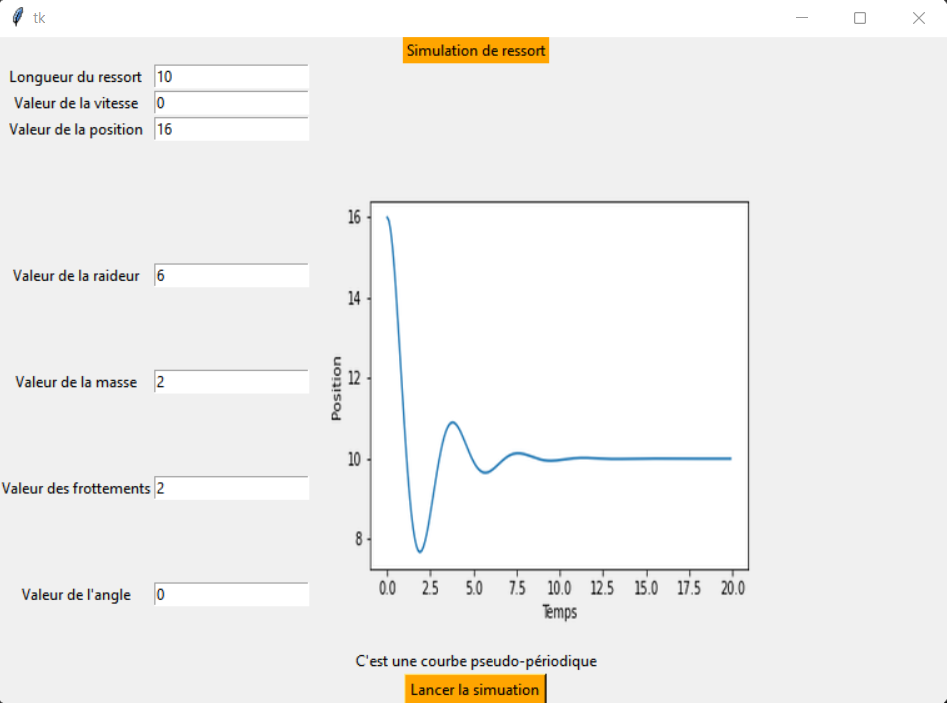
**Ivan Milosavljevic T02**

**Documentation du programme de la simulation d’un ressort**

***Introduction***

Ce programme vise à simuler l’évolution d’un ressort en fonction du temps. Pour l’utilisation du programme, il suffit de l’exécuter et une fenêtre va s’ouvrir, ensuite il faut entrer des valeurs numériques des paramètres du ressort dans les différentes cases (longueur du ressort, vitesse…) et après il faut appuyer sur le bouton « lancer la simulation » pour lancer le calcul, et un graphique va apparaître dans la fenêtre. Ce graphique est donc l’évolution du ressort en fonction du temps avec les paramètres que l’utilisateur a entré.

Par exemple :



Par la suite je vais expliquer implémentation du programme etl’utilité de chaque fonction. Je vais aussi présenter les tests que j’ai fait pour valider les différents composants du programme.

***def calculate\_position(t, f, v0, x0, k, m, theta, L0):***

Cette fonction permet de calculer des valeurs de la position du ressort, en fonction du temps, avec des valeurs fixes qui sont les frottements (f), la vitesse(v0), le position initiale (k0), la raideur du ressort (k), la masse du ressort (m), l’angle du ressort (theta) et la longueur du ressort (L0).

Si on pose par exemple au début :

calculate\_position(0, 1, 0, 15, 5, 1, 0, 0)

15.0

On voit qu’au temps initial (0) , la position du ressort reste inchangée, elle est toujours à 15.

On pose au début :

gamma = f/m

w0 = sqrt(k/m)

delta = gamma\*\*2 - 4\*w0\*\*2

g = 9.81

F = m\*g\*sin(theta)/k + L0

Car nous allons utiliser ces valeurs pour toutes les équations de la fonction.

