Mathematical Economics

Alpha Chiang

Chapter 13

Further Topics in Optimization

Example 1

In [2]:

```
from sympy import *
import numpy as np
from sympy import Symbol, dsolve, Function, Derivative, Eq
from scipy.optimize import minimize, rosen, rosen_der
x = Symbol("x")
y = Symbol('y')
Z = Symbol("Z")
lamd1 = Symbol("\\lambda_1")
lamd2 = Symbol("\\lambda_2")
eq1 = Eq(Z, x*y + lamd1*(100 - x - y) + lamd2*(40 - x))
display(eq1)

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

cons = ({'type': 'ineq',
         'fun' : lambda x: np.array([x[0] + x[1] - 100, x[0] - 40])})
x0 = np.array([2,2,1])
res = minimize(f, x0, constraints=cons)
res
```

$$Z = \lambda_1 \left(-x - y + 100\right) + \lambda_2 \left(40 - x\right) + xy$$

Out[2]:

```
     fun: 2500.000000003316
     jac: array([50., 50.,  0.])
 message: 'Optimization terminated successfully'
    nfev: 8
     nit: 2
```

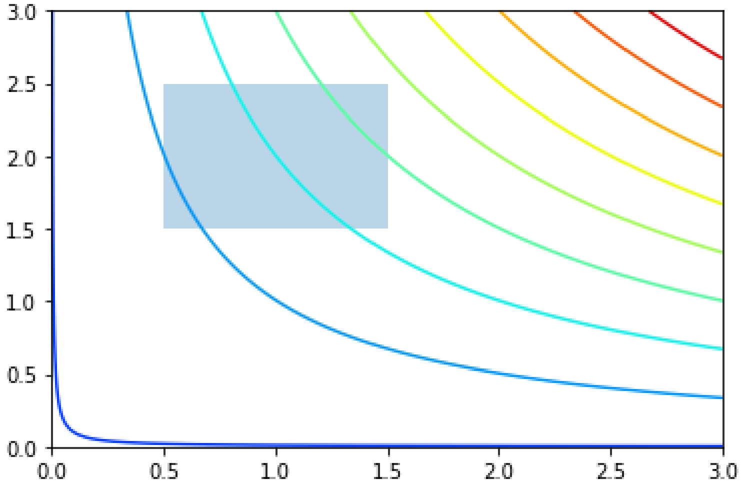
```
   njev: 2
 status: 0
success: True
      x: array([50., 50.,  1.])
```

In [3]:

```
%matplotlib inline
import scipy.linalg as la
import numpy as np
import scipy.optimize as opt
import matplotlib.pyplot as plt
import pandas as pd
x = np.linspace(0, 3, 100)
y = np.linspace(0, 3, 100)
X, Y = np.meshgrid(x, y)
Z = f(np.vstack([X.ravel(), Y.ravel()])).reshape((100,100))
plt.contour(X, Y, Z, np.arange(-1.99,10, 1), cmap='jet');
plt.fill([0.5,0.5,1.5,1.5], [2.5,1.5,1.5,2.5], alpha=0.3)
plt.axis([0,3,0,3])
```

Out[3]:

```
(0.0, 3.0, 0.0, 3.0)
```



In [4]:

```
# Another way for example 1
def func(x, sign=1.0):
    return sign*(x[0]*x[1])
def func_deriv(x, sign=1.0):
    dfdx0 = sign*(x[1])
    dfdx1 = sign*(x[0])
```

```
    return np.array([ dfdx0, dfdx1 ])
# take the derivative of objective function
```

In [5]:

```
cons = ({'type': 'ineq',
         'fun' : lambda x: np.array([x[0] + x[1] - 100]),
         'jac' : lambda x: np.array([1,1])},
        {'type': 'ineq',
         'fun' : lambda x: np.array([x[0] - 40]),
         'jac' : lambda x: np.array([1, 0])})
# for jac we take derivatives of constraints
```

