

TP3 – TRAITEMENT DU SIGNAL

OBJECTIFS

- Revoir l'ouverture d'un fichier
- Méthode d'intégration liée à Euler
- Voir la précision d'un capteur
- Représenter une acquisition faite sur un système.

Durée : 2h00

On a effectué une mesure sur une imprimante 3D à l'aide d'un accéléromètre. Un capteur accélérométrique permet de mesurer une accélération. Il est généralement constitué d'une masselotte fixée au boîtier de la pièce en mouvement par un ressort. L'élongation du ressort permet de déterminer l'accélération du boîtier par rapport à un référentiel galiléen. Ces composants ont été largement miniaturisés durant les 10 dernières années pour s'intégrer aux téléphones portables, tablettes, manettes de jeux et autres objets électroniques ayant besoin de mesurer un mouvement.



FIGURE 1 – Accéléromètre miniature

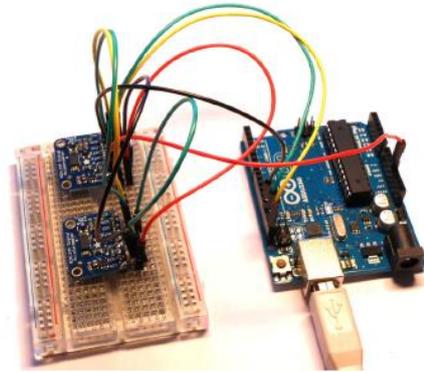


FIGURE 2 – Montage de l'accéléromètre sur l'arduino pour la mesure

Généralement, un capteur accélérométrique a été branché sur un arduino chargé de récupérer les mesures par un bus I2C et de retransmettre ces mesures à l'ordinateur par un protocole série. Les données envoyées par l'arduino sont pour chaque ligne : *le temps écoulé (en secondes) et les accélérations mesurées suivant X, Y et Z ont été enregistrées dans le fichier mesure_accelero.txt*

1. IMPORTATION DES MESURES ET VISUALISATION

On souhaite lire le fichier *mesure_accelero.txt*

Vous devez tout d'abord utiliser la bibliothèque os et vous placer dans le répertoire approprié !

```
from pylab import *
import numpy as np
import os
from matplotlib import *

os.chdir('votre répertoire')
fic=open("mesure_accelero.txt","r")
tab=[]
for ligne in fic :
    L=ligne.strip().split(',')
    tab.append(L)
fic.close()
```

1. Dans un nouveau programme, ouvrir le fichier de mesure et recopier le programme précédent.

Remplacer par votre répertoire de travail
Expliquer ce qui est stocké dans *tab*

2. Ecrire le programme qui permet de remplacer chacune des valeurs de *tab* par la valeur en flottant (*float*) et de le transformer en matrice (*array*)

Les pas de temps *DT* sont donnés en micro-secondes mais ne permettent pas en l'état de tracer l'évolution des courbes d'accélération en fonction du temps. Par ailleurs, les accélérations sont données en incréments et doivent être étalonnées, sachant que 10 ms^{-2} correspond à 255 incréments.

3. Écrire le programme permettant de calculer un vecteur des temps *T* qui permettra de tracer les courbes en fonction du temps (en secondes). Étalonner les courbes de façon à exprimer les données en ms^{-2}

4. Tracer les trois courbes d'accélération. Que constatez vous en début d'essai lorsque le capteur est immobile ? Pouvez-vous interpréter les valeurs d'accélération obtenues ?

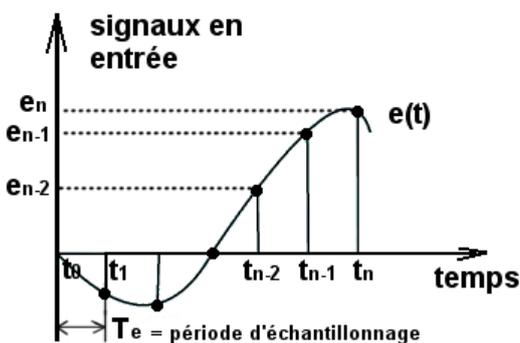
1.1. Filtrage

On souhaite lisser la courbe en appliquant un filtre passe haut ou passe bas. Le choix se fera en fonction du meilleur résultat.

