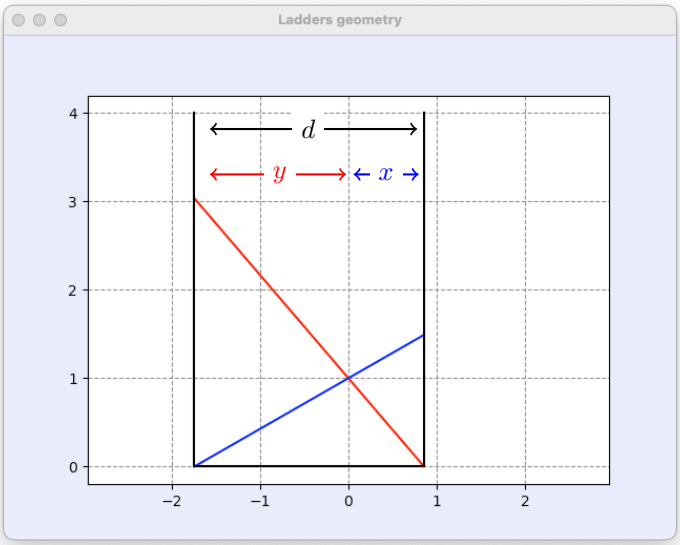
Python 22-23 for dummies
Problème 9
Un problème d'échelles !



Deux échelles de longueurs respectives $a = 3$ et $b = 4$ mètres sont posées contre deux murs verticaux opposés. Les échelles se croisent à une hauteur $c = 1$ mètre du sol. On cherche à déterminer la distance d entre les deux murs.

Concrètement, le problème consiste à déterminer x et y définis sur la figure. Il s'agira donc de résoudre un système de deux équations à deux inconnues. Les deux équations à résoudre peuvent s'écrire sous la forme d'expressions polynômiales faisant intervenir x et y de manière conjointe. Comme le titulaire du cours déteste la trigonométrie, il a trouvé la solution en n'utilisant aucune fonction trigonométrique :-)

Pour résoudre ces équations, on se propose d'implémenter la méthode de Newton-Raphson :

1. Tout d'abord, il s'agit de trouver le système des deux équations non-linéaires permettant d'obtenir x et y à partir des données a, b, c. Vous ne ferez usage d'aucune fonction trigonométrique ou mathématique particulière, car vous ne pourrez pas utiliser `numpy` ou `math` dans votre implémentation. Un joli problème de géométrie pour l'examen d'entrée au passage !

2. Ensuite, il faut écrire une fonction qui effectue une itération de Newton-Raphson pour le système trouvé pour un jeu quelconque de paramètres.

   `xnew = laddersIterate(geometry,x)`.

   - L'argument `geometry` est une liste avec les trois paramètres (a, b, c) du problème.
   - L'argument `x` est une liste avec la valeur courante pour les deux inconnues (x, y)
   - La fonction renverra la nouvelle valeur `xnew` après avoir effectué l'itération.

3. Finalement, il s'agit d'écrire une fonction qui implémente l'entièreté de la méthode de Newton-Raphson pour un jeu quelconque de paramètres. En particulier, il s'agit de déterminer un candidat initial : ce qui n'est pas particulièrement simple... C'est la partie délicate du devoir !
   Pour le reste, on peut évidemment utiliser la première fonction que vous avez écrite et ensuite s'inspirer vraiment très largement du programme de test fourni : ce tout petit plagiat est évidemment totalement admis et autorisé (pour une fois :-)

   `xsol = laddersSolve(geometry,tol,nmax)`.

   - Les arguments `tol` et `nmax` contiennent la tolérance requise sur la norme de la solution et le nombre maximum d'itérations à effectuer.
   - La fonction renverra la solution trouvée `xsol`. Si le schéma n'a pas convergé ou qu'aucune solution ne peut être trouvée, votre fonction renverra simplement `(-1,-1)`.

4. Attention : il n'est -pas- permis d'utiliser les fonctions de `numpy` et `math` dans votre devoir : donc, pas de cosinus, sinus, tangente et autres fonctions mathématiques. Vous avez uniquement droit aux deux fonctions suivantes `scipy.linalg.norm` et `scipy.linalg.solve` : cela vous permet de calculer la norme d'un vecteur et de résoudre le système linéaire : ce qui est normalement requis dans votre implémentation : on pourrait résoudre manuellement le système de deux équations, mais c'est définitivement trop fastidieux à faire :-)

5. Comme d'habitude, un programme `laddersTest.py` vous est fourni pour tester votre fonction.

```
from numpy import *
from scipy.linalg import norm

geometry = [4,3,1]
print(" ========= my Newton-Raphson scheme with your proposed step :-)")
x = array([1.0,1.5]); tol = 10e-12; nmax = 50
n = 0; delta = tol+1
while (norm(delta) > tol and n < nmax):
  xold = x
  x = laddersIterate(geometry,xold)
  delta = x-xold; n = n+1
  print(" Estimated error %9.2e at iteration %d : " % (norm(delta),n),x)
print(" Computed distance is : %13.6f " % sum(x))

print(" ========= your full computation :-)")
sol = laddersSolve(geometry,1e-14,50)
print(" Computed distance is : %13.6f " % sum(sol))

import matplotlib.pyplot as plt
plt.figure("Ladders geometry")
a = geometry[0]; b = geometry[1]; c = geometry[2]; ab = max(a,b)
x = sol[0]; y = sol[1]; d = x + y
hx = sqrt(b*b - d*d); hy = sqrt(a*a - d*d)
plt.plot([-x,y],[hx,0],'-r')
plt.plot([-x,y],[0,hy],'-b')
plt.plot([-x,-x,y,y],[ab,0,0,ab],'k')
plt.axis('equal')
plt.show()
```

6. Evidemment, sur le serveur, nous allons considérer d'autres géométries que celles de ce programme : il est donc nécessaire d'écrire le code de manière tout-à-fait générale :-)

7. Votre fonction (avec les éventuelles sous-fonctions que vous auriez créées) sera soumise via le site web du cours.

8. Attention : il ne faut pas recopier le programme de test `main` dans votre soumission : uniquement les deux fonctions que vous avez écrites. Vérifier bien que votre programme fonctionne correctement sur le serveur et pas uniquement sur votre ordinateur : aucun recours ne sera valable si le devoir n'est pas exécuté correctement sur le serveur. Pour rappel, toutes vos soumissions seront systématiquement analysées par un logiciel anti-plagiat. Faites vraiment votre programme seul... en vous inspirant uniquement des programmes fournis par l'enseignant :-)