In [6]:

```
res = minimize(func, [10,10],  jac=func_deriv,
               constraints=cons, method='SLSQP', options={'disp': True})

print(res.x)
```

```
Optimization terminated successfully    (Exit mode 0)
            Current function value: 2500.000000003316
            Iterations: 2
            Function evaluations: 2
            Gradient evaluations: 2
[50. 50.]
```

Example 2

In [7]:

```
x1 = Symbol("x_")
x2 = Symbol('x_2')
Z = Symbol("Z")
lamd1 = Symbol("\\lambda_1")
lamd2 = Symbol("\\lambda_2")
eq1 = Eq(Z, (x1-4)**2 + (x2-4)**2 +
         lamd1*(6 - 2*x1 - 3*x2) + lamd2*(-12 + 3*x1 + 2*x2))
display(eq1)

def f(x):
    return ((x[0] - 4)**2 + (x[1] - 4)**2)

cons = ({'type': 'ineq',
         'fun' : lambda x: np.array([2*x[0] + 3*x[1] - 6,
                                     -3*x[0] - 2*x[1] +12])})

x0 = np.array([2,2,1])
```

```
res = minimize(f, x0, constraints=cons)
res
```

$$Z = \lambda_1 \left(-2x - 3x_2 + 6\right) + \lambda_2 \left(3x + 2x_2 - 12\right) + \left(x - 4\right)^2 + \left(x_2 - 4\right)^2$$

Out[7]:

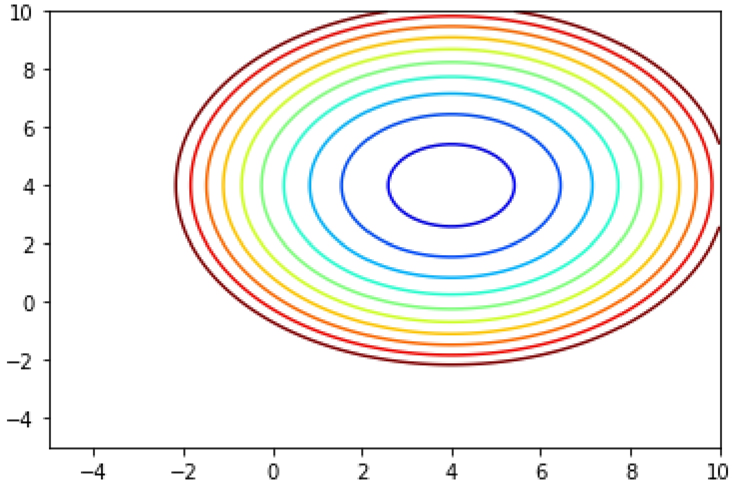
```
     fun: 4.92307692307701
     jac: array([-3.69230771, -2.46153849,  0.        ])
 message: 'Optimization terminated successfully'
    nfev: 16
     nit: 4
    njev: 4
  status: 0
 success: True
       x: array([2.15384616, 2.76923076, 1.        ])
```

In [8]:

```
x = np.linspace(-5, 10, 100)
y = np.linspace(-5, 10, 100)
X, Y = np.meshgrid(x, y)
Z = f(np.vstack([X.ravel(), Y.ravel()])).reshape((100,100))
plt.contour(X, Y, Z, np.arange(-1.99,40, 4), cmap='jet');
plt.axis([-5,10,-5,10])
```