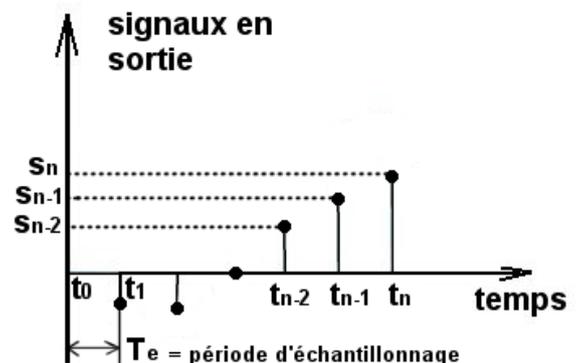
L'écriture de la fonction de transfert du filtre est donnée dans le domaine « continu » ; par exemple pour

un filtre du premier ordre (passe bas): $S(p) = \frac{K}{1 + \tau.p} . E(p)$ avec $K=1$

Or le « micro-contrôleur » qui doit réaliser l'opération de filtrage fonctionne dans le domaine « échantillonné » ; c'est-à-dire que la prise des informations d'entrée « *e(t)* » ainsi que la délivrance des résultats de calcul « *s(t)* » après traitement, s'effectuent de manière discontinue à des instants successifs définis par la « période d'échantillonnage » T_e .



Traitement



5. Créer la fonction python **filtrage(e, Te, tau)** qui permet de filtrer le signal *e(t)*. On supposera les conditions initiales égales à *e*.

6. Utiliser cette fonction pour filtrer votre signal (qui doit être contenu dans la variable *X*).

7. Superposer votre courbe filtrée avec le signal de départ.

8. Augmenter *tau* ; Que constatez-vous ?

1.2. Intégration de l'accélération et traitement des mesures

Pour obtenir la vitesse de déplacement, il faut intégrer l'accélération.

9. Proposer un programme permettant de calculer les composantes de vitesses V_X , V_Y et V_Z .
Tracer l'évolution des composantes de vitesses : que constatez vous ?

Pour un résultat exploitable, une première précaution consiste à régler l'offset en début de mesure :

```
#Etalonnage sur les 3 premières secondes
```

```
ni=1500
```

```
nf=9000
```

```
Ax2=np.zeros(ni)
```

```
Ay2=np.zeros(ni)
```

```
Az2=np.zeros(ni)
```

```
for i in range (len(Ax)):
```

```
    if t[i]<t[ni]:
```

```
        Ax2[i]=Ax[i]
```

```
        Ay2[i]=Ay[i]
```

```
        Az2[i]=Az[i]
```

```
plot(t,Ax2,"k")
```

```
EtalX=np.sum(Ax2)/1500
```

```
EtalY=np.sum(Ay2)/1500
```

```
EtalZ=np.sum(Az2)/1500
```

```
Axe=Ax-EtalX;
```

```
Aye=Ay-EtalY;
```

```
Aze=Az-EtalZ;
```

10. Refaire le calcul des vitesses et interpréter les courbes obtenues

1.3. Double intégration et tracé de trajectoires

11. Intégrer à nouveau les composantes de vitesse de façon à obtenir les composantes de position du capteur. Tracer les courbes et interpréter le résultat.

12. Tracer la trajectoire du capteur calculée dans le plan (\vec{x}, \vec{y}) et conclure sur les possibilités de mesure de position à l'aide d'un capteur accélérométrique.

2. GENERATION DE TRAJECTOIRE POUR UNE IMPRIMANTE 3D

Comme vous le savez sans doute, une imprimante 3D est un appareil permettant de fabriquer un objet volumique (généralement en plastique) par ajout de matière. Les imprimantes grand public sont constituées d'une tête chauffante faisant fondre un fil de plastique pour le déposer sur la zone à imprimer. La tête est montée sur 3 axes de translation de façon à déposer le fil fondu et progressivement former la géométrie de la pièce à fabriquer.



FIGURE 3 – Imprimante 3D cartésienne grand public

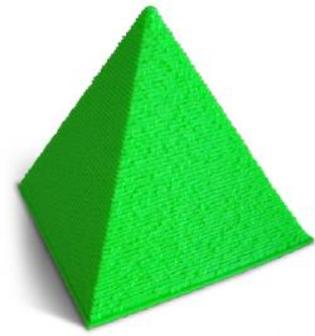


FIGURE 4 – Pyramide imprimée sur une imprimante 3D

La tête est régulée en température de façon à faire fondre à la bonne température le fil. Néanmoins, l'asservissement étant lent, le fil doit sortir à une vitesse constante pour maintenir les bons réglages. Cette contrainte impose à la tête de parcourir une trajectoire à vitesse constante elle aussi.

