

## Proposition de corrigé

Concours : Concours Commun Polytechniques

Année : 2018

Filière : PSI

Épreuve : Informatique

Ceci est une proposition de corrigé des concours de CPGE, réalisée bénévolement par des enseignants de Sciences Industrielles de l'Ingénieur et d'Informatique, membres de l'[UPSTI](http://www.upsti.fr) (Union des Professeurs de Sciences et Techniques Industrielles), et publiée sur le site de l'association :

<https://www.upsti.fr/espace-etudiants/annales-de-concours>

### A l'attention des étudiants

Ce document vous apportera des éléments de corrections pour le sujet traité, mais n'est ni un corrigé officiel du concours, ni un corrigé détaillé ou exhaustif de l'épreuve en question.

L'UPSTI ne répondra pas directement aux questions que peuvent soulever ces corrigés : nous vous invitons à vous rapprocher de vos enseignants si vous souhaitez des compléments d'information, et à vous adresser à eux pour nous faire remonter vos éventuelles remarques.

### Licence et Copyright

Toute représentation ou reproduction (même partielle) de ce document faite sans l'accord de l'UPSTI est **interdite**. Seuls le téléchargement et la copie privée à usage personnel sont autorisés (protection au titre des [droits d'auteur](#)).

En cas de doute, n'hésitez pas à nous contacter à : [corrigesconcours@upsti.fr](mailto:corrigesconcours@upsti.fr).

### Informez-vous !

Retrouvez plus d'information sur les [Sciences de l'Ingénieur](#), l'[orientation](#), les [Grandes Ecoles](#) ainsi que sur les [Olympiades de Sciences de l'Ingénieur](#) et sur les [Sciences de l'Ingénieur au Féminin](#) sur notre site : [www.upsti.fr](http://www.upsti.fr)

L'équipe UPSTI

# Correction sujet d'informatique CCP PSI 2018

$$Q1 - \frac{dY(t)}{dt} = F(Y(t), t) \text{ avec } Y(t) = \begin{pmatrix} u(t) \\ v(t) \\ x(t) \\ y(t) \end{pmatrix} \text{ et } F(Y(t), t) = \begin{pmatrix} -\frac{1}{2m}(\pi R^2 \rho_{air} C_1 V^2 \cos(\alpha) + 2\rho_{air} R^3 C_2 V \Omega \sin(\alpha)) \\ -g - \frac{1}{2m}(\pi R^2 \rho_{air} C_1 V^2 \sin(\alpha) - 2\rho_{air} R^3 C_2 V \Omega \cos(\alpha)) \\ u(t) \\ v(t) \end{pmatrix}$$

Q2 -

python

```
def euler(T,N,F,Y0):
    Y=zeros((N,len(Y0))) %Il y a une erreur dans l'annonce
    t=arange(0,T,T/N)
    Y[0]=Y0
    for i in range(1,N):
        Y[i]=Y[i-1]+T/N*F(Y[i-1],t[i-1])
    return(t,Y)
```

Remarque : il semblerait qu'il y ait des erreurs car  $\alpha$  dépend du temps.

Q3 -

- len(image1) renvoie la valeur 1024.
- len(image1[12][244]) renvoie la valeur 3.
- len(image1[12][244][2]) est un entier (compris entre 0 et 255).

Q4 - Pour chaque pixel, chaque couleur est codée sur 8 bits c'est à dire un octet. Donc chaque pixel demande 3 octets or il y a  $768 \cdot 1024 = 786432$  pixels. Cela donne donc 2359296 octets soit environ 2,25 Mo.

Q5 - Un point correspond à  $1000 \cdot 10 = 10000$  images donc  $2,25 \cdot 10^4$  Mo soit quasiment 22 Go. Il y a 10 caméras donc cela fait au final 220 Go. Pour un set cela représente en moyenne 21,5 To. Il faut donc de gros disques durs (mémoire morte) pour sauvegarder les données d'un match.

Q6 - La mémoire vive, à l'inverse de la mémoire morte est une mémoire volatile qui se réinitialise lors d'une mise hors tension. L'inconvénient principal de la mémoire morte par rapport à la mémoire vive est une vitesse d'écriture et de lecture plus faible.