Out[8]:

```
(-5.0, 10.0, -5.0, 10.0)
```



13.2 The Constraint Qualification

Example 3

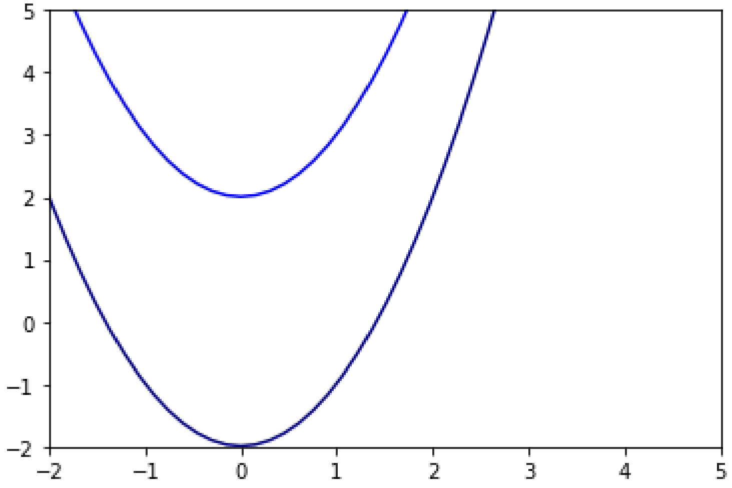
In [9]:
```
x1 = Symbol("x_1")
x2 = Symbol('x_2')
Z = Symbol("Z")
lamd1 = Symbol("\\lambda_1")
lamd2 = Symbol("\\lambda_2")
eq1 = Eq(Z, x2 - x1**2 +
         lamd1*(10 - x1**2 - x2)**3 + lamd2*(-2 + x1))
display(eq1)

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

cons = ({'type': 'ineq',
         'fun' : lambda x: np.array([-(10 - x[0]**2 - x[1])**3,
                                     -x[0] + 2])})

x0 = np.array([2,2,1])

res = minimize(f, x0, constraints=cons)
res
```

$$Z = \lambda_1 \left(-x_1^2 - x_2 + 10\right)^3 + \lambda_2 \left(x_1 - 2\right) - x_1^2 + x_2$$

Out[9]:
```
     fun: 1.9999981835694722
     jac: array([-4.,  1.,  0.])
 message: 'Optimization terminated successfully'
    nfev: 145
     nit: 36
    njev: 36
  status: 0
 success: True
       x: array([2.        , 5.99999818, 1.        ])
```

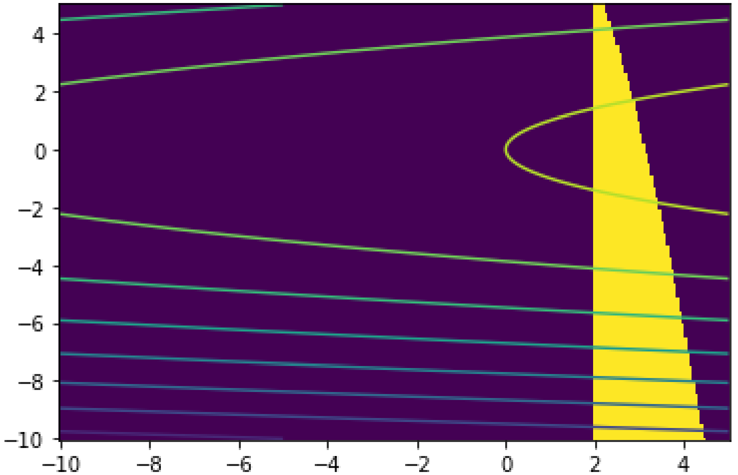
In [11]:
```
x = np.linspace(-5, 10, 100)
y = np.linspace(-5, 10, 100)
X, Y = np.meshgrid(x, y)
Z = f(np.vstack([X.ravel(), Y.ravel()])).reshape((100,100))
plt.contour(X, Y, Z, np.arange(-1.99,40, 4), cmap='jet');
plt.axis([-2,5,-2,5])
```

Out[11]:
```
(-2.0, 5.0, -2.0, 5.0)
```



In [12]:

```
X, Y = np.meshgrid(np.linspace(-10, 5, 256)
                   , np.linspace(-10, 5, 256))

plt.figure()
plt.pcolormesh(X, Y, (-(10 - X**2 - Y)**3 <= 0) &
               (-X + 2 <= 0),shading='auto')
plt.contour(X, Y, X - Y**2)
plt.show()
```

