Mathematical Economics

Alpha Chiang

Chapter 12

Optimization with Equality Constraints

Lagrange-Multiplier Method

In [5]:

```
from sympy import *
import numpy as np
from scipy.linalg import cholesky, solve_triangular
from scipy.linalg import cho_solve, cho_factor
from scipy.linalg import solve
from scipy.optimize import minimize
from sympy import Symbol, dsolve, Function, Derivative, Eq
x1 = Symbol("x_1")
x2 = Symbol('x_2')
Z = Symbol("Z")
lamd = Symbol("\\lambda")
eq1 = Eq(Z, x1*x2 + 2*x1 + lamd*(60 - 4*x1 -2*x2))
display(eq1)




def f(x):
    return -(x[0]*x[1] + 2*x[0])

cons = ({'type': 'eq',
         'fun' : lambda x: np.array([4*x[0] + 2*x[1] - 60])})

x0 = np.array([1,1,4])
res = minimize(f, x0, constraints=cons)
res
```

$$Z = \lambda(-4x_1 - 2x_2 + 60) + x_1x_2 + 2x_1$$

Out[5]:
```
     fun: -127.99999999999983
     jac: array([-16.        ,  -7.99999809,   0.        ])
 message: 'Optimization terminated successfully'
    nfev: 16
     nit: 4
    njev: 4
  status: 0
 success: True
       x: array([ 7.9999997, 14.0000006,  4.       ])
```

In [8]:
```
res = minimize(f, x0, constraints=cons,method="trust-constr")
```

Example 1

In [9]:
```
y = Symbol('y')
x = Symbol('x')
eq1 = Eq(Z, x*y + lamd*(6 - x - y))
display(eq1)

def f(x):
    return -(x[0]*x[1])

cons = ({'type': 'eq',
         'fun' : lambda x: np.array([x[0] + x[1] - 6])})

x0 = np.array([1,1,3])
res = minimize(f, x0, constraints=cons)
res
```

$$Z = \lambda(-x - y + 6) + xy$$

Out[9]:
```
     fun: -8.999999999999998
     jac: array([-3., -3.,  0.])
 message: 'Optimization terminated successfully'
    nfev: 8
     nit: 2
    njev: 2
  status: 0
 success: True
       x: array([3., 3., 3.])
```

Example 2

In [10]:
```
x1 = Symbol("x_1")
x2 = Symbol('x_2')
Z = Symbol("Z")
lamd = Symbol("\\lambda")
eq1 = Eq(Z, x1**2 + x2**2   + lamd*(2 - x1  - 4*x2))
display(eq1)


def f(x):
    return (x[0]**2 + x[1]**2)

cons = ({'type': 'eq',
         'fun' : lambda x: np.array([x[0] + 4*x[1] - 2])})

x0 = np.array([1,1,1])
res = minimize(f, x0, constraints=cons)
res
```

$$Z = \lambda\left(-x_1 - 4x_2 + 2\right) + x_1^2 + x_2^2$$

Out[10]:
```
     fun: 0.2352941176470589
     jac: array([0.23529412, 0.94117649, 0.        ])
 message: 'Optimization terminated successfully'
    nfev: 16
     nit: 4
    njev: 4
  status: 0
 success: True
       x: array([0.11764705, 0.47058824, 1.        ])
```

Example 2 --Using derivatives and matrices--

In [11]:
```
x1 = Symbol("x_1")
x2 = Symbol('x_2')
Z = Symbol("Z")
lamd = Symbol("\\lambda")

def z(x1,x2,lamd):
    return x1**2 + x2**2   + lamd*(2 - x1  - 4*x2)
def Z(x1,x2,lamd):
    dZ1 = diff(z(x1,x2,lamd),x1)
    dZ2 = diff(z(x1,x2,lamd),x2)
    dZ3 = diff(z(x1,x2,lamd),lamd)
    return dZ1,dZ2,dZ3
Z(x1,x2,lamd)
```

Out[11]: (-\lambda + 2*x_1, -4*\lambda + 2*x_2, -x_1 - 4*x_2 + 2)

In [12]:
```
# We can build a matrix
A = np.array([
    [2, 0, -1],
    [0, 2, -4],
    [-1, -4, 0]
])
b = np.array([0, 0, -2])
solve(A, b)
```

Out[12]: array([0.11764706, 0.47058824, 0.23529412])

