1 Correction du TD 12

On rapelle le problème d'optimisation à résoudre :

$$\left\{ \begin{array}{l} \min_U J(U) = u_1^2 + u_1^2 - u_1 u_2 + u_2 \\ \text{sous contrainte } \theta(U) = u_0 + 2u_1 - 1 = 0 \text{ et } u_1 \geqslant 0.5 \end{array} \right. \tag{1}$$

Les implémentations de l'algorithme Arrow-Hurwicz proposées ici ne sont pas génériques. Il n'est pas suggéré de les réutiliser à moins d'utiliser pleinement le calcul matriciel de `numpy`.

Première demi-heure : Premier problème .

Le module `cvxopt` utilise une fonction qui retourne la valeur de la fonction à optimiser, sa dérivée, sa dérivée seconde.

$$f(x,y) = x^2 + y^2 - xy + y \tag{2}$$
$$\frac{\partial f(x,y)}{\partial x} = 2x - y \tag{3}$$
$$\frac{\partial f(x,y)}{\partial y} = 2y - x + 1 \tag{4}$$
$$\frac{\partial^2 f(x,y)}{\partial x^2} = 2 \tag{5}$$
$$\frac{\partial^2 f(x,y)}{\partial y^2} = 2 \tag{6}$$
$$\frac{\partial^2 f(x,y)}{\partial x \partial y} = -1 \tag{7}$$

```
from cvxopt import solvers, matrix
import random

def fonction(x=None,z=None) :
    if x is None :
        x0 = matrix ( [[ random.random(), random.random() ]])
        return 0,x0
    f = x[0]**2 + x[1]**2 - x[0]*x[1] + x[1]
    d = matrix ( [ x[0]*2 - x[1], x[1]*2 - x[0] + 1 ] ).T
    h = matrix ( [ [ 2.0, -1.0], [-1.0, 2.0] ])
    if z is None: return  f, d
    else : return f, d, h

A = matrix([ [ 1.0, 2.0 ] ]).trans()
b = matrix ( [[ 1.0] ] )

sol = solvers.cp ( fonction, A = A, b = b)
print (sol)
print (sol['x'])
```

Solution numérique : `4.29e-01, 2.86e-01`

Seconde et troisième demi-heure : L'algorithme de Arrow-Hurwicz .

```
def fonction(X) :
    x,y = X
    f = x**2 + y**2 - x*y + y
    d = [ x*2 - y, y*2 - x + 1  ]
    return f, d

def contrainte(X) :
    x,y = X
    f = x+2*y-1
    d = [ 1,2]
    return f, d

X0  = [ random.random(),random.random() ]
p0  = random.random()
epsilon = 0.1
rho     = 0.1

diff = 1
iter = 0
while diff > 1e-10 :
    f,d   = fonction( X0 )
    th,dt = contrainte( X0 )
    Xt    = [ X0[i] - epsilon*(d[i] + dt[i] * p0) for i in range(len(X0)) ]

    th,dt = contrainte( Xt )
    pt    = p0 + rho * th

    iter += 1
    diff = sum ( [ abs(Xt[i] - X0[i]) for i in range(len(X0)) ] )
    X0 = Xt
    p0 = pt
    if iter % 100 == 0 :
        print ("i {0} diff {1}".format(iter,diff),":", f,X0,p0,th)

print (diff,iter,p0,X0)
```

