CODES

Regression linéaire [python]

**import matplotlib.pyplot as plt**

**import numpy as np**

**Y = 9.81\*np.linspace(0.11,0.11+(8\*0.05),8)**

**X = np.array([0.15, 0.248, 0.349, 0.47, 0.602, 0.72, 0.85, 0.97])**

**print("X = ", X)**

**print("Y = ", Y)**

**ecart = []**

**for i in range(len(Y)-1):**

**ecart.append(X[i+1]-X[i])**

**print(ecart)**

**pas = Y[1] - Y[0]**

**k = 1/(np.mean(ecart)/(pas))**

**print("k = ", k, " en N/m")**

**#Y\_droite = k\*(X - X[0]) + np.mean(Y) - (k\*(X[-1] - X[0])/2)**

**Y\_droite = k\*(X - np.mean(X)) + np.mean(Y)**

**ecart\_type = np.sqrt(np.mean([x\*x/(pas\*pas) for x in ecart]) - (np.mean(ecart)/pas)\*\*2)**

**print(ecart\_type )**

**plt.plot(X,Y)**

**plt.scatter(X,Y)**

**plt.xlabel("élongation (m)")**

**plt.ylabel("poids (N)")**

**plt.plot(X,Y\_droite)**

**plt.show()**

Résolution de tir [C++ & Python]

**#include <iostream>**

**#include <cmath>**

**#include <string>**

**#include <vector>**

**#include <fstream>**

**using namespace std;**

**// general function**

**auto linspace(double low, double up, int n)**

**{**

**vector<double> array;**

**array.resize(n);**

**for (int i=0; i<n; i++)**

**{**

**array[i] = low + (i\*(up-low)/(n-1));**

**}**

**return array;**

**}**

**double grad(double u\_\_,double u\_,double u,double dx)**

**{**

**return( ((u\_ - u\_\_) + (u - u\_))/(2\*dx) );**

**}**

**double grad2(double u\_\_,double u\_,double u,double dx)**

**{**

**return( (u - 2\*u\_ + u\_\_ )/(dx\*dx) );**

**}**

**int write\_data(vector<double> data, int n, int index)**

**{**

**ofstream output\_file;**

**output\_file.open("data/data\_" + to\_string(index) + ".txt");**

**for (int i=0; i < n; i++)**

**{**

**output\_file << data[i] << '/';**

**}**

**output\_file.close();**

**return 0;**

**}**

**int infos(double dt, double m, double r, double K\_D, double K\_M, double k\_r, double J, double alpha, double v\_0, vector<double> v\_x , vector<double> v\_z, double w\_0)**

**{**

**cout << "dt = " << dt << endl;**

**cout << "masse = " << m << endl;**

**cout << "rayon = " << r << endl;**

**cout << "K\_D = " << K\_D << endl;**

**cout << "K\_M = " << K\_M << endl;**

**cout << "k\_r = " << k\_r << endl;**

**cout << "J = " << J << endl;**

**cout << "alpha = " << alpha\*(180/3.141591628) << endl;**

**cout << "v\_0 = " << v\_0 << endl;**

**cout << "v\_x[0] = " << v\_x[0] << endl;**

**cout << "v\_z[0] = " << v\_z[0] << endl;**

**cout << "w\_0 = " << w\_0 << endl;**

**return 0;**

**}**

**// solve**

**double solve\_theta(double theta\_time, double v\_x\_time\_1, double v\_x\_time, double v\_z\_time\_1, double v\_z\_time)**

**{**

**// return ( theta\_time + asin( ((v\_x\_time\*v\_z\_time\_1) - (v\_x\_time\_1\*v\_z\_time)) / ( sqrt( (v\_x\_time\*v\_x\_time + v\_z\_time\*v\_z\_time) \* ( v\_x\_time\_1\*v\_x\_time\_1 + v\_z\_time\_1\*v\_z\_time\_1 ) ) ) ) );**

**return ( atan(v\_z\_time / v\_x\_time) );**

**}**

**double solve\_speed\_x(double v\_x, double v\_z, double t, double dt, double theta, double m, double g, double K\_D, double K\_M, double k\_r, double J, double w\_0)**

**{**

**// cout << (-cos(theta) \* K\_D \* ( v\_x\*v\_x + v\_z\*v\_z ) ) << endl;**

**// cout << -( sin(theta) \* K\_M \* (w\_0\*exp(-(k\_r/J)\*t)) \* ( sqrt( v\_x\*v\_x + v\_z\*v\_z ) ) ) << endl;**

**// cout << K\_D << endl;**

**// cout << K\_M << endl;**

**return( v\_x + ( (-cos(theta) \* K\_D \* ( v\_x\*v\_x + v\_z\*v\_z ) ) - ( sin(theta) \* K\_M \* (w\_0\*exp(-(k\_r/J)\*t)) \* ( sqrt( v\_x\*v\_x + v\_z\*v\_z ) ) ) )\*(dt/m) );**

**}**

**double solve\_speed\_z(double v\_x, double v\_z, double t, double dt, double theta, double m, double g, double K\_D, double K\_M, double k\_r, double J, double w\_0)**