**Pour delta < 0, on va prendre un f = 1 et un k = 5 en comparaison pour ainsi obtenir une courbe pseudo-périodique à l’aide de l’équation :**

w1 = sqrt(w0\*\*2 - (gamma\*\*2/4))

A = x0

B = 2\*v0 + x0\*gamma / (2\*w1)

return exp(-gamma/2\*t) \* (A\*cos(w1\*t) + B\*sin(w1\*t)) + F

calculate\_position(0, 1, 0, 15, 5, 1, 0, 9)

24.0

calculate\_position(1, 1, 0, 15, 5, 1, 0, 9)

5.51051384927362

calculate\_position(3, 1, 0, 15, 5, 1, 0, 9)

12.432391441344208

calculate\_position(25, 1, 0, 15, 5, 1, 0, 9)

8.999962306104772

calculate\_position(35, 1, 0, 15, 5, 1, 0, 9)

9.000000305991076

On remarque que les valeurs tendent à osciller, et tendent vers 9, or c’est synonyme d’une courbe pseudo-périodique. Si on calcule avec une machine avec les mêmes valeurs on trouve bien les mêmes résultats.

**Pour delta > 0, on peut prendre f = 10 et k = 2 et donc avoir une courbe apériodique à l’aide de l’équation :**

w1 = sqrt(gamma\*\*2/4-w0\*\*2)

B = (2\*v0 + gamma\*x0 - 2\*w1\*x0) / (-4\*w1)

A = x0 - B

return exp(-gamma/2\*t) \* (A\*exp(w1\*t) + B\*exp(-w1\*t)) + F

calculate\_position(0, 10, 0, 15, 2, 1, 0, 9)

24.0

calculate\_position(7, 10, 0, 15, 2, 1, 0, 9)

12.669052726906791

calculate\_position(15, 10, 0, 15, 2, 1, 0, 9)

9.71647329969934

calculate\_position(30, 10, 0, 15, 2, 1, 0, 9)

9.033508992356573

On voit que les valeurs tendent vers 9, sans osciller, qui est la longueur du ressort qu’on a défini dans la fonction, ce qui est logique.

**Pour delta = 0, on peut prendre par exemple f = 2 et k = 1 pour faire en sorte que delta soit égale à 0 pour avoir une courbe critique à l’aide de l’équation :**

A = x0

B = v0 + gamma\*x0/2

return exp(-gamma/2\*t) \* (A + B\*t) + F

calculate\_position(0, 2, 0, 15, 1, 1, 0, 9)

24.0

calculate\_position(5, 2, 0, 15, 1, 1, 0, 9)

9.606415229917692

calculate\_position(8, 2, 0, 15, 1, 1, 0, 9)

9.04528745476684

calculate\_position(12, 2, 0, 15, 1, 1, 0, 9)

9.0011981214089

Les valeurs tendent vers 9 la longueur du ressort, sans osciller, synonyme d’une courbe critique.

Pour ce qui est de « label\_type » ça permet d’afficher un message en dessous du graphe qui

donne des précisions à l’utilisateur sur la nature des mouvements du ressort.

***def plotgraph(tmax, f):***

Cette fonction permet de tracer une courbe d’une fonction en fonction du temps.

On a :

abscisse = []

origine = []

t = 0

while t < tmax:

abscisse.append(f(t))

origine.append(t)

t += 0.1

On va donc insérer les valeurs du temps et de la fonction dans les listes vides « abscisse » et « origine ». Le temps qui commence bien sûr a 0, et avec un pas de 0.1, pour qu’on ait une

courbe précise. Cet algorithme va donc être réitéré autant de

fois qu’il est nécessaire pour que « t » soit supérieur à « tmax ».

Par exemple :

def fonction(t):

return 3 \* t

plotgraph(0.3, fonction)

[0, 0.30000000000000004, 0.6000000000000001]

[0, 0.1, 0.2]

La fonction marche bien.

Ensuite nous allons utiliser les valeurs de la liste pour créer une courbe à l’aide de plt.plot

Après le programme va convertir le graph généré par matplotlib en une image , et écraser

l’image précédemment généré (si elle existe) et va sauvegarder cette image.

