Positionnement par GPS

Pour calculer sa position (x,y,z), un récepteur GPS reçoit des messages en provenance de 4 satellites indiquant les positions (x_i,y_i,z_i) de chaque satellite ainsi que l'instant t_i où le message a été émis depuis le satellite. Les satellites sont désignés par leur numéro $0,1,2,3$ et on dispose les informations reçues dans une matrice 4×4 :

$$M=\begin{bmatrix} x_0 & y_0 & z_0 & t_0\\ x_1 & y_1 & z_1 & t_1\\ x_2 & y_2 & z_2 & t_2\\ x_3 & y_3 & z_3 & t_3 \end{bmatrix};\qquad \text{par exemple : } M=\begin{bmatrix} 0.94 & 2.05 & 0.00 & 23.0\\ 1.95 & 0.00 & 2.06 & 2.1\\ 1.02 & 0.88 & 1.14 & 30.6\\ 2.03 & 0.97 & 0.00 & 23.2 \end{bmatrix}$$

On fera les hypothèses simplificatrices suivantes :

(1) La terre est assimilée à une sphère de rayon $R=6378$ km ;
(2) Les distances sont exprimées en prenant comme unité de mesure le rayon de la terre et les temps sont donnés en millisecondes ;
(3) Le récepteur est situé au niveau de la mer ;
(4) Les signaux émis par les satellites se propagent à la vitesse $c=3\cdot 10^5$ km$\cdot$s^{-1} (vitesse de la lumière) ;
(5) Les signaux émis par les satellites sont reçus en même temps par le récepteur, à l'instant t ;
(6) On ne tient pas compte des différentes erreurs de mesure nécessairement commises sur les positions et les temps.

Le but est de déterminer la position (x,y,z) du récepteur à partir des informations contenues dans la matrice M.
On utilise les définitions suivantes :

```
import numpy as np
from scipy.optimize import fsolve
from numpy.linalg import solve
Mex=np.array([[0.94,2.05,0.00,23.0],[1.95,0.00,2.06,2.1],
              [1.02,0.88,1.14,30.6],[2.03,0.97,0.00,23.2]])
```

On pourra utiliser les deux fonctions suivantes :

- Si **A** est une matrice carrée inversible de taille n et **b** est un vecteur de taille n, alors **solve(A,b)** détermine la solution du système linéaire $Ax=b$;

- Si **f** est une fonction possédant une racine réelle et **a** est un nombre, alors **fsolve(f,a)** détermine une approximation de la racine de f la plus proche de a.

Exemples.

```
A=np.matrix([[1,2],[2,1]])
b=np.array([1,1])
print solve(A,b)
```

```
[ 0.33333333 0.33333333]
```

□

```
print fsolve(lambda x:x**2-2,1)
```

```
[ 1.41421356]
```

1 Quelques calculs à partir de la matrice M

Question 1 : Pour obtenir le coefficient de la matrice **M** représentant y_2, on écrit :

①□ **M[3,2]** ②□ **M[2,1]** ③□ **M[1,2]** ④□ **M[2,3]**

Question 2 : Pour extraire le vecteur $\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$ de **M**, on écrit :

①□ **M[0,:]** ②□ **M[1,:]** ③□ **M[:,0]** ④□ **M[:,1]**

Question 3 : Pour former le vecteur $\begin{bmatrix} x_1^2+y_1^2+z_1^2-c^2t_1^2 \\ x_2^2+y_2^2+z_2^2-c^2t_2^2 \\ x_3^2+y_3^2+z_3^2-c^2t_3^2 \end{bmatrix}$, on écrit :

①□ **M[1:3,0]******2+M[1:3,1]******2+M[1:3,2]******2-c******2*M[1:3,3]******2**
②□ **M[0,1:]******2+M[1,1:]******2+M[2,1:]******2-c******2*M[3,1:]******2**
③□ **M[0,1:3]******2+M[1,1:3]******2+M[2,1:3]******2-c******2*M[3,1:3]******2**
④□ **M[1:,0]******2+M[1:,1]******2+M[1:,2]******2-c******2*M[1:,3]******2**

