4 Mouvement circulaire

Programme à compléter

Fichiers Python
Programmes à compléter
Fiche d'accompagnement
hatier-clic.fr/psi293

L'objectif de cette activité est d'exploiter une chronophotographie d'un mouvement circulaire, pour étudier l'accélération du système dans ce type de mouvement.

Prérequis théoriques

- Accélération, calcul approché.
- Repère de Frenet.
- Accélération, expression théorique dans le repère de Frenet.

Apports théoriques supplémentaires de l'activité

- Caractéristiques du vecteur accélération dans le cas d'un mouvement circulaire (uniforme ou non).

Question 1

La question **1a** consiste à caractériser les deux phases du mouvement de la cabine de la centrifugeuse, grâce au programme python *centrifugeuse.py* qui trace ses positions lors du premier tour. Il faut, pour répondre, connaître la définition d'un mouvement circulaire, un mouvement uniforme, un mouvement accéléré.
Dans la deuxième partie de la question, on identifie dans quelle(s) phase(s) du mouvement l'accélération du système est non-nulle.
La question **1b** implique la détermination des expressions des coordonnées des vecteurs vitesse et accélération à partir des formules vectorielles des calculs approchés de ces vecteurs afin de compléter le programme python *centrifugeuse.py*.
Le programme complété calcule les coordonnées du vecteur vitesse approché et du vecteur accélération approché et affiche ces deux vecteurs sur la chronophotographie du système. Le tracé du vecteur accélération permet de confirmer ou non la réponse donnée à la question **1a**.

Question 2

La question **2** consiste à étudier l'accélération tangentielle a_t au cours du mouvement du système grâce au programme python *centrifugeuse_frenet.py*. Ce programme calcule les coordonnées, dans le repère de Frenet, du vecteur accélération approché. Il trace également l'évolution de ses deux coordonnées (a_n et a_t) au cours du temps.
Cette question nécessite de connaître l'expression du vecteur accélération dans le repère de Frenet (question **1b**).
Dans la question **2a**, on justifie le signe de la composante tangentielle a_t du vecteur accélération. Il faut pour cela prendre en compte les différentes phases du mouvement du système évoquées à la question **1a**.
Puis on justifie, dans la question **2b**, la valeur finale de la composante tangentielle a_t du vecteur accélération en connaissant la nature du mouvement du système pendant la deuxième phase du mouvement.
Ces deux questions permettent ensuite de prédire, dans la question **2c**, l'évolution de cette même composante tangentielle de l'accélération a_t lors du ralentissement du système.

Enfin on mesure, dans la question **2d**, l'accélération atteinte par le système en mouvement.
Il faut pour cela savoir que l'accélération est uniquement normale (a_n) dans le cas d'un mouvement uniforme. Ainsi la lecture graphique de la valeur de l'accélération normale a_n donne la valeur de l'accélération du système lors de cette phase du mouvement.
Cette valeur est ensuite comparée avec la valeur attendue, égale à $4g$ (g = 9,81 m·s^{-2}).

Question 3

La question **3a** consiste à déterminer les expressions numériques théoriques des coordonnées du vecteur accélération dans le repère de Frenet (a_n et a_t) dans les deux phases du mouvement étudié. Ce travail nécessite d'utiliser l'expression théorique du vecteur accélération dans le repère de Frenet **(doc. 2)** et les caractéristiques du mouvement dans ces deux phases, tels que le rayon de la trajectoire et l'évolution de la valeur de la vitesse **(doc. 1)**.
La deuxième partie de cette question implique des modifications du programme python *centrifugeuse_frenet.py* pour y intégrer les expressions numériques théoriques obtenues. Le programme complété calcule les valeurs théoriques des coordonnées de l'accélération (a_n et a_t).
On peut ainsi comparer les valeurs calculées à partir de la chronophotographie et les valeurs théoriques obtenues.
À la question **3b**, les modifications du programme permettent d'utiliser de nouvelles positions du même mouvement, obtenue en divisant par deux la durée Δt entre deux positions successives enregistrées.
Dans la deuxième partie de cette question, on peut alors comparer la précision des coordonnées de l'accélération (a_n et a_t) calculées à partir des deux chronophotographies du même mouvement. Les coordonnées calculées sont pour cela comparer aux valeurs théoriques.

