INSA Toulouse Année 2021-2022
MOdIA

Optimisation

Cet examen a des questions numériques et théoriques. On répond aux questions numériques en faisant une fonction qui doit compiler. Toutes les fonctions doivent se trouver dans le même fichier. Toutes les questions numériques seront précédées de la balise **NUM :**

L'objectif de cet examen est de travailler sur des données statistiques que nous nommerons "brozek" et d'expliquer "brozek" en fonction du reste des variables. Il se trouve que "brozek" a une signification mais je n'ai pas compris car je n'ai pas validé mon examen de stats.

L'objectif est d'essayer d'implémenter un algorithme de minimisation de

$$\min_{\|x\|_1 \leq c} \frac{1}{2}\|Ax - b\|_2^2$$

Pour recupérer les matrices A et b, on tapera le code suivant, après avoir téléchargé le fichier `data.npy`.

```
import numpy as np
def GET :
  toto=np.load('data.npy')
  b=toto[:,0]
  A=np.copy(toto)
  A[:,0]=1.
  return A,b
```

On vérifie que A a bien 14 variables, la première correspond à l'"intercept". Vous avez plutôt intérêt à savoir ce que c'est car sinon, c'est -2 points pour l'examen de Mme Maugis.

1. **NUM :** Coder la fonction `GET` définie ci-dessus dans votre fichier python. Vous ferez évidemment des imports classiques de librairies. Je trouve

   ```
   A,b=GET()
   print(np.linalg.norm(A),np.linalg.norm(b))
   #4438.6228232977855 324.75062740509065
   ```

   (Il n'y a pas de points pour cette question, c'est juste une question éliminatoire.)
2. On se concentre sur le problème sans contrainte $\min_x f(x)$ où $f(x) = \frac{1}{2}\|Ax - b\|_2^2$ (ou encore $c = +\infty$). On suppose que la matrice A^TA est inversible
   (a) Calculez le gradient et la Hessienne de f
   (b) **NUM :** Créez trois fonctions `def f(x) :`, `def gradf(x) :`, `def Hessf(x) :` qui vous donnent le respectivement la valeur de f, de son gradient et de sa Hessienne au point x. Je trouve

```
A,b=GET()
n=14
np.random.seed(10)
y=np.random.randn(n)
print('Testf',f(y).shape,np.linalg.norm(f(y)))
print('TestGradf',gradf(y).shape,np.linalg.norm(gradf(y)))
print('TestHessf',Hessf(y).shape,np.linalg.norm(Hessf(y)))
#Testf () 7733636.11831896
#TestGradf (14,) 17319676.346308164
#TestHessf (14, 14) 19592934.364495266
```

(c) Montrez que le problème de minimisation de f sans contrainte admet une solution unique si A^TA est inversible. Donnez cette solution.

(d) **NUM :** Créez une fonction `def least_square(A,b) :` qui donne la solution de minimisation sans contrainte. On pourra utiliser la fonction `np.linalg.solve`. Je trouve (en une ligne)

```
A,b=GET()
x=least_square(A,b)
print('LS',np.linalg.norm(x),f(x))
#LS 15.40651626417451 1892.5678738066536
```

Au passage comment s'appelle $f(x)$, ce nombre `1892.5678738066536` ? Je ne vous donne pas de points si vous répondez juste mais je retire 4 points à votre examen de stats si vous répondez faux.

(e) **NUM :** Implémentez un algorithme de gradient pour minimiser f. Vous appellerez cet algorithme `def gradient(x0,step,tol=1.e-6,nitermax=1000) :` où `x0` est le point de départ que vous prendrez nul, `step` est le pas, `tol` est la tolérance et `nitermax` est le nombre maximal d'itérations. Le critère d'arrêt est que la norme du gradient de f est plus petite que `tol`.

(f) Lancez votre algorithme de gradient et essayez de trouver des pas pour lesquels vous convergez. Vous laisserez au moins un test commenté dans votre code avec des explications dessus. Si vous n'arrivez pas à faire converger votre algorithme, ne déprimez pas trop, vous êtes justes incapables de résoudre un problème linéaire :(et passez à la question suivante.