**{**

**// cout << ( - m\*g - ( sin(theta) \* K\_D \* ( v\_x\*v\_x + v\_z\*v\_z ) ) + ( cos(theta) \* K\_M \* ( w\_0\*exp(-(k\_r/J)\*t) ) \* ( sqrt(v\_x\*v\_x + v\_z\*v\_z) ) ) )\*(dt/m)<< endl;**

**return( v\_z + ( - m\*g - ( sin(theta) \* K\_D \* ( v\_x\*v\_x + v\_z\*v\_z ) ) + ( cos(theta) \* K\_M \* ( w\_0\*exp(-(k\_r/J)\*t) ) \* ( sqrt(v\_x\*v\_x + v\_z\*v\_z) ) ) )\*(dt/m) );**

**}**

**// main function**

**int main(int argc, char \*argv[])**

**{**

**// init variables**

**int nt = 50000;**

**vector<double> t = linspace(0.0, 10.0, nt);**

**double dt = t[1] - t[0];**

**double g = 9.81;**

**double m = 0.6;**

**double r = 0.12;**

**double K\_D = 0.5 \* 1.2 \* 3.141591628 \* r \*r \* 1.15;**

**// cout << K\_D << endl;**

**double K\_M = 0.5 \* 1.2 \* 3.141591628 \* r \* r \* r;**

**double k\_r = 0.001;**

**double J = 0.667\*m\*r\*r;**

**vector<double> v\_x;**

**vector<double> v\_z;**

**vector<double> theta;**

**v\_x.resize(nt);**

**v\_z.resize(nt);**

**theta.resize(nt);**

**vector<double> x;**

**vector<double> z;**

**x.resize(nt);**

**z.resize(nt);**

**x[0] = 0;**

**z[0] = 1.9;**

**//cout << "ARGC = " << argc << endl;**

**double alpha = 40\*(3.141591628)/180;**

**double v\_0 = 9.5;**

**if (argc == 3)**

**{**

**v\_0 = strtod(argv[1], NULL)\*1.0;**

**alpha = strtod(argv[2], NULL)\*(3.141591628)/180;**

**//cout << v\_0 << endl;**

**//cout << alpha << endl;**

**};**

**// cout << v\_0 << endl;**

**// cout << alpha << endl;**

**double w\_0 = 2\*3.141591628\*3;**

**theta[0] = alpha;**

**v\_x[0] = cos(theta[0])\*v\_0;**

**v\_z[0] = sin(theta[0])\*v\_0;**

**//infos(dt, m, r, K\_D, K\_M, k\_r, J, alpha, v\_0, v\_x ,v\_z, w\_0);**

**// solve**

**for (int time=0; time<nt; time++)**

**{**

**v\_x[time+1] = solve\_speed\_x(v\_x[time], v\_z[time], t[time], dt, theta[time], m, g, K\_D, K\_M, k\_r, J, w\_0);**

**v\_z[time+1] = solve\_speed\_z(v\_x[time], v\_z[time], t[time], dt, theta[time], m, g, K\_D, K\_M, k\_r, J, w\_0);**

**theta[time+1] = solve\_theta(theta[time], v\_x[time+1], v\_x[time], v\_z[time+1], v\_z[time]);**

**//cout << "time = " << time << endl;**

**//cout << "v\_x = " << v\_x[time] << endl;**

**//cout << "v\_z = " << v\_z[time] << endl;**

**//cout << "theta = " << (180/3.141591628)\*theta [time] << endl;**

**//cout << endl;**

**}**

**for (int time=0; time<nt; time++)**

**{**

**x[time+1] = x[time] + v\_x[time]\*dt;**

**z[time+1] = z[time] + v\_z[time]\*dt;**

**if (z[time+1] < 0)**

**{**

**z[time+1] = 0;**

**}**

**}**

**write\_data(x,nt,1);**

**write\_data(z,nt,2);**

**return 0;**

**}**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from matplotlib import style**

**import os**

**#style.use('dark\_background')**

**os.system("./main 9.5 40")**

**with open("data/data\_1.txt", 'r') as file:**

**x = file.read()**

**with open("data/data\_2.txt", 'r') as file:**

**z = file.read()**

**x = x.split("/")**

**z = z.split("/")**

**X = []**

**Z = []**

**for t in range(len(x)-1):**

**X.append(float(x[t]))**

**Z.append(float(z[t]))**

**# print(X)**

**print()**

**print()**

**# print(Z)**

**t = np.linspace(0,2,40)**

**g = 9.81**

**v\_x = 7.27742**

**v\_z = 6.10648**

**def x\_f(t):**

**return(v\_x\*t)**

**def z\_f(t):**

**if -g\*t\*t/2 + v\_z\*t + 1.9 > 0:**

**return(-g\*t\*t/2 + v\_z\*t + 1.9)**

**else:**

**return(0)**

**X\_f = []**

**Z\_f = []**

**for time in range(len(t)):**

**X\_f.append(x\_f(t[time]))**

**Z\_f.append(z\_f(t[time]))**

**X\_ = []**

**Z\_ = []**

**for i in range(len(Z)):**

**if Z[i] != 0:**

**Z\_.append(Z[i])**

**X\_.append(X[i])**

**plt.plot(X\_,Z\_)**

**#plt.scatter(X\_f, Z\_f, color="red", marker='o')**

**plt.plot([6.7-0.15, 6.7+0.15], [3.1, 3.1], color="red")**

**plt.show()**

Caractéristique [Python]