Programme à compléter : *centrifugeuse.py*

```
1  #importation des modules
2  import numpy as np
3  import matplotlib.pylab as plt
4  import csv
5  import sys
6
7  #initialisation du pointage
8  table = []
9  #initialisation du temps en seconde
10 temps = []
11 #initialisation des abscisses
12 x = []
13 #initialisation des ordonnées
14 y = []
15
16 with open('pointage_centrifugeuse.csv', newline='') as csvfile:
17     spamreader = csv.reader(csvfile, delimiter=';', quotechar='|')
18     for row in spamreader:
19         table.append(row)
20
21 #taille du tableau importé
22 N = len(table)
23 M = len(table[0])
24
25 #initialisation des constantes du problème
26 R = 7.00          #Rayon de la centrifugeuse en m
27 k = 6.26          #constante k en m.s-2
28 v0 = 16.6         #constante v0 en m.s-1
29 Dt = 0.200        #pas de temps en s
30
31 #Initialisation des vitesses
32 vx = np.zeros(N-2)        #vitesse calculée selon l'axe des abscisses
33 vy = np.zeros(N-2)        #vitesse calculée selon l'axe des ordonnées
34
35
36 #Initialisation des accélérations
37 ax = np.zeros(N-2)        #accélération calculée selon l'axe des abscisses
38 ay = np.zeros(N-2)        #accélération calculée selon l'axe des ordonnées
39
40
41 #Vérification de la structure de ce tableau
   (3 colonnes ont été exportées : temps, X et Y)
42 if M != 3 :
43     print("Problème dans la réalisation de votre pointage.
   Reprenez le travail et exportez uniquement deux courbes
   (Mouvement_X et Mouvement_Y)")
44     sys.exit()
45
46 #importation des coordonnées X, Y et du temps
47 for i in range(1,N):
48     #Génération des listes, avec transformation des données
49     temps.append(float(table[i][0].replace(',','.')))
50     x.append(float(table[i][1].replace(',','.')))
51     y.append(float(table[i][2].replace(',','.')))
52
53 #Création d'une fonction permettant de tracer les vecteurs
54 def trace_vect(x,y,Vectx,Vecty,titre,couleur,position):
55     q = plt.quiver(x,y,Vectx,Vecty,color = couleur,width=0.003)
56     plt.quiverkey(q, X=0.5, Y=position, U=10,label=titre, labelpos='E', color=couleur)
```

Modules

Le module `numpy` permet des opérations sur les listes.
Le module `matplotlib` fournit les fonctionnalités graphiques.
Le module `csv` permet l'importation des données contenues dans le fichier *.csv.
Le module `sys` permet le contrôle de l'importation de ces données.

Création des listes

On créée des listes vides qui seront remplies par la suite.

Importation des données

Importation des données contenues dans le fichier `pointage_centrifugeuse.csv` dans la liste `table`.

Initialisation des constantes du problème

On définit les constantes du problème.

Les listes vitesse et accélération

Les listes vitesse (`vx` et `vy`) et accélération (`ax` et `ay`) sont initialement remplies grâce à la fonction `np.zeros` avec autant de zéros qu'il y aura de valeurs calculées.

L'initialisation des listes dépends de la méthode de construction de la liste dans la boucle for (lignes 64 à 72).

> Remarque p. 7

Vérification de la structure des données importées

On vérifie avant de débuter les calculs que la structure de la liste `table` est celle attendue.
Le test `!=` signifie « différent de ».

Remplissage des listes prédéfinies

On remplit les listes `temps`, `x` et `y` avec les données importées et contenues dans la liste `table`.