***def runProgram():***

Cette fonction va nous permettre de démarrer le calcul. On a :

g = 9.81

f = float(Frott.get())

v0 = float(Vitesse.get())

x0\_réel = float(Position.get())

k = float(Raideur.get())

m = float(Masse.get())

theta = float(Angle.get())

L0 = float(Longueur.get())

x0 = x0\_réel - m\*g\*sin(theta)/k - L0

Qui nous permet de récupérer les valeurs que l’utilisateur a entre dans l’interface graphique.

Par exemple si on tape 5 dans la case « valeur des frottements » float(Frott.get()) va

récupérer la valeur 5 tout simplement, idem pour les autres. Si l’utilisateur entre des valeurs

négatives pour les frottements, la masse, la raideur, la vitesse, et la longueur du ressort, le

programme s’arrête. Il s’arrête aussi si la raideur du ressort, la masse, la longueur du

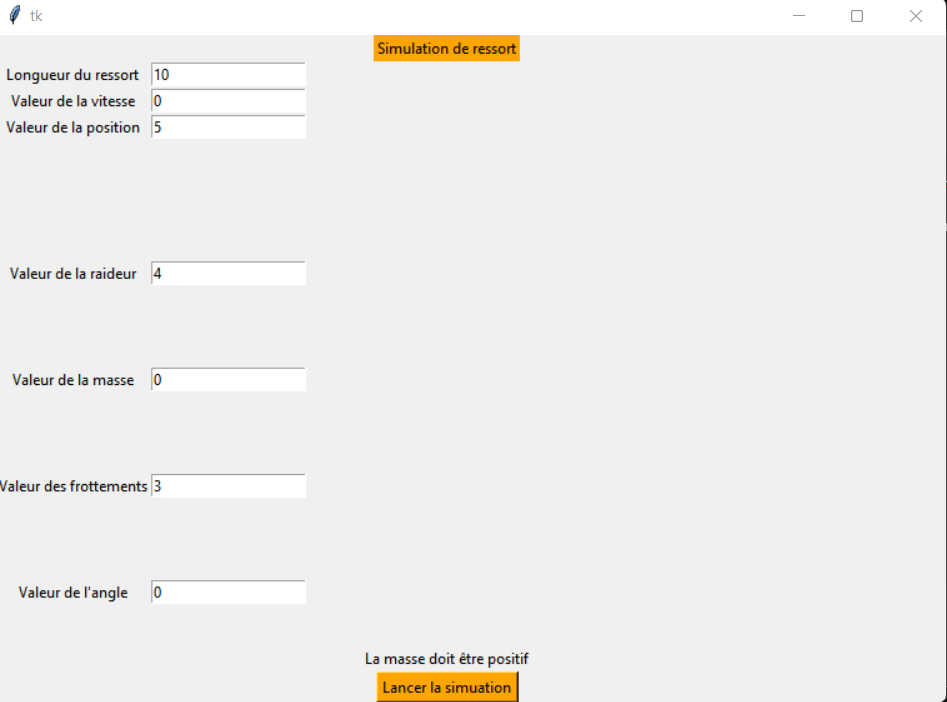
ressort est égale à 0, sinon ça n’a aucun sens physique.