Q7 -

python

```
def detection(image1,image2):
    tableau=zeros((len(image1),len(image1[0]))) %initialisation du tableau
    for i in range(len(image1)) : %parcours des lignes
        for j in range(len(image1[0])): %parcours des colonnes
            delta=0 %initialisation de la valeur de delta
            for k in range len(image1[0][0]) %parcours des couleurs
                delta+=(image1[i][j][k]-image2[i][j][k])**2 %Calcul du delta du pixel (i,j)
            if delta>255 :
                delta=255
            tableau[i][j]=delta
    return(tableau)
image_gray = detection(im1,im2)
```

Q8 -

python

```
def filtre(image,seuil):
    tableau=zeros((len(image),len(image[0]))) %initialisation du tableau
    for i in range(len(image)) : %parcours des lignes
        for j in range(len(image[0])): %parcours des colonnes
            if image[i][j]>seuil: %teste la valeur de delta
                tableau[i][j]=1
    return(tableau)
```

Q9 - La seule solution possible est le test 3 car il est le seul à tester les 8 pixels autour du pixel considéré.

## Q10 -

- La ligne 2 sert à ouvrir le dossier du point litigieux
- La ligne 3 sert à parcourir tous les noms des fichiers de ce dossier
- La première boucle FOR sert à trouver le fichier dont le numéro de l'image dans la séquence est le plus élevé
- La seconde boucle FOR sert à obtenir la liste des positions des balles éventuelles dans un set.

## Q11 -

### 1. la fonction *deplacement*

python

```
def deplacement(pos1,pos2):
    dx=pos2[0]-pos1[0] %le type des elements des listes est int (entier)
    dy=pos2[1]-pos1[1]
    return([dx,dy])

Lx=2*deplacement(pos1,pos2)[0]+1
Ly=2*deplacement(pos1,pos2)[1]+1
```

### 2. la fonction *distance\_quad* (remarque : on a considéré que *liste\_balle\_i* était la liste des coordonnées des points possible, une liste de liste donc)

python

```
def distance_quad(xc,yc,liste_balle_i):
    distances=[]
    for i in range(len(liste_balle_i)):
        distances+=[(liste_balle_i[2*i]-xc)**2+(liste_balle_i[2*i+1]-yc)**2]
    return(distances)
```

### 3. la fonction *cherche\_balle* (remarque : l'utilisation de la fonction *distance\_quad* n'est pas forcément le plus simple ici)

python

```
def cherche_balle(xc,yc,Lx,Ly,liste_balle_i):
    for i in range(len(liste_balle_i)): %selection des coordonnees des balles dans le rectangle
        Balles0k = [ ]

    for k in range(len(liste_balle_i)//2):
        xk = liste_balle_i[2*k]
        yk = liste_balle_i[2*k+1]
        if xk >= xc-Lx/2 and xk <= xc+Lx/2 and yk >= yc-Ly/2 and yk <= yc+Ly/2 :
            Balles0k += [xk,yk]

    if Balles0k == [ ]:
        return [None,None]
    else :
        Distances = distance_quad(xc,yc,Balles0k)
        curseur = 0
        DistanceMin = Distances[curseur]
        for k in range(1,len(Distances)):
            if Distances[k] < DistanceMin :
                curseur = k
                DistanceMin = Distances[curseur]
        return [Balles0k[2*curseur],Balles0k[2*curseur+1]]
```