> Remarque p. 7

Définition d'une fonction python

On définit grâce à la commande `def` la fonction `trace_vect` permettant de tracer un vecteur (grâce à la commande `plt.quiver`).

```
57
58  ############################################################
59  ###         DEBUT DE LA PARTIE A MODIFIER                ###
60  ############################################################
61
62  #Calculs approchés des coordonnées des vitesses
63  for i in range(1,N-2):
64      vx[i] = 0
65      vy[i] = 0
66
67  #Calculs approchés des coordonnées des accélérations
68  for i in range(2,N-3):
69      ax[i] = 0
70      ay[i] = 0
71
72  ############################################################
73  ###         FIN DE LA PARTIE A MODIFIER                  ###
74  ############################################################
75
76
77  #imposer la taille de la zone de travail
78  plt.figure(1, figsize=(6, 6))
79
80
81  if ay[N-4]!= 0 :
82      #tracer les vecteurs vitesses calculées
83      trace_vect(x,y,vx,vy,"Vecteur vitesse calculée",'g',1.09)
84
85      #tracer les vecteurs accélérations calculées
86      trace_vect(x,y,ax,ay,"Vecteur accélération calculée",'r',1.06)
87
88
89  #tracer les positions
90  plt.plot(x,y,"r.",label="Positions de la cabine")
91
92  #tracer le centre de rotation
93  plt.plot(0,0,"bo")
94
95  #tracer les rayons
96  for i in range(0,N-1):
97      plt.plot([0, x[i]], [0, y[i]], 'k--', lw=0.2)
98
99  #position du bloc légende
100 plt.legend(loc='upper right')
101
102 #étiquettes des axes
103 plt.xlabel(r'$ x $'+' (m)')
104 plt.ylabel(r'$ y $'+' (m)')
105
106 #Titre du graphique
107 # plt.title('Positions et accélération de la cabine',loc='left')
108 plt.show()
```

À modifier : calculs de la vitesse et de l'accélération

On écrit ici les expressions pour calculer les coordonnées de la vitesse (`vx` et `vy`) et de l'accélération (`ax` et `ay`) déterminées à la question **1b**.
Avant modification, le programme calcule des coordonnées nulles en tout point.

Taille de la fenêtre graphique

Imposer cette taille (exprimé en pouces) permet d'avoir un repère orthonormé (même échelle en abscisse qu'en ordonnée). Les points sont ainsi répartis sur un cercle. Sans cette instruction, la fenêtre graphique s'adapterait à l'écran (échelle différente en abscisse et en ordonnée). Les points tracés seraient alors répartis sur une ellipse.

Tracés des vecteurs

On utilise la fonction `trace_vect` définie précédemment (ligne 54) pour tracer le vecteur vitesse et le vecteur accélération.

Tracés des positions et de l'origine du cercle

On trace les positions du système.
On trace aussi un point au centre

Tracés des rayons

Pour clarifier le graphique obtenu, on trace, pour chaque point un segment reliant le point à l'origine.

À modifier : titre du graphique

On active cette ligne (en supprimant le symbole # en début de ligne) pour afficher le titre du graphique à la question **1b** lorsque les vecteurs vitesse et accélération sont tracés.