(g) La raison pour laquelle l'algorithme de gradient fonctionne si mal est que la matrice est mal conditionnée. On a une fonction numpy qui calcule le conditionnement, c'est `np.linalg.cond`. Pour mieux conditionner la matrice, on va appliquer un truc de statisticien, on va recentrer toutes les variables (sauf l'intercept, on ne veut pas se faire tuer par Mme Maugis) et on va les réduire (ou les normaliser). Si vous n'avez aucune idée de ce que je viens de dire, pas de problème, il suffit de copier coller le code ci-dessous et cela fera ce qu'il faut. Mais vous aurez -4 points à l'examen de statistique :).

```
def GET2():
    A,b=GET()
    for i in range(1,A.shape[1]) :
        A[:,i]=A[:,i]-A.shape[0]/np.sum(A[:,i])
```

```
    for i in range(A.shape[1]) :
        A[:,i]/=np.linalg.norm(A[:,i])
    return A,b
A,b=GET()
A2,b=GET2()
print(np.linalg.cond(A.T.dot(A),np.linalg.cond(A2.T.dot(A2)))
#318177392.581283 100139.66608469417
```

Vous avez vu comment le conditionnement de `A2.T.dot(A2)` est beaucoup plus petit que celui de `A.T.dot(A)` ? c'est beau hein ? ?

(h) Relancez l'algorithme de gradient avec cette nouvelle matrice. Personnellement, je trouve ce genre de résultat

```
A,b=GET2()
x0=np.zeros(n)
x=gradient(x0,1.e-1,nitermax=4.e5)
print(np.linalg.norm(x-least_square(A,b)))
#0.0200449843931244
```

3. On s'intéresse au cas $c \neq +\infty$, c'est à dire au problème de minimisation sous contrainte $\min_{g(x)\leq 0} f(x)$ avec $g(x) = \|x\|_1 - c$. Montrez que ce problème admet une solution.
4. Comme g est non différentiable, on va la régulariser et on pose

$$g_\varepsilon(x) = \varepsilon \sum_i \sqrt{1 + \left(\frac{x_i}{\varepsilon}\right)^2} - c.$$

Montrez que pour tout x $g_\varepsilon(x)$ tend vers $g(x)$ quand ε tend vers 0.

5. On fixe ε et on s'intéresse à $\min_{g_\varepsilon(x)\leq 0} f(x)$. Montrez que ce problème admet une solution (on pourra montrer que $g_\varepsilon \geq g$). Montrez que les contraintes sont qualifiées et écrire KKT.
6. Soit $\lambda \geq 0$. On remplace le problème de minimisation de la question précédente par

$$\min_x f(x) + \lambda g_\varepsilon(x)$$

Montrez que ce problème admet une solution et que si λ est bien choisi, les points critiques de ce problème vérifient les conditions de KKT du problème précédent.

7. Implémentez deux fonctions `def g(x) :`, `def gradg(x) :`, qui caculent respectivement les valeurs de g_ε et de son gradient. On pourra prendre $\varepsilon = 10^{-3}$
8. Implémentez un algorithme de gradient à pas fixe pour minimiser $f+\lambda g_\varepsilon$. On appellera cet algorithme `def gradient2(x0,lamb=0,step,tol=1.e-6,nitermax=1000) :` La variable `lamb` est la valeur du paramètre λ.
9. Lancez le code suivant

```
res=np.zeros((10,14))
for i in range(10) :
    x2=least_square(A,b)
    tmp=gradient2(x2,1.e-2,lam=i/5.,eps=1.e-3,nitermax=1e5)
```

```
    res[i,:]=tmp
for i in range(14) :
    plt.plot(range(10),res[:,i],label=str(i))
plt.legend()
plt.show()
```

Si vous êtes capable de me dire ce qui se passe, je vous tire mon chapeau, je ne vous donne pas de points mais par contre je vous remets tous les points que je vous ai enlevé dans votre examen de stats.