2 Distances entre les satellites et le récepteur

Question 4 : Dans le système d'unités choisi (unité de longueur : rayon terrestre, unité de temps : milliseconde), la valeur numérique de c est :

①□ $4.7\cdot10^{-2}$ ②□ $4.7\cdot10^{4}$ ③□ $1.9\cdot10^{6}$ ④□ $1.9\cdot10^{12}$

Question 5 : On considère les informations transmises par le satellite numéro i. On note (x,y,z) la position du récepteur et t l'instant où il reçoit les informations.

(a) Exprimer en fonction de (x,y,z) et (x_i,y_i,z_i) la distance entre le récepteur et le satellite :

(b) Exprimer cette même distance en fonction de c, t et t_i :

(c) En égalant ces deux quantités et en simplifiant, on obtient la relation :

① ☐ $x_i x + y_i y + z_i z = \frac{1}{2}\left(1 - x_i^2 - y_i^2 - z_i^2 - c^2(t-t_i)^2\right)$

② ☐ $x_i x + y_i y + z_i z = \frac{1}{2}\left(-1 - x_i^2 - y_i^2 - z_i^2 + c^2(t-t_i)^2\right)$

③ ☐ $x_i x + y_i y + z_i z = \frac{1}{2}\left(1 + x_i^2 + y_i^2 + z_i^2 + c^2(t-t_i)^2\right)$

④ ☐ $x_i x + y_i y + z_i z = \frac{1}{2}\left(1 + x_i^2 + y_i^2 + z_i^2 - c^2(t-t_i)^2\right)$

Cette équation sera notée (E_i) dans la suite.

Question 6 : Compléter la fonction ci-dessous afin qu'elle calcule la différence entre les deux distances calculées pour un récepteur en position (x, y, z) recevant l'information du satellite i au temps t :

```
def difference(M,i,x,y,z,t):
```

3 Le système d'équations

Question 7 : On considère le système linéaire obtenu en prenant les équations (E_1), (E_2), (E_3) dans cet ordre. On note matriciellement ce système sous la forme :

$$A \begin{bmatrix} x \\ y \\ z \end{bmatrix} = b$$

Expliciter la matrice A et le vecteur b :

Question 8 : Écrire une fonction **`matrice_systeme(M,t)`** qui renvoie la matrice A à partir des informations contenues dans M et de l'instant t de réception. Écrire de même la fonction **`vecteur_systeme(M,t)`** qui renvoie le vecteur b :

```
def matrice_systeme(M,t):
```

```
def vecteur_systeme(M,t):
```

Question 9 : En utilisant les fonctions précédentes, écrire une fonction **`position(M,t)`** qui détermine la position (x, y, z) du récepteur :

```
def position(M,t):
```

4 Une situation plus réaliste

Dans cette partie, on ne suppose plus que t est connu (il n'est en effet pas réaliste de supposer que le récepteur GPS possède une horloge parfaitement synchronisée avec celle des satellites) et on va également le déterminer à partir des informations des satellites. On va pour cela utiliser les informations du satellite 0.

Question 10 : On considère la fonction f qui à une valeur de t associe la quantité :

$$((x-x_0)^2+(y-y_0)^2+(z-z_0)^2-c^2(t-t_0)^2)^2+(x^2+y^2+z^2-1)^2$$

où (x, y, z) est la position obtenue avec **`position(M,t)`**. Programmer ce calcul sous la forme d'une fonction **`f(M,t)`** :

```
def f(M,t):
```

Question 11 : Écrire une fonction **position_async(M)** qui détermine l'instant t où les informations ont été reçues par le récepteur GPS qui calcule sa position.

```
def position_async(M):
```

Corrections

Q. 1 : Le coefficient y_2 est situé en ligne 3 et colonne 2 (notations du cours de mathématiques) ou en ligne 2 et colonne 1 (conventions PYTHON), donc y_2 correspond à `M[2,1]` : réponse 2.

