes de niveau de la fonction c et de ses points selles
plt.figure(figsize = (8,8))
plt.contour(X,Y,Cout,100)
plt.colorbar()
plt.title("Lignes de niveau de la fonction c sur [-50,50]x[-50,50]", fontsize =␣
 ↪12)
plt.axis("square")
plt.plot(pointC[0][0],pointC[0][1],'r*') # affiche (-10,-5)
plt.plot(pointC[1][0],pointC[1][1],'r*') # affiche (10,5)
plt.show()
```

```
Les points critiques de c sur R^2 sont : { (-10, -5) , (10, 5) }

Classification des points critiques de la fonction c sur R^2:
(-10, -5) est un point selle
(10, 5) est un point selle

Il n'y a donc pas d'extrema sans contrainte, ce qui est cohérent
puisque sans contrainte la fonction est convexe et concave.
```

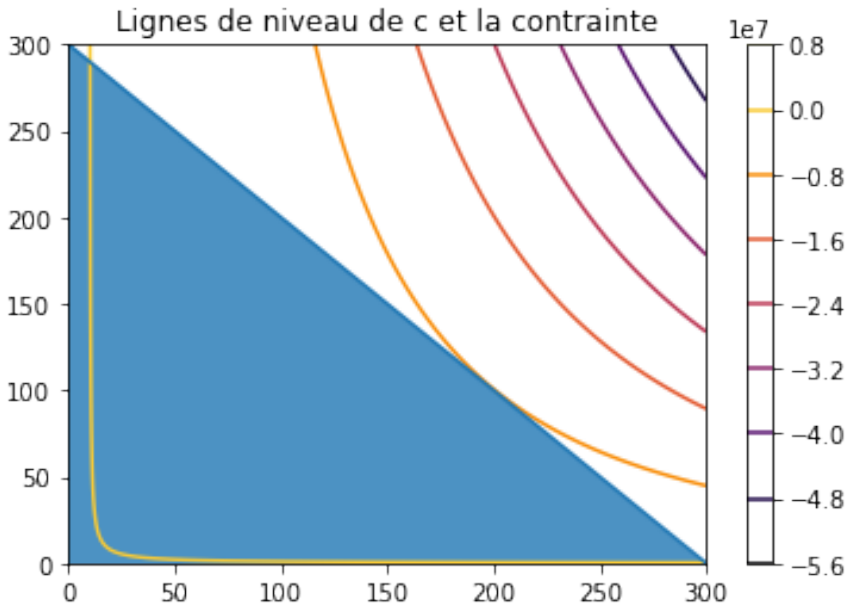


3.4 Les lignes de niveaux et l'ensemble de contrainte(s)

On rappelle que les bords de l'ensemble (en noir), defini par $y = 300 - x$, $x = 0$ et $y = 0$ sont inclus dans l'ensemble des contraintes.

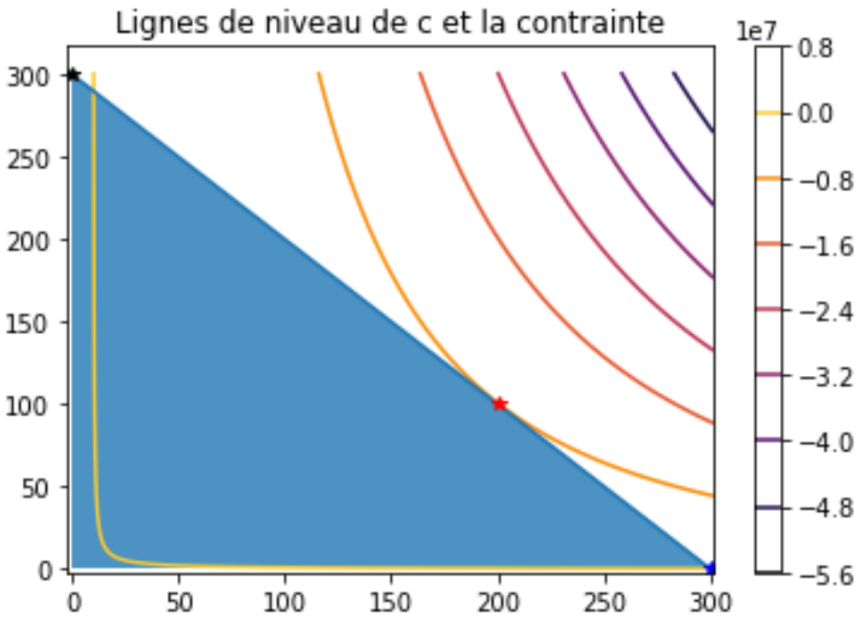


[16]:
```
def c(X):
    return (600 + 200*X[0] + 200*X[1] - 2*(X[0]**2)*X[1])

