# Annexe : Code python

## Robots footballeurs : Commande par optimisation de trajectoire polynomiale

Gabin LEMPREZ

Culture Sciences de l'Ingénieur

Édité en Octobre 2021

école normale supérieure paris—saclay

*Cette Annexe complète la ressource « Robots footballeurs : Commande par optimisation de trajectoire polynomiale » [1].*

```python
# trajectoire_statique_deg3.py

import numpy as np
from matplotlib import pyplot as plt
import time

#                       PARAMETRES

t1 = time.perf_counter() #initialise l'horloge

X = np.linspace(0,1,10)
xb, yb= 1, 0.4
obstacles = [(0.35, -0.1),(0.6, -0.5)]
R = 0.15 #parametre uniquement visuel : rayon en dessous duquel il y a collision
P0 = np.array([0.001,0.001]) #initialisation
beta = 0.1 #intensité de l'évitement
eps1 = 0.001 #seuil sur la norme du gradient sous lequel on considère qu'elle
             #est nulle
eps2 = 0.0001 #seuil sur la stagnation de l'amelioration du critere

def y(x, P) :
    return (P[1]*(x**3 - xb**2 * x) + P[0]*(x**2 - xb*x) + yb*x/xb)

def critere(P) :
    #fonction a minimiser
    Longueur = 0
    for i in range(len(X)-1) :
        Longueur += np.sqrt((X[i+1] - X[i])**2 + (y(X[i+1], P) - y(X[i],P))**2)

    minDist = 1
    for obstacle in obstacles :
        for x in X :
            dist = np.sqrt((obstacle[0]-x)**2 + (obstacle[1]-y(x,P))**2)
            if dist <= minDist:
                minDist = dist

    return Longueur + beta/minDist


#_____________________METHODE DU GRADIENT_____________________________________

def methodeGradient(P):
    #renvoie P qui minimise le critere selon la methode du gradient
    #le vecteur initial est donné en argument

    fini = False
    k = 0
    alpha = 1

    while fini == False :
        g = descentDirection(P)
        if (k>1000) or (max(abs(g))<eps1)  or (abs(critere(P-alpha*g) -
                                                   critere(P))<eps2):
            #critere d'arret sur la norme infinie par defaut
            fini = True
        else :
            alpha = stepSearch(P,g)
            P = P - alpha * g
            k += 1
    print("nombre d'iterations : ", k)
    return P
```

```
060|
061|
062|
063| def stepSearch(x,p):
064|     #renvoie le pas pour la descente de gradient
065|     tau = 0.5 #constante dans [0,1], faire varier, 0.5 par défaut
066|     l = 0
067|     alpha = 1
068|     while critere(x - alpha*p) >= critere(x) and l<100 :
069|         alpha = tau*alpha
070|         l += 1
071|     return alpha
072|
073| def descentDirection(x) :
074| #calcule le gradient de la fonction critere en x (Euler implicite)075|
dp = 0.01
076| d1 = (1/dp)*(critere(x+np.array([dp,0])) - critere(x))077|
d2  =  (1/dp)*(critere(x+np.array([0,dp]))  -  critere(x)) 078|
return np.array([d1,d2])
079|
080| coeff = methodeGradient(P0)
081| t2 = time.perf_counter() #valeur finale à l'horloge
082|
083| #_____________________________mise en forme_____________________________________
084|
085| print('vecteur parametre : ',coeff)
086| plt.plot(X, [y(x,coeff) for x in X], label = 'trajectoire')
087| plt.plot(xb,yb, color='red', marker = 'x')
088| plt.plot(0,0, marker='o', color= 'red')
089| for obstacle in obstacles :
090|     #trace les obstacles
091|     plt.gca().add_patch(plt.Circle((obstacle[0], obstacle[1]),
092|                                    R, color='green', fill=False,
093|                                    label = 'obstacle'))
094|  plt.xlim(0,1)
095| plt.ylim(-0.5,0.5)
096| plt.xlabel("x")
097| plt.ylabel("y")
098| plt.legend()
099| plt.show()
100|
101| print('temps de calcul : ' , t2-t1)
102|
103|
104|
```

## Références :


[1]: Robots footballeurs : Commande par optimisation de trajectoire polynomiale, G. Lemprez, J. Ojeda, octobre 2021, https://eduscol.education.fr/sti/si-ens-paris-saclay/ressources_pedagogiques/robots-footballeurs-commande-par-optimisation-de-trajectoire-polynomiale