Q. 2 : Il faut extraire la colonne numéro 1 (conventions PYTHON) de la matrice, on écrit donc `M[:,1]` : réponse 4.

Q. 3 : On ne considère que les trois dernières lignes de M, on aura donc des expressions de la forme `M[1:,???]`. Il faut faire la somme de la première colonne de M au carré (`M[1:,0]**2`) avec la deuxième (`M[1:,1]**2`) et la troisième (`M[1:,2]**2`) et retrancher la quatrième au carré multipliée par c^2 (`c**2*M[1:,3]**2`). On obtient donc :

```
M[1:,0]+M[1:,1]**2+M[1:,2]**2-c**2*M[1:,3]**2
```

C'est la réponse 4.

Q. 4 :

```
c=3e5/6378/1000
print "%e" % c # Afficher c en notation scientifique
```

```
4.703669e-02
```

C'est la réponse 1.

Q. 5 : La distance entre les points (x,y,z) et (x_i,y_i,z_i) est :

$$\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}$$

Le message a été émis à l'instant t_i, reçu à l'instant t et voyage à la vitesse c, il a donc parcouru une distance égale à $c(t-t_i)$. En égalant ces deux quantités :

$$(x-x_i)^2+(y-y_i)^2+(z-z_i)^2=c^2(t-t_i)^2$$

En développant les carrés :

$$x^2+y^2+z^2-2x_ix-2y_iy-2z_iz=c^2(t-t_i)^2-(x_i^1+y_i^2+z_i^2)$$

Par hypothèse $x^2+y^2+z^2=1$, donc :

$$x_ix+y_iy+z_iz=\frac{1-c^2(t-t_i)^2+(x_i^1+y_i^2+z_i^2)}{2}$$

C'est la réponse 4.

Q. 6 :

```
c=0.047
def difference(M,i,x,y,z,t):
    d1=(x-M[i,0])**2+(y-M[i,1])**2+(z-M[i,2])**2
    d2=c**2*(t-M[i,3])**2
    return d1-d2
```

Q. 7 : On obtient $A = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix}$ et $b = \frac{1}{2}\begin{bmatrix} 1-c^2(t-t_1)^2+(x_1^1+y_1^2+z_1^2) \\ 1-c^2(t-t_2)^2+(x_2^1+y_2^2+z_2^2) \\ 1-c^2(t-t_3)^2+(x_3^1+y_3^2+z_3^2) \end{bmatrix}$.

Q. 8 :

```
def matrice_systeme(M,t):
    return M[1:,0:3]
def vecteur_systeme(M,t):
    return (1+M[1:,0]**2+M[1:,1]**2+M[1:,2]**2-c**2*(t-M[1:,3])**2)/2
```

Q. 9 :

```
def position(M,t):
    A=matrice_systeme(M,t)
    b=vecteur_systeme(M,t)
    return solve(A,b)
print position(Mex,52.5)
```

```
[ 0.6904638 0.70211987 0.18011595]
```

Q. 10 :

```
def f(M,t):
    v=position(M,t)
    x=v[0]
    y=v[1]
    z=v[2]
    return difference(M,0,x,y,z,t)**2+(x**2+y**2+z**2-1)**2
```

Q. 11 : On résout l'équation $f(t) = 0$. On utilise pour cela la fonction **fsolve** en prenant comme point de départ la plus grande valeur parmi $t_0, \ldots, t_3$.

```
def position_async(M):
    t=fsolve(lambda t:f(M,t),M[:,3].max())
    return position(M,t)
print position_async(Mex)
```

```
[ 0.69273377 0.70136814 0.18120751]
```