Nous cherchons dans ce problème à générer la trajectoire de consigne pour la tête d'impression, de façon à créer une pyramide miniature.

2.1. Trajectoire pour former une pyramide

La génération de la trajectoire brute (sans tenir compte de la vitesse de déplacement) nécessitant un peu de temps, le programme vous est donné dans le fichier *pyramide.py*.

13. Ouvrir le fichier et exécuter les commandes de façon à observer la trajectoire brute calculée.

2.2. Génération de la trajectoire machine

Cette trajectoire brute est constituée de segments plus ou moins long, mais ne présente aucun échantillonnage temporel : chaque segment visible est uniquement constitué de deux points extrémité.

Pour obtenir une trajectoire machine, il faut échantillonner la courbe en pas de temps réguliers de 10 ms en assurant une vitesse constante de $V_0 = 1\text{mm/s}$.

14. Créer une liste de temps pour la trajectoire brute, en considérant que chaque segment est parcouru à la vitesse V_0 .

15. Créer une liste de temps pour la trajectoire machine (en secondes).

16. Calculer par interpolation les coordonnées (x, y, z) pour chaque pas de temps machine puis tracer la trajectoire machine de façon à s'assurer qu'elle est bien identique à la trajectoire brute.

17. Calculer la vitesse d'avance obtenue par la trajectoire machine et vérifier qu'elle est bien constante et égale à V_0 .

TP3 – TENNIS

OBJECTIFS :

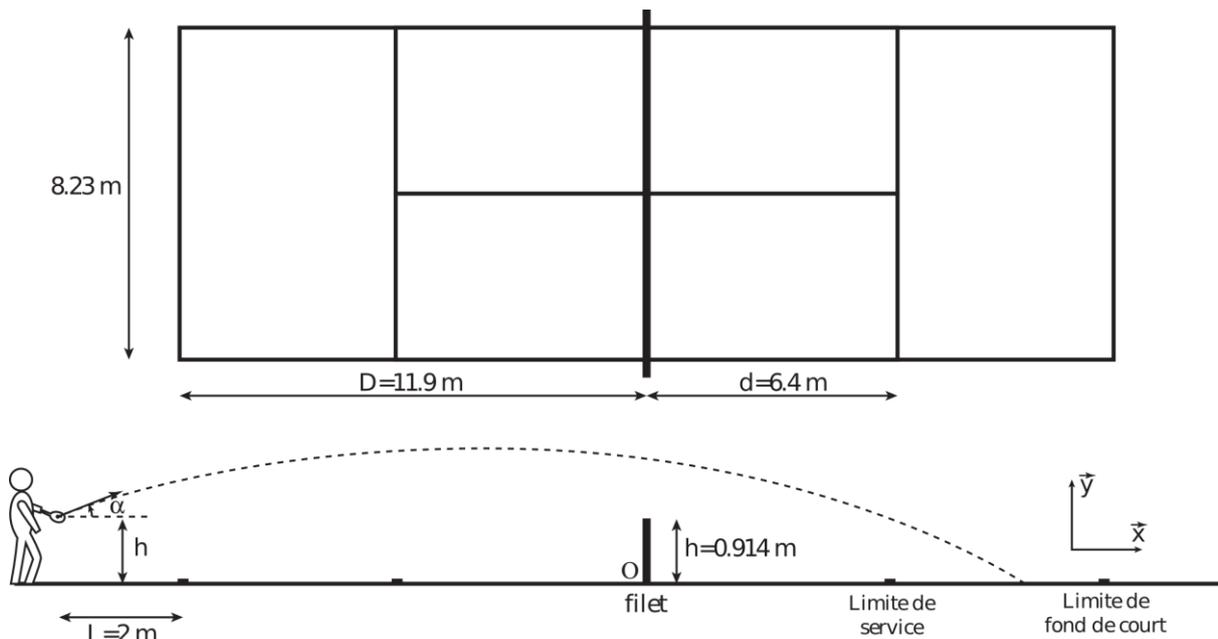
Le but de ce TP est de créer une petite application permettant de tracer la trajectoire d'une balle de tennis en fonction des paramètres de frappe du joueur... il s'agit surtout de coder la trajectoire et vous pourrez finaliser l'aspect GUI dans un deuxième temps.