Pour ca j’ai fait avec des True et des False. J’ai posé True pour la variable

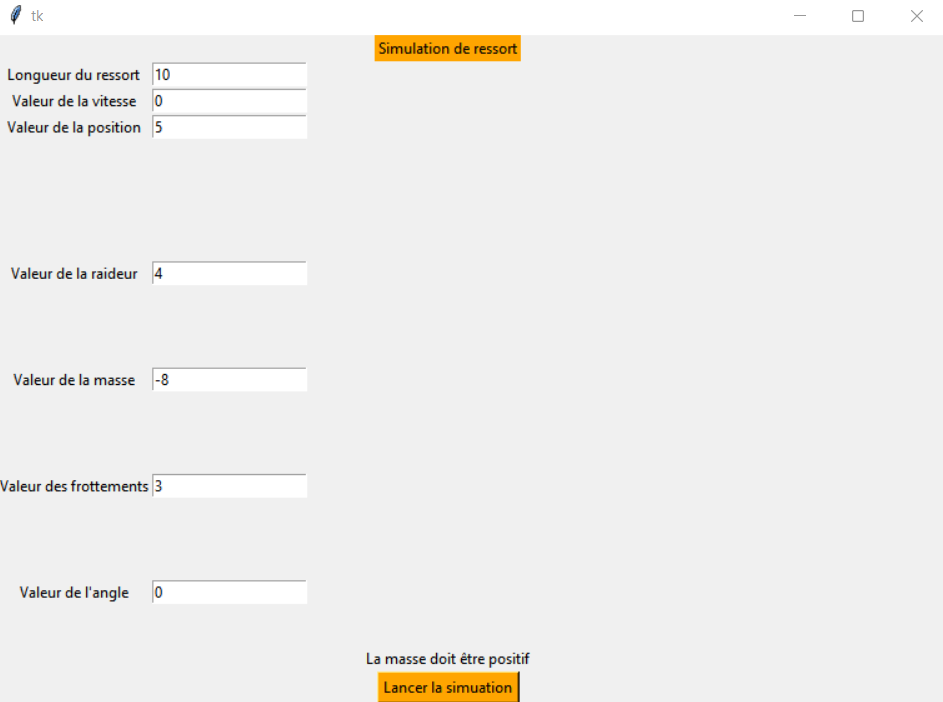
« valeurs\_impossibles » , et si les valeurs sont mauvaises pour la masse, la longueur, la raideur et les frottements, « valeurs\_impossibles » va devenir False, et va afficher dans le

programme un label disant quelle valeur est fausse.

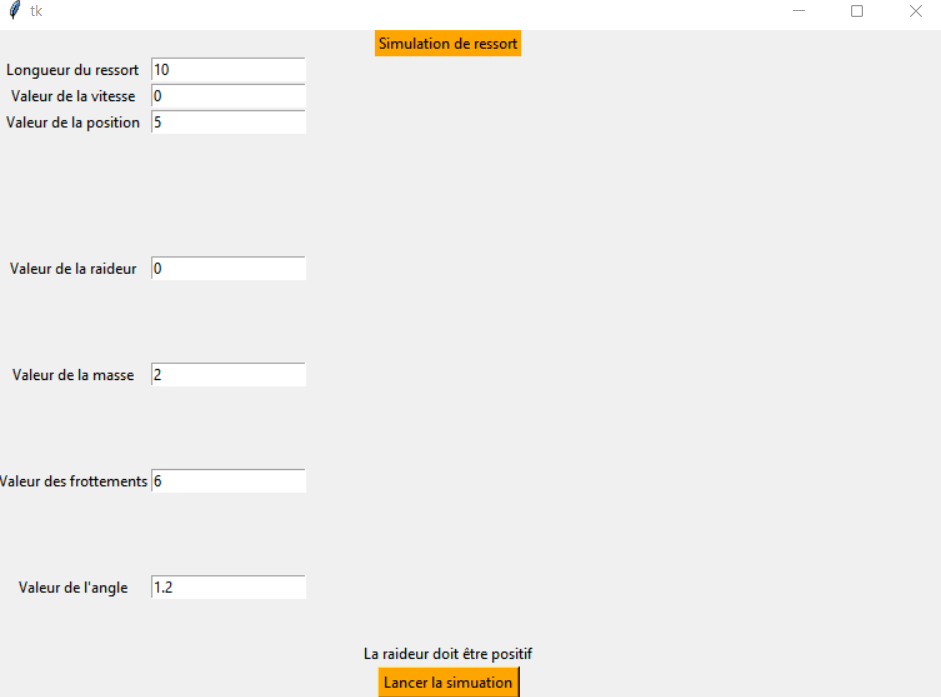
Si on met que la masse = 0 :



Ou que la masse = -8 :



Ou que la raideur = 0 :



Le programme ne peut pas être lancé lorsqu’on pose des valeurs impossibles.