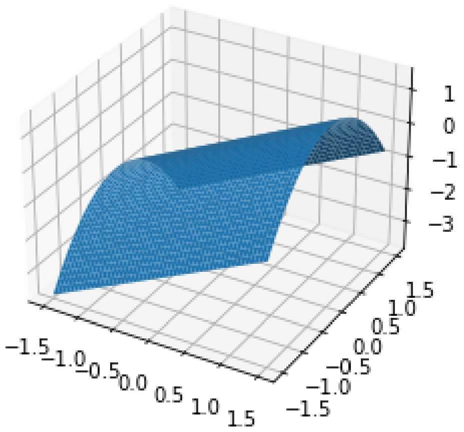


In [13]:

```
import numpy as np
```

```
x = np.linspace(-1.5, 1.5)

[X, Y] = np.meshgrid(x, x)

import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.gca(projection='3d')

ax.plot_surface(X, Y, X - Y**2)
```

Out[13]: <mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x1a636a83190>



13.3 Economic Applications

Example 1

In [14]:
```
x = Symbol("x")
y = Symbol('y')
Z = Symbol("Z")
lamd1 = Symbol("\\lambda_1")
lamd2 = Symbol("\\lambda_2")
eq1 = Eq(Z, x*y**2 +
          lamd1*(100 - x -y) + lamd2*(120 - 2*x -y))
display(eq1)
```

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

cons = ({'type': 'ineq',
         'fun' : lambda x: np.array([(x[0] + x[1] - 100)
                                     ,(2*x[0] + x[1] - 120)])})

x0 = np.array([10,20,0])

res = minimize(f, x0, constraints=cons)
res
# this cannot be solved by these method
# we should use derivatives and matrices below
```

$$Z = \lambda_1 \left(-x - y + 100\right) + \lambda_2 \left(-2x - y + 120\right) + xy^2$$

Out[14]:

```
     fun: 3.139129143754785e-07
     jac: array([ 2.93285041e-09, -1.15913628e-02,  0.00000000e+00])
 message: 'Optimization terminated successfully'
    nfev: 46
     nit: 10
    njev: 10
  status: 0
 success: True
       x: array([ 1.07033402e+02, -5.41557940e-05,  0.00000000e+00])
```

In [15]:

```python
x = Symbol("x")
y = Symbol('y')
Z = Symbol("Z")
lamd1 = Symbol("\\lambda_1")
lamd2 = Symbol("\\lambda_2")
def z(x,y,lamd1,lamd2):
    return x*y**2+lamd1*(100- x -y)+lamd2*(120 - 2*x-y)

def Z(x,y,lamd1,lamd2):
    dZ1 = diff(z(x,y,lamd1,lamd2),x)
    dZ2 = diff(z(x,y,lamd1,lamd2),y)
    dZ3 = diff(z(x,y,lamd1,lamd2),lamd1)
    dZ4 = diff(z(x,y,lamd1,lamd2),lamd2)
    return dZ1,dZ2,dZ3, dZ4
Z(x,y,lamd1,lamd2)
# Assume lamd1 = 0
```

Out[15]:

```
(-\lambda_1 - 2*\lambda_2 + y**2,
 -\lambda_1 - \lambda_2 + 2*x*y,
```

```
-x - y + 100,
-2*x - y + 120)
```

first install gekko from https://gekko.readthedocs.io/en/latest/

In [16]:

```python
# first install gekko from https://gekko.readthedocs.io/en/latest/
# a1 = x, a2 = y, a3 = lambda2
from gekko import GEKKO
m = GEKKO()
a1,a2,a3 = [m.Var(1) for i in range(3)]
m.Equations([a2**2 - 2*a3==0,\
             2*a1*a2 - a3==0,\
             120 - 2*a1 - a2==0])
m.solve(disp=False)
print(a1.value,a2.value,a3.value)
```

```
[20.0] [79.999999999] [3199.9999999]
```

Furkan zengin