# la contrainte est x + y <= 300 donc y <= 300 - x

# affichage des lignes de niveau de la fonction coût c et de la contrainte y <=
 ↪300 - x
x = np.linspace(0,300,1000)
y = np.linspace(0,300,1000)
X,Y = np.meshgrid(x,y)
cout = c([X,Y])

plt.figure()
plt.contour(X,Y,cout, cmap = 'inferno')
plt.colorbar()
plt.plot(x, 300 - x) # affiche y = 300 - x
plt.fill([0, 0, 300], [300, 0,0], alpha = 0.8) # sans oublier que x >= 0 et y >=
 ↪0

plt.title('Lignes de niveau de c et la contrainte')
```

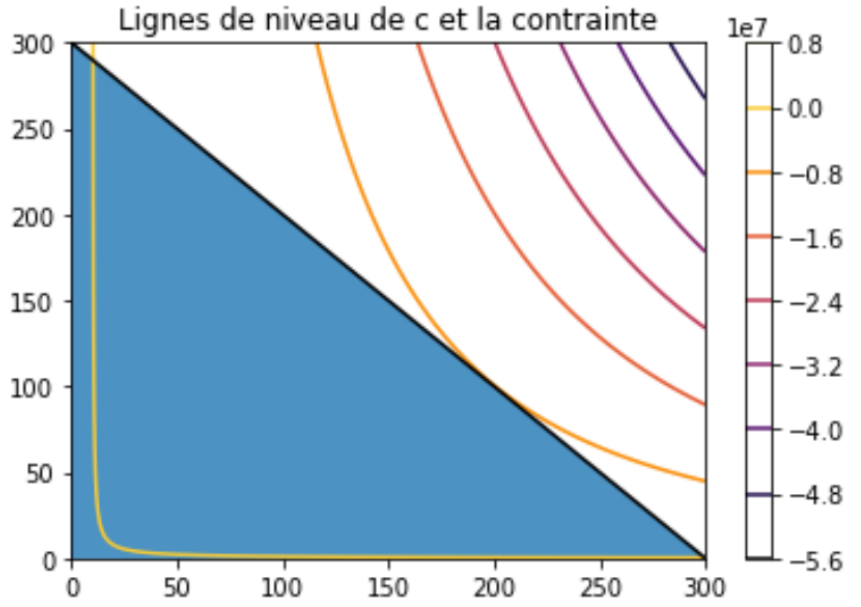


3.5 Trouve et affiche les extrema sous contrainte de la fonction coût

Avec l'ensemble de contrainte K, on voit que la fonction c admet un point de minimum (en rouge) et deux points de maximum (en noir et bleu).



On rappelle que les bords de l'ensemble (en noir), defini par $y = 300 - x$, $x = 0$ et $y = 0$ sont inclus dans l'ensemble des contraintes.



L'ensemble K est compact puisque:

- fermé car c'est la contre-image d'intervalle fermé $]-∞; 0]$parunefonctioncontinue
- borné par la boule centré en 0 et de rayon 500

Sur l'ensemble des contraintes K, la fonction c, continue, admet donc un point de maximum et un point de miminum.

[17]:
```
def c(X):
    return (600 + 200*X[0] + 200*X[1] - 2*(X[0]**2)*X[1]) # on rappelle la␣
 ↪fonction coût c