Ensuite on a :

**plotgraph(tmax, lambda t: calculate\_position(t, f, v0, x0, k, m, theta, L0))**

Lorsqu’on appelle plotgraph, l’ordinateur va pouvoir créer une courbe contenant les valeurs

calculées à l’aide de la fonction calculate\_position, en fonction du temps tmax. On met

**lambda t: calculate\_position(t, f, v0, x0, k, m, theta, L0) ,** sinon plotgraph ne pourra pas utiliser la fonction car il y a plusieurs variables différentes, donc on fixe toutes les variables sauf t

pour ainsi créer une fonction à une variable qui est le temps, avec les autres valeurs qui sont

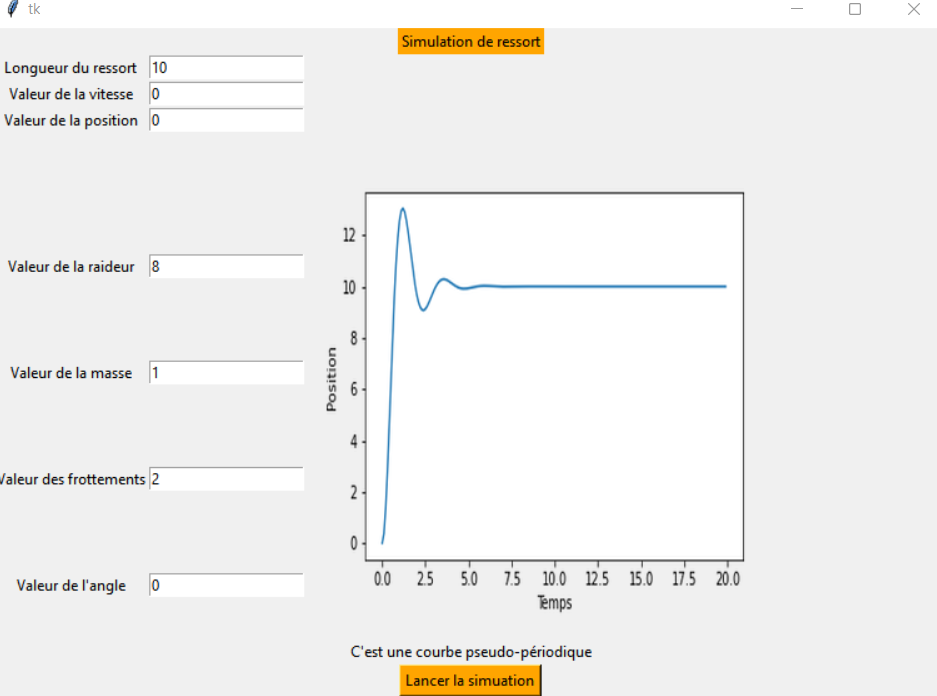
des constantes.

Pour tester on peut écrire :

plotgraph(20, lambda t: calculate\_position(t, f, v0, x0, k, m, theta, L0))

Avec f = 2, v0 = 0 ,x0 = 0, k = 8, m = 1, theta = 0, L0 = 10

On appuie ensuite sur le bouton lancer la simulation pour exécuter la fonction runProgram() :



On a donc bien une courbe sur un intervalle de temps [0 ;20] et une courbe en accord avec les valeurs données.

Pour ce qui est du temps maximum j’ai posé 20 car pour la majorité des cas le ressort va

arrêter d’osciller au bout de 20 secondes, mais on peut très bien le changer, il suffit de

changer la valeur de « tmax » dans la fonction  « runProgram ».

***def Showgraph():***

Cette fonction permet de faire en sorte que l’image soit affichée directement dans l’interface graphique lorsqu’on appuie sur le bouton. En effet, global img permet de rendre la variable

« img » global au lieu de local vu qu’on est dans une fonction, comme ça l’objet avec l’image

ne va pas être détruire a la sortie de la fonction.

En lien avec la fonction précédent « runProgram » qui renvoie à la valeur True, dans ce cas

Showgraph pourra générer une image. Dans le cas contraire Showgraph ne va rien faire.

Pour tester la fonction il suffit d’entrer des valeurs dans l’interface graphique et d’appuyer sur le bouton lancer le programme, vous pourrez voir que l’image s’affiche directement.