Programme à compléter : *centrifugeuse_frenet.py*

```
1  #importation des modules
2  import numpy as np
3  import matplotlib.pylab as plt
4  import csv
5  import sys
6
7  #initialisation du pointage
8  table = []
9  #initialisation du temps en seconde
10 temps = []
11 #initialisation des abscisses
12 x = []
13 #initialisation des ordonnées
14 y = []
15
16 ############################################################
17 ###              IMPORTATION DES DONNES                  ###
18 ############################################################
19 #Le fichier csv d'où l'importation des valeurs est effectuée
20 fichier_importe = 'pointage_centrifugeuse.csv'
21 #Ne pas modifier les 4 lignes suivantes
22 with open(fichier_importe, newline='') as csvfile:
23     spamreader = csv.reader(csvfile, delimiter=';', quotechar='|')
24     for row in spamreader:
25         table.append(row)
26
27 Dt = 0.200       #pas de temps en s
28 ############################################################
29 ###                                                      ###
30 ############################################################
31
32 #taille du tableau importé
33 N = len(table)
34 M = len(table[0])
35
36 #initialisation des constantes du problème
37 R = 7.00         #Rayon de la centrifugeuse en m
38 k = 6.26         #constante k en m.s-2
39 v0 = 16.6        #constante v0 en m.s-1
40
41 #Initialisation des vitesses
42 vx = np.zeros(N-1)        #vitesse calculée selon l'axe des abscisses
43 vy = np.zeros(N-1)        #vitesse calculée selon l'axe des ordonnées
44
45 #Initialisation des accélérations
46 ax = np.zeros(N-1)        #accélération calculée selon l'axe des abscisses
47 ay = np.zeros(N-1)        #accélération calculée selon l'axe des ordonnées
48 at = np.zeros(N-1)       #accélération selon le vecteur unitaire tangent
49 an = np.zeros(N-1)       #accélération selon le vecteur unitaire normal
50 anth = np.zeros(N-1)     #accélération selon le vecteur unitaire tangent
51 atth = np.zeros(N-1)     #accélération selon le vecteur unitaire normal
52
53 #Vérification de la structure de ce tableau
54 (3 colonnes ont été exportées : temps, X et Y)
   if M != 3 :
55     print("Problème dans la réalisation de votre pointage.
56 Reprenez le travail et exportez uniquement deux courbes
   (Mouvement_X et Mouvement_Y)")
       sys.exit()
57
```

Modules

Le module `numpy` permet des opérations sur les listes.
Le module `matplotlib` fournit les fonctionnalités graphiques.
Le module `csv` permet l'importation des données contenues dans le fichier *.csv.
Le module `sys` permet le contrôle de l'importation de ces données.

Création des listes

On créée des listes vides qui seront remplies par la suite.

À modifier : importation des données

Importation, dans la liste `table` des données contenues dans le fichier `pointage_centrifugeuse.csv`.
Cette ligne est à modifier à la question **3b** afin d'importer les données du deuxième fichier de pointage.

À modifier : initialisation des constantes du problème

On définit la durée entre deux positions successives. Elle dépend du fichier de pointage importé (question **3b**).

Initialisation des constantes du problème

On définit les constantes du problème.

Listes vitesse et accélération

Les listes vitesse (`vx` et `vy`) et accélération (`ax`, `ay`, `at`, `an`, `anth` et `atth`) sont initialement remplies grâce à la fonction `np.zeros` avec autant de zéros qu'il y aura de valeurs calculées.

L'initialisation des listes dépends de la méthode de construction de la liste dans la boucle for (lignes 70 à 78 puis 89 à 96).