# comme arg[max(c(X))] = arg[min(-c(X))]
def cop(X):
    return (-c(X)) # on définit la fonction cop pour trouver le maximum de la␣
 ↪fonction sous contrainte

# contrainte: x + y <= 300 donc 0 <= - x - y + 300
def psi(X):
    return -X[0] - X[1] + 300

cons = ({'type':'ineq', 'fun': lambda X: psi(X)})

# fonction qui retient les itérations dans l'optimisation
def reporter(p):
    global Xn
    Xn.append(p)

# on prend un point initial, comme il y a une partie concave et une partie␣
 ↪convexe sur R^2
# il faut faire attention au point inital choisi sinon cela ne va pas converger

# trouvons le point de minimum
x0 = [29,36]
Xn = [x0]
res_mini = opt.minimize(c,x0, method = 'SLSQP',callback =␣
 ↪reporter,constraints=cons)
res_mini
Xn # permet de voir les itérations faites pour trouver le point de minimum sous␣
 ↪la contrainte

# trouvons le point de maximum
x0 = [0,212]
Xn = [x0]
res_maxi = opt.minimize(cop,x0, method = 'SLSQP',callback =␣
 ↪reporter,constraints=cons)
res_maxi
Xn # permet de voir les itérations faites pour trouver le point de maximum sous␣
 ↪la contrainte

# les points d'extrema sous contrainte sont très proche de nos résultats␣
 ↪précédents

# Affichage des extremas, sous la contrainte, de la fonction de coût c
```

```
x = np.linspace(-2,301.5,1000)
y = np.linspace(-2,301.5,1000)
X,Y = np.meshgrid(x,y)
cout = c([X,Y])

m = res_mini.x # point de minimum
n = res_maxi.x # point de maximum

plt.figure()
plt.contour(X,Y,cout, cmap = 'inferno')
plt.colorbar()
plt.plot(x, 300 - x) # la contrainte est x + y <= 300 donc y <= 300 - x
plt.fill([0, 0, 300], [300, 0,0], alpha = 0.8) # sans oublier que x >= 0 et y >=␣
 ↪0
plt.plot(m[0],m[1],'r*')
plt.plot(n[0],n[1],'k*')
plt.plot(n[1],n[0],'b*')

plt.title('Lignes de niveau de c et la contrainte')
```

[17]:
```
      fun: -8020016.003438291
      jac: array([-80299.5625, -80414.625 ])
  message: 'Positive directional derivative for linesearch'
     nfev: 34
      nit: 12
     njev: 8
   status: 8
  success: False
        x: array([200.76680794, 100.24009478])
```

[17]:
```
[[29, 36],
 array([ 1393.49999953, -1093.50000047]),
 array([283.42847678,  16.57157912]),
 array([107.3840197 , 190.12039784]),
 array([ 6042.11618144, -5741.95498967]),
 array([199.99445945,  99.84867219]),
 array([202.14905285,  99.84309112]),
 array([-95.79706143, 395.19767624]),
 array([200.76680794, 100.24009478])]
```

[17]:
```
      fun: -60600.000017659164
      jac: array([-200., -200.])
  message: 'Optimization terminated successfully'
     nfev: 18
      nit: 5
     njev: 5
   status: 0
```

```
      success: True
            x: array([4.22558757e-08, 3.00000000e+02])
```

[17]:
```
[[0, 212],
 array([ 44., 256.]),
 array([8.02694150e-03, 2.99991973e+02]),
 array([-1.64391357e-03,  3.00001644e+02]),
 array([4.22558741e-08, 3.00000000e+02]),
 array([4.22558757e-08, 3.00000000e+02])]
```