Solution numérique : 352 itérations, `[0.4285714278354612, 0.2857142845509337]`

Quatrième demi-heure : Lagrangien augmenté .

```
def fonction(X,c) :
    x,y = X
    f = x**2 + y**2 - x*y + y
    d = [ x*2 - y, y*2 - x + 1 ]

    v = x+2*y-1
    v = c/2 * v**2

    dv = [ 2*(x+2*y-1), 4*(x+2*y-1) ]
    dv = [ c/2 * dv[0], c/2 * dv[1] ]
    return f + v, d, dv

def contrainte(X) :
    x,y = X
    f = x+2*y-1
    d = [ 1,2]
    return f, d

X0  = [ random.random(),random.random() ]
```

```
p0  = random.random()
epsilon = 0.1
rho     = 0.1
c       = 1

diff = 1
iter = 0
while diff > 1e-10 :
    f,d,dv = fonction( X0,c )
    th,dt = contrainte( X0 )
    Xt    = [ X0[i] - epsilon*(d[i] + dt[i] * p0 + dv[i]) for i in range(len(X0)) ]

    th,dt = contrainte( Xt )
    pt    = p0 + rho * th

    iter += 1
    diff = sum ( [ abs(Xt[i] - X0[i]) for i in range(len(X0)) ] )
    X0 = Xt
    p0 = pt
    if iter % 100 == 0 :
        print ("i {0} diff {1}".format(iter,diff),":", f,X0,p0,th)

print (diff,iter,p0,X0)
```

Solution numérique : 222 itérations, `[0.4285714280857629, 0.2857142852113399]`

Pour aller plus loin ou pour ceux qui ont fini plus tôt .

Prolongement 1

Le problème à résoudre est le suivant :

$$\left\{ \begin{array}{l} \min_U J(U) = u_1^2 + u_1^2 - u_1 u_2 + u_2 \\ \text{sous contrainte } \theta(U) = u_0 + 2u_1 - 1 = 0 \text{ et } u_1 \geqslant 0.3 \end{array} \right. \tag{8}$$

```
from cvxopt import solvers, matrix
import random

def fonction(x=None,z=None) :
    if x is None :
        x0 = matrix ( [[ random.random(), random.random() ]])
        return 0,x0
    f = x[0]**2 + x[1]**2 - x[0]*x[1] + x[1]
    d = matrix ( [ x[0]*2 - x[1], x[1]*2 - x[0] + 1 ] ).T
    h = matrix ( [ [ 2.0, -1.0], [-1.0, 2.0] ])
    if z is None: return  f, d
    else : return f, d, h

A = matrix([ [ 1.0, 2.0 ] ]).trans()
b = matrix ( [[ 1.0] ] )

G = matrix ( [[0.0, -1.0] ]).trans()
h = matrix ( [[ -0.3] ] )

sol = solvers.cp ( fonction, A = A, b = b, G=G, h=h)
print (sol)
print (sol['x'])
```

Solution numérique : `[ 0.40, 0.30]`

Version de l'algorithme de Arrow-Hurwicz :

```
import numpy,random

X0 = numpy.matrix ( [[ random.random(), random.random() ]]).transpose()
P0 = numpy.matrix ( [[ random.random(), random.random() ]]).transpose()

A  = numpy.matrix([ [ 1.0, 2.0 ], [ 0.0, -1.0]  ])
tA = A.transpose()
b = numpy.matrix ( [[ 1.0], [-0.30] ] )

epsilon = 0.1
rho     = 0.1
c       = 1

first = True
iter  = 0
while first or abs(J - oldJ) > 1e-8 :
    if first :
        J = X0[0,0]**2 + X0[1,0]**2 - X0[0,0]*X0[1,0] + X0[1,0]
        oldJ = J+1
        first = False
    else :
        oldJ = J
        J = X0[0,0]**2 + X0[1,0]**2 - X0[0,0]*X0[1,0] + X0[1,0]

    dj   = numpy.matrix ( [ X0[0,0]*2 - X0[1,0], X0[1,0]*2 - X0[0,0] + 1 ] ).transpose()

    Xt = X0 - ( dj + tA * P0 ) * epsilon
    Pt = P0 + ( A * Xt - b) * rho

    if Pt [1,0] < 0 : Pt[1,0] = 0

    X0,P0 = Xt,Pt
    iter += 1
    if iter % 100 == 0 :
        print ("iteration",iter, J)

print (iter)
print (Xt)
```

Solution numérique : 1058 itérations, `[0.39982696, 0.30007329]`

Prolongement 2

Non corrigé.

fin correction TD ?? □