In [13]:
```
x1 = Symbol("x_1")
x2 = Symbol('x_2')
Z = Symbol("Z")
lamd = Symbol("\\lambda")

def z(x1,x2,lamd):
    return x1**2 + x2**2   + lamd*(2 - x1  - 4*x2)
def g(x1,x2):
    return x1 + 4*x2 - 2
def Z(x1,x2,lamd):
    dZ1 = diff(z(x1,x2,lamd),x1,2)
    dZ2 = diff(z(x1,x2,lamd),x2,2)
    dZ3 = diff(z(x1,x2,lamd),lamd,2)
    dZ4 = diff(g(x1,x2),x1)
    dZ5 = diff(g(x1,x2),x2)
    return dZ1,dZ2,dZ3,dZ4,dZ5
Z(x1,x2,lamd)
# By using these values we can build a hessian matrix
# and we can check maximum and minimum values
```

Out[13]: (2, 2, 0, 1, 4)

In [14]:
```
x1 = Symbol("x_1")
x2 = Symbol('x_2')
U = Symbol("U")
lamd = Symbol("\\lambda")
B = Symbol("B")
r = Symbol("r")
eq1 = Eq(U, x1*x2   + lamd*(B - x1 - (x2/(1+r)) ))
```

```
display(eq1)


def u(x1,x2,lamd,B,r):
    return x1*x2 + lamd*(B - x1 - (x2/(1+r)))
def U(x1,x2,lamd,B,r):
    dU1 = diff(u(x1,x2,lamd,B,r),x1)
    dU2 = diff(u(x1,x2,lamd,B,r),x2)
    dU3 = diff(u(x1,x2,lamd,B,r),lamd)
    return dU1,dU2,dU3
display(U(x1,x2,lamd,B,r))
a = lamd/(lamd/(1+r))
display(a)
```

$$U = \lambda\left(B - x_1 - \frac{x_2}{r+1}\right) + x_1 x_2$$

(-\lambda + x_2, -\lambda/(r + 1) + x_1, B - x_1 - x_2/(r + 1))

$$r + 1$$

In [15]:

```
x1 = Symbol("x_1")
x2 = Symbol('x_2')
U = Symbol("U")
lamd = Symbol("\\lambda")
B = Symbol("B")
r = Symbol("r")
eq1 = Eq(U, x1*x2   + lamd*(B - x1 - (x2/(1+r)) ))
display(eq1)


def u(x1,x2,lamd,B,r):
    return x1*x2 + lamd*(B - x1 - (x2/(1+r)))

def g(x1,x2,B,r):
    return x1 + x2/(1+r) - B


def U(x1,x2,lamd,B,r):
    dU1 = diff(u(x1,x2,lamd,B,r),x1,2)
    dU2 = diff(u(x1,x2,lamd,B,r),x2,2)
    dU3 = diff(u(x1,x2,lamd,B,r),x1,x2)
    dU4 = diff(u(x1,x2,lamd,B,r),lamd,2)
    dg1 = diff(g(x1,x2,B,r),x1)
```

```
    dg2 = diff(g(x1,x2,B,r),x2)

    return dU1,dU2,dU3,dU4,dg1,dg2
display(U(x1,x2,lamd,B,r)) # can be used again to build the hessian
```

$$U = \lambda \left(B - x_1 - \frac{x_2}{r+1}\right) + x_1 x_2$$

(0, 0, 1, 0, 1, 1/(r + 1))

In [17]:
```
dU1 = diff(u(x1,x2,lamd,B,r),x1,2)
dU2 = diff(u(x1,x2,lamd,B,r),x2,2)
dU3 = diff(u(x1,x2,lamd,B,r),x1,x2)
dU4 = diff(u(x1,x2,lamd,B,r),lamd,2)
dg1 = diff(g(x1,x2,B,r),x1)
dg2 = diff(g(x1,x2,B,r),x2)
M1 = Matrix([[dU1, -dU3, -dg2],
 [-dU3, dU2, dU3],
 [-dg2, dU3, dU1]])
display(M1)
display(M1.det())
```

$$\begin{bmatrix} 0 & -1 & -\frac{1}{r+1} \\ -1 & 0 & 1 \\ -\frac{1}{r+1} & 1 & 0 \end{bmatrix}$$