1. PARAMETRAGE

On suppose en première approximation que la balle est renvoyée dans la direction du court de tennis. La direction est donc définie par l'angle α entre l'horizontale et la vitesse de la balle. Soit v_0 la vitesse de la balle juste après la frappe. On supposera que le joueur frappe la balle à une distance $L = 2 \text{ m}$ derrière la limite de fond de cours, à une hauteur $h = 0.914 \text{ m}$ (identique à la hauteur du filet). On note $M(x, y)$ le centre de gravité de la balle. Le centre O du repère est choisi au pied du filet.

La balle doit passer au-dessus du filet (hauteur $h = 0.914 \text{ m}$) et toucher le sol avant la limite de fond de court (distance filet-fond de court : $D = 11.9 \text{ m}$).



2. TRAJECTOIRE DE LA BALLE

3.1. Premier tracé

Nous allons définir la méthode de calcul de la trajectoire pour que celle-ci soit effectivement la trajectoire de la balle correspondant aux paramètres de frappe du joueur.

On néglige dans un premier temps le frottement de l'air sur la balle. Sachant que le joueur donne à l'instant initial (instant de la frappe) une vitesse \vec{V}_0 d'angle α avec l'horizontale (axe \vec{x}) et en appliquant le PFD à la balle, on en déduit les équations paramétriques de la trajectoire de la balle :

$$x(t) = V_0 \cos(\alpha) \cdot t + x_0$$

$$y(t) = -\frac{g}{2} t^2 + V_0 \sin(\alpha) \cdot t + y_0$$

Q1. Créer une fonction **Calcul** permettant de calculer la trajectoire parabolique de la balle.
 Q2. Puis créer la fonction **traceTrajectoire** en précisant dans l'appel à la fonction **Calcul** le paramètre de vitesse et d'angle puis testez votre application.

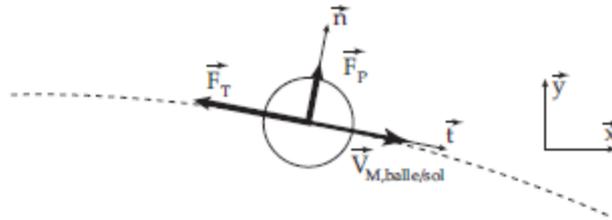
Sachant que l'on souhaite afficher 400 points sur une durée de simulation de 4s, complétez votre script en définissant les constantes ci-dessus :

- l'angle initial : $\alpha = 40^\circ$
- le vecteur temps **t**
- le vecteur vitesse initiale bidimensionnel **Vit0** (composantes en x et en y) $v_0 = 14m.s^{-1}$
- le vecteur position initiale bidimensionnel **Pos0** (composantes en x et en y) $x_0 = -12.5m$; $y_0 = 1m$
- les variables **x** et **y** correspondant respectivement à **x(t)** et **y(t)**
- en faisant tracer la trajectoire de la balle.

3.2. Trajectoire réelle

Hormis pour le service, la trajectoire de la balle est sensiblement perturbée par l'action de l'air. La viscosité de l'air conduit à un effort de traînée, proportionnel au carré de la vitesse : $\vec{F}_T = -\frac{1}{2}\lambda.V.\vec{V}$ où $\lambda = 26.10^{-4}N.m^{-2}.s^{-2}$. Lorsque les balles sont coupées (le joueur donne une vitesse de rotation à la balle), l'effet Magnus conduit à un effort de portance normal à la vitesse, orienté vers le haut ou vers le bas selon le sens de rotation : $\vec{F}_P = F_P.\vec{n}$ où $F_P = \mu.V.\omega$ avec une vitesse maximale de rotation de l'ordre de $\omega = \pm 100rad.s^{-1}$ et un coefficient de portance $\mu = 10^{-4}N.m^{-1}.s^{-2}$.

On notera θ l'angle $\theta = (\vec{x}(t), \vec{y}(t))$ et m la masse de la balle ($m=58g$)



Q3. Ecrire les équations différentielles du mouvement tenant compte de l'action de l'air sur la balle.

Vous devez obtenir les équations suivantes. Ces équations sont-elles linéaires ?

$$a_x = -0.5.\lambda.\sqrt{v_x^2 + v_y^2} * \frac{v_x}{m} \quad ; \quad a_y = -g - 0.5.\lambda.\sqrt{v_x^2 + v_y^2} * \frac{v_y}{m}$$

Le schéma temporel adopté pour l'intégration est le suivant :

$$\begin{cases} v_{k+1} = v_k + \Delta t * a_k \\ p_{k+1} = p_k + \Delta t * v_k + \frac{1}{2}\Delta t^2 * a_k \end{cases}$$

où a est l'accélération (a_x ou a_y), v est la vitesse (v_x ou v_y) et p la position (x ou y).