> Remarque p. 7

Vérification de la structure des données importées

On vérifie avant de débuter les calculs que la structure de la liste `table` est celle attendue.
Le test `!=` signifie « différent de ».

```
58  #importation des coordonnées X, Y et du temps
59  for i in range(1,N):
60      #Génération des listes, avec transformation des données
61      temps.append(float(table[i][0].replace(',','.')))
62      x.append(float(table[i][1].replace(',','.')))
63      y.append(float(table[i][2].replace(',','.')))
64
65  #Création d'une fonction permettant de tracer les vecteurs
66  def trace_vect(x,y,Vectx,Vecty,titre,couleur,position):
67      q = plt.quiver(x,y,Vectx,Vecty,color = couleur,width=0.003)
68      plt.quiverkey(q, X=0.5, Y=position, U=10,label=titre, labelpos='E', color=couleur)
69
70  #Calculs approchés des coordonnées des vitesses
71  for i in range(1,N-2):
72      vx[i] = (x[i+1]-x[i-1])/(2*Dt)
73      vy[i] = (y[i+1]-y[i-1])/(2*Dt)
74
75  #Calculs approchés des coordonnées des accélérations
76  for i in range(2,N-3):
77      ax[i] = (vx[i+1]-vx[i-1])/(2*Dt)
78      ay[i] = (vy[i+1]-vy[i-1])/(2*Dt)
79
80  #Coordonnées de l'accélération et de la vitesse dans le repère de Frenet
81  #accélération
82  at =  np.multiply(ay,x)/R - np.multiply(ax,y)/R
83  an = - np.multiply(ay,y)/R - np.multiply(ax,x)/R
84
85  ############################################################
86  ###          DEBUT DE LA PARTIE A MODIFIER                ###
87  ############################################################
88
89  #Calculs théoriques de l'accélération
90  for i in range(1,N-1):
91      if temps[i] <= 2.65 :
92          anth[i] = 0
93          atth[i] = 0
94      else :
95          anth[i] = 0
96          atth[i] = 0
97
98  #tracer la norme de l'accélération
99  plt.plot(temps[2:N-3],an[2:N-3],".",label="an")
100 #plt.plot(temps[2:N-3],anth[2:N-3],label="anth")
101 plt.plot(temps[2:N-3],at[2:N-3],".",label="at")
102 #plt.plot(temps[2:N-3],atth[2:N-3],label="atth")
103
104 ############################################################
105 ###          FIN DE LA PARTIE A MODIFIER                  ###
106 ############################################################
107
108 #position du bloc légende
109 plt.legend(loc='upper left')
110
111 #étiquettes des axes
112 plt.xlabel(r'$ t (s)$')
113 plt.ylabel(r'$ accélération (m/s²)$')
114
115 #Titre du graphique
116 # plt.title('Positions et accélération de la cabine',loc='left')
117 plt.grid()
    plt.show()
```

Remplissage des listes prédéfinies

On remplit grâce à la commande `append` les listes `temps`, `x` et `y` avec les données importées et contenues dans la liste `table`.

> Remarque p. 7

Définition d'une fonction python

On définit la fonction `trace_vect` permettant de tracer un vecteur.

Calculs approchés des coordonnées

Calculs des coordonnées cartésiennes du vecteur vitesse `vx` et `vy`) et accélération (`ax` et `ay`).
Ces calculs utilisent les coordonnées du système importées (ligne 20).

Calculs dans le repère de Frenet.

Calculs des coordonnées dans le repère de Frenet du vecteur accélération (`at` et `an`).
Ces calculs utilisent les coordonnées du système importées (ligne 20).

À modifier : calculs de l'accélération théorique

On écrit ici les expressions, déterminées à la question **3a**, permettant de calculer les coordonnées du vecteur accélération théorique (`atth` et `anth`) dans le repère de Frenet.
On distingue les deux phases du mouvement.
Avant modification, le programme calcule des coordonnées nulles en tout point.

À modifier : Tracés des courbes

On trace les coordonnées du vecteur accélération (`at` et `an`) dans le repère de Frenet.
A la question **3a**, on active les lignes 100 et 102 en supprimant le symbole `#` pour tracer les coordonnées théorique (`atth` et `anth`) calculées.

À modifier : titre du graphique

On active cette ligne (en supprimant le symbole `#` en début de ligne) pour afficher le titre du graphique à la question **3a** lorsque les vecteurs vitesse et accélération sont tracés.

Remarque : Construction d'une liste

Il existe deux méthodes de construire une liste :

- **Méthode 1**

```
#Initialisation de la liste A
A = np.zeros(n)        #Liste A contenant n termes

#Construction de la liste
for i in range(1,n):
    A[i] = A[i-1] + b
```

La liste est initialement remplie grâce à la fonction `np.zeros` d'autant de zéros qu'il y aura de valeurs calculées.

Le $i^{ème}$ zéro de la liste est remplacé par la valeur calculée.

- **Méthode 2**

```
#Initialisation de la liste A
A = []        #Liste A vide

#Construction de la liste
for i in range(1,n):
    A.append(A[i-1] + b)
```

La liste est initialement vide.

La valeur calculée est ajoutée à la liste grâce à la fonction `append`.