$$\frac{2}{r+1}$$

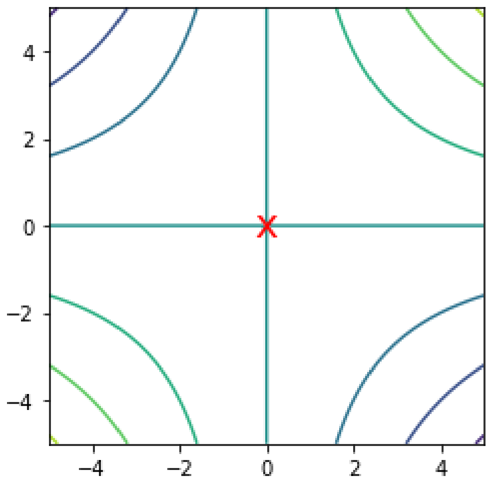
Plots of Examples

Using https://www2.hawaii.edu/~jonghyun/courses.html

Example 1

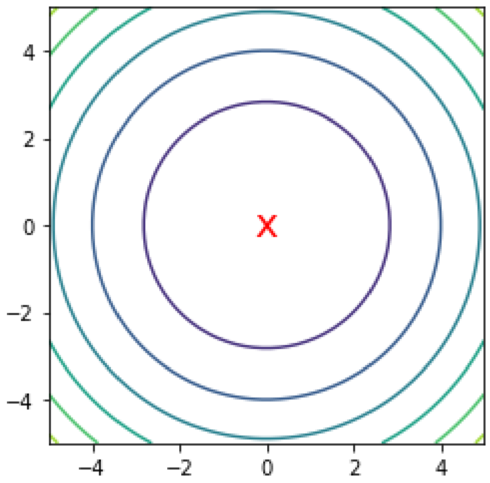
In [18]:
```
import scipy.optimize as opt
import numpy as np
import matplotlib.pyplot as plt
def func(x):
    return x[0]*x[1]
x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
```

```python
X,Y = np.meshgrid(x,y)
XY = np.vstack([X.ravel(), Y.ravel()])
Z = func(XY).reshape(50,50)
plt.contour(X, Y, Z)
plt.text(0, 0, 'x', va='center', ha='center',
color='red', fontsize=20)
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
```



In [19]:

```python
import scipy.optimize as opt
import numpy as np
import matplotlib.pyplot as plt
def func(x):
    return x[0]**2 + x[1]**2
x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
X,Y = np.meshgrid(x,y)
XY = np.vstack([X.ravel(), Y.ravel()])
Z = func(XY).reshape(50,50)
plt.contour(X, Y, Z)
plt.text(0, 0, 'x', va='center', ha='center',
color='red', fontsize=20)
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
```



EXERCISE 12.5 --Q1--

In [20]:
```
x = Symbol("x")
y = Symbol('y')
U = Symbol("U")
lamd = Symbol("\\lambda")
eq1 = Eq(U, (x+2)*(y+1)   + lamd*(130 -4*x  - 6*y))
display(eq1)


def f(x):
    return -((x[0]+2) * (x[1]+1))

cons = ({'type': 'eq',
         'fun' : lambda x: np.array([4*x[0] + 6*x[1] - 130])})

x0 = np.array([1,1,3])
res = minimize(f, x0, constraints=cons)
res
```

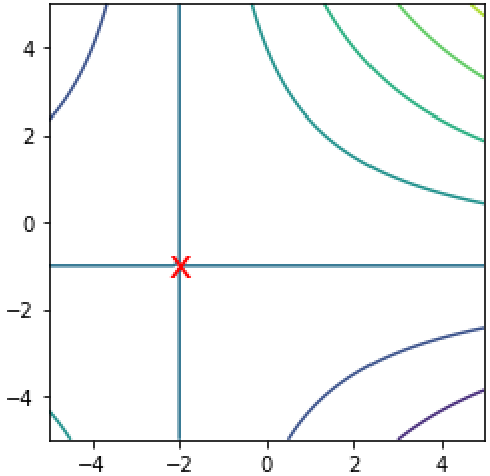
$$U = \lambda\left(-4x - 6y + 130\right) + \left(x + 2\right)\left(y + 1\right)$$

Out[20]:
```
     fun: -215.99999999999963
     jac: array([-12., -18.,   0.])
 message: 'Optimization terminated successfully'
    nfev: 16
     nit: 4
    njev: 4
```

```
 status: 0
success: True
      x: array([16.0000008 , 10.99999947,  3.        ])
```

In [21]:

```
def func(x):
    return (x[0] +2)*(x[1] + 1)
x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
X,Y = np.meshgrid(x,y)
XY = np.vstack([X.ravel(), Y.ravel()])
Z = func(XY).reshape(50,50)
plt.contour(X, Y, Z)
plt.text(-2, -1, 'x', va='center', ha='center',
color='red', fontsize=20)
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
```



Furkan Zengin