La procédure d'intégration numérique consiste, à partir des conditions initiales, à appliquer récursivement
 – le calcul de a_k par les équations du mouvement,
 – le calcul de v_k et p_k par le schéma numérique.

On souhaite tout d'abord tenir compte de la viscosité de l'air uniquement dans le calcul de la trajectoire (pas de l'effet Magnus).

- Q4. Dupliquez la partie de code correspondant au calcul du tir parabolique puis ajoutez la variable de viscosité en argument (variable booléenne indiquant si le calcul se fait avec ou sans viscosité).*
- Q5. Proposer alors un programme permettant de résoudre numériquement l'équation du mouvement de la balle.*

La prise en compte de l'effet Magnus amène les modifications suivantes sur l'expression des accélérations :

$$a_x = \frac{-0.5 \cdot \lambda \cdot \sqrt{v_x^2 + v_y^2} \cdot v_x - \mu \cdot v_y \cdot \omega}{m} \quad ; \quad a_y = -g + \frac{-0.5 \cdot \lambda \cdot \sqrt{v_x^2 + v_y^2} \cdot v_y + \mu \cdot v_x \cdot \omega}{m}$$

- Q6. Proposer enfin un autre programme permettant de tenir compte de l'effet Magnus puis tracer.*

3.3. Prise en compte du rebond

Jusqu'à présent, le calcul se fait sans tenir compte du rebond de la balle sur le sol.

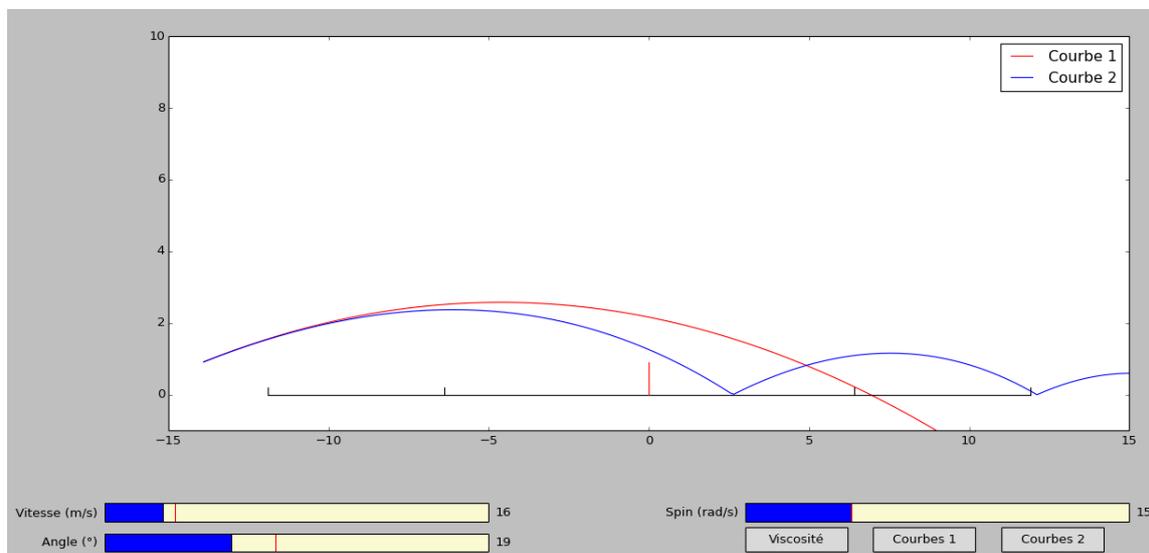
Une modélisation mathématique du rebond peut revenir simplement à changer le signe de la composante en y de la vitesse v_y et à l'atténuer d'un facteur e (coefficient de restitution au rebond) : $v_y = -e \cdot v_y$. avec $e=0.8 \dots$ Encore faut-il détecter le rebond ?!...