#### 4. La fonction `traj_balle` :

python

```
def traj_balle(seq_balle,vit_init):
    Balles=seq_balle[0] %initialisation de la liste finale
    xc,yc=0,0 %initialisation du centre de la zone
    Lx=0 ; Ly=0 %initialisation des dimensions du rectangle
    ListeVide==0 %Initialisation de l'indicateur de liste vide
    for i in range(1,len(seq_balle)):
        if i==1: %Cas du 1er point
            dx,dy=vit_init[0]*1/1000,vit_init[1]*1/1000
            Lx,Ly=2*dx+1,2*dy+1
        elif ListeVide==0: %si la balle est presente dans la zone
            dx,dy=deplacement(Balles[-2],Balles[-1])[0], deplacement(Balles[-2],Balles[-1])[0]
            Lx,Ly=2*dx+1,2*dy+1
        else: %si la balle n'est pas presente dans la zone
            dx,dy=2*deplacement(Balles[-2],Balles[-1])[0], 2*deplacement(Balles[-2],Balles[-1])[0]
            Lx,Ly=2*(2*dx+1)-1,2*(2*dy+1)-1
            ListeVide=0

        xc,yc=Balles[-1][0]+dx,Balles[-1][1]+dy %Calcul du point central
        Point=cherche_balle(xc,yc,Lx,Ly,seq_balle[i]) %Detection de la balle
        if Point==[None,None]: %si pas de balle, on passe a l'image suivante en l'indiquant
            ListeVide=1
        else:
            Balles+=Point
    return(Balles)
```

**Q12** - L'inconvénient majeur de cet algorithme est sa complexité d'ordre au moins 3. Il n'est en effet pas possible de le calculer car le candidat n'a pas l'algorithme complet. Vu le nombre d'images à traiter cela risque d'être très long.

**Q13** -

python

```
def pos_loc(e,f,x1,x2,y1,y2):
    x1i,y1i,z1i=e/(x1-x2)*x1, e/(x1-x2)*y1, e*f/(x1-x2)
    return([x1i,y1i,z1i])
```

**Q14** -

python

```
def pos_glo(posi,T1,Rx1,Ry1):
    posi=array(posi)
    posi=posi+T1
    posi=Rx1.dot(posi)
    posi=Ry1.dot(posi)
    return([posi[0],posi[1],posi[2]])
```

**Q15** -

python

```
def traj3D(coord_loc,T1,Rx1,Ry1):
    coord_glo=[]
    for i in range(len(coord_loc)):
        coord_glo+=pos_glo(coord_loc[i],T1,Rx1,Ry1)
    return(coord_glo)
```

Q16 -

python

```
def det_impact(coord_glo,eps):
    %Cette fonction renvoie les coordonnees de l'impact sur un aller simple de la balle
    R=0.033
    for i in range(len(coord_glo)): %Pour trouver un y est dans l'intervalle
        if coord_glo[i][1]>=R AND coord_glo[i][1]<=R+eps:
            x_imp,z_imp=coord_glo[i][0], coord_glo[i][2]
            return([x_imp,z_imp])

    yi=coord_glo[0][1]
    for i in range(1,len(coord_glo)): %Cherche le chagement de signe de y
        yim1=yi
        yi=coord_glo[i][1]
        if yim1*yi<0:
            x_imp=(coord_glo[i][0]+coord_glo[i-1][0])/2
            z_imp=(coord_glo[i][2]+coord_glo[i-1][2])/2
            return([x_imp,z_imp])

    return([x_imp,z_imp])
```

Q17 -

python

```
def res_final(impact,L,l):
    x,z=impact[0],impact[1]
    if x<0 OR x>L OR z>0 OR z<-l:
        return('OUT')
    else:
        return('IN')
```

Q18 -

python

```
def vi_traj3D(coor_glo):
    M=array(coor_glo)
    x,y,z=M[:,0],M[:,1],M[:,2]
    gca(projection='3d').plot(x,y,z)
    xlabel('x (m)')
    ylabel('y (m)')
    zlabel('z (m)')
    title('Trajectoire de la balle')
    show()
```

Q19 - Une clé est un attribut ou un ensemble d'attributs d'un type d'entités ou d'association dont la connaissance d'une valeur identifie sans ambiguïté une occurrence unique de cette classe. Une clé primaire est une des clés possibles choisie pour servir de moyen d'identification d'une occurrence.

Q20 -

SQL

```
SELECT id FROM MATCHS WHERE joueur1="Federer" OR joueur2="Federer" ;
```

Q21 -

SQL

```
SELECT MAX(mid) FROM POINTS WHERE mid="4" ;
```

Q22 -

SQL

```
SELECT fichier FROM POINTS JOIN MATCHS ON mid=MATCHS.id WHERE nom="Federer-Murray" ;
```