- Q7. Expliquez comment détecter un rebond et modifiez votre script afin de prendre en compte le rebond et l'atténuation de la vitesse qui en résulte.*
- Q8. D'autres phénomènes peuvent être pris en compte comme l'incidence du spin sur le rebond, l'effet du vent sur la trajectoire. Vous pouvez aussi vous lancer dans le calcul d'une trajectoire en trois dimensions...*

3. GRAPHISMES

Le début du programme est constitué de trois fonctions, une fonction Trace_terrain() permettant de tracer le terrain de tennis, et deux fonctions plot_courbe1() et plot_courbe2(), renvoyant le résultat $r=[x,y]$; contenant les deux les abscisses x et ordonnées y de la trajectoire en fonction du temps t pour deux courbes rouge et bleu.

La seconde partie du programme relève de la construction de l'interface graphique :



4.1. Terrain

Nous commencerons par tracer le terrain de tennis dans la fenêtre de l'application. Pour cela on choisit de représenter le terrain vu de profil (image du bas sur la première figure) ... la 3D c'est pour plus tard !!

Ouvrez le fichier *Graphique_eleve.py* dans python. Vous trouverez la déclaration de l'interface graphique avec les importations de widgets nécessaires, les paramètres de l'étude et quatre fonctions principales plus 3 petites fonctions de changement de variable globale (*chgt*).

La première fonction permet de tracer le terrain dans la fenêtre graphique et *courbe 1* et *courbe 2* permettent de créer la trajectoire.

La dernière fonction (*update_figure(val)*) est sans doute la plus importante : elle est lancée à chaque rafraichissement de la fenêtre graphique. On y trouve un appel aux fonctions *courbe1()* et *courbe2()* permettant de tracer respectivement les trajectoires de la balle.

Nous nous intéressons maintenant à la fonction *TraceTerrain()*.

Celle-ci est à peine commencée, pour vous donner les outils de base : la largeur et la hauteur de la fenêtre graphique (exprimées en pixels) sont placées dans les variables *largeur* et *hauteur*. Une commande permet de définir la couleur du tracé (le noir) et le rouge pour le filet. On remarquera que le système d'axe a pour origine le coin en haut à gauche et l'axe x est orienté vers la droite tandis que y est orienté vers le haut

Q9. A vous de réaliser votre procédure permettant de tracer le terrain de tennis en haut de la fenêtre graphique. Je vous invite à utiliser un repère intermédiaire dont l'origine est au pied du filet, l'axe x vers la droite et l'axe y vers le haut, et dont les dimensions sont exprimées en mètre pour faciliter par la suite le tracé de la trajectoire.

Q10. Testez votre application en lançant le programme.

Le terrain de tennis doit se dessiner correctement, de manière analogue à la fenêtre proposée en sur la figure page 3

4.2. Trajectoire de la balle

Nous allons maintenant nous occuper de tracer la trajectoire de la balle sur la fenêtre graphique.

Les paramètres sont renseignés dans les fonctions *courbe1* et *courbe 2*. Il vous suffit de compléter les parties manquantes (A COMPLETER) avec les programmes mis en place dans la partie 2.1 et 2.2. N'oubliez pas de renseigner le vecteur vitesse initiale.

La fonction *courbe 1* renvoie :

- *X* contenant la liste des abscisses de la trajectoire pour une durée *t* (vecteur temps créé à la ligne 98)
- *Y* contenant la liste des ordonnées de la trajectoire pour une durée *t*

La fonction *courbe 2* renvoie :

- *pxr* contenant la liste des abscisses de la trajectoire pour une durée *t*
- *pyr* contenant la liste des ordonnées de la trajectoire pour une durée *t*

4.3. Sliders et Boutons

Pour aller plus vite, j'ai utilisé ici les widgets associés à Matplotlib plutôt que de faire une interface Tkinter, mais cette solution est aussi envisageable et ressemblerait à la proposition que vous allez coder ici.

Q11. En vous inspirant du slider 1 Vitesse, réaliser les 2 autres sliders : Angle et Spin. Bien les positionner par rapport à la figure page 3

*Q12. En vous inspirant du bouton Courbe 1, réaliser les 2 autres boutons : Viscosité et Courbe 2. Il faudra alors compléter les fonctions **chgt2** et **chgtvis** pour affecter les valeurs $c2$ et $visc$ lors d'un click ... S'inspirer de **chgt1**.*

4. POUR ALLER PLUS LOIN ... LE RETOUR

On choisit maintenant de ne garder que la courbe 2 (la plus réelle) imaginer une solution pour tracer la trajectoire retour :

- *En déterminant la valeur de y sur le bord droit du terrain*
- *En mettant des sliders pour le joueur de droite (vitesse, angle, spin)*
- *En traçant la trajectoire de retour en vert.*