**import numpy as np**

**import matplotlib.pyplot as plt**

**from matplotlib import style**

**import os**

**#style.use('dark\_background')**

**g = 9.81**

**res = [(0,0)]**

**#for v in range(4\*20, 10\*20):**

**for v in range(6\*20, 14\*20):**

**#for v in range(8\*20, 20\*20):**

**for alpha in range(10\*4, 80\*4):**

**print("./main {0} {1}".format(v\*0.05,alpha\*0.25))**

**os.system("./main {0} {1}".format(v\*0.05,alpha\*0.25))**

**with open("data/data\_1.txt", 'r') as file:**

**x = file.read()**

**with open("data/data\_2.txt", 'r') as file:**

**z = file.read()**

**x = x.split("/")**

**z = z.split("/")**

**X = []**

**Z = []**

**for t in range(len(x)-1):**

**X.append(float(x[t]))**

**Z.append(float(z[t]))**

**for pos in range(len(X)):**

**# check if the ball is at the right pos in x**

**#if ((3.66-0.2) <= X[pos] <= (3.66+0.2)):**

**if ((7.25-0.2) <= X[pos] <= (7.25+0.2)):**

**#if ((11.9-0.2) <= X[pos] <= (11.9+0.2)):**

**# check if the ball is at the right pos in z**

**if (3.1 <= Z[pos] <= (3.1 + 0.2)):**

**# check if the ball is going down**

**if Z[pos-1] > Z[pos]:**

**if res[-1] != (v\*0.05,alpha\*0.25):**

**res.append( (v\*0.05,alpha\*0.25) )**

**print("BUCKET !")**

**res.pop(0)**

**print(res)**

**#with open("data/data\_lance-franc-v.txt", 'w') as data:**

**with open("data/data\_3-points-v.txt", 'w') as data:**

**#with open("data/data\_center-v.txt", 'w') as data:**

**for i in res:**

**data.write(str(i[0])+"/")**

**#with open("data/data\_lance-franc-alpha.txt", 'w') as data:**

**with open("data/data\_3-points-alpha.txt", 'w') as data:**

**#with open("data/data\_center-alpha.txt", 'w') as data:**

**for i in res:**

**data.write(str(i[1])+"/")**

**for i in res:**

**plt.scatter(i[0],i[1], color='red')**

**#plt.savefig("renders/lance-franc.png")**

**plt.savefig("renders/3-points.png")**

**#plt.savefig("renders/center.png")**

**plt.show()**

Simulation [Python]

**import numpy as np**

**import matplotlib.pyplot as plt**

**import matplotlib.image as mpimg**

**def dist(x1,y1,x2,y2):**

**return(np.sqrt((x2-x1)\*\*2 + (y2-y1)\*\*2))**

**def main():**

**Nx = 800**

**Ny = 400**

**Nt = 40000**

**tau = 0.53**

**x = np.linspace(0, Nx-1, Nx)**

**y = np.linspace(0, Ny-1, Ny)**

**X, Y = np.meshgrid(x, y)**

**# vitesses**

**Nl = 9 #nombre de vitesses discretes**

**#vecteurs vitesse discrets**

**cxs = np.array([0, 0, 1, 1, 1, 0, -1, -1, -1])**

**cys = np.array([0, 1, 1, 0, -1, -1, -1, 0, 1])**

**# poids**

**weights = np.array([4/9, 1/9, 1/36, 1/9, 1/36, 1/9, 1/36, 1/9, 1/36])**

**# conditions initiales**

**V = np.ones((Ny,Nx,Nl)) + 0.01\*np.random.randn(Ny,Nx,Nl)**

**#on assigne à celle vers la droite (3eme noeud)**

**V[:,:,3] = 2.3**

**Cylindre = np.full((Ny,Nx), False) # False : libre, True : obstacle**

**center = (Nx//4, Ny//2)**

**radius = 40**

**for y in range(Ny):**

**for x in range(Nx):**

**if dist(center[0],center[1],x,y) < radius :**

**Cylindre[y][x] = True**

**# img = mpimg.imread('airless.png')**

**# for y in range(Ny):**

**# for x in range(Nx):**

**# if img[y][x][3] != 0:**

**# Cylindre[y][x] = True**

**# solve**

**for time in range(Nt):**

**#boundaries conditions**

**V[:,-1, [6,7,8]] = V[:,-2, [6,7,8]]**

**V[:,0, [2,3,4]] = V[:,1, [2,3,4]]**

**#V[0,:, [8,1,2]] = V[1,:, [8,1,2]]**

**#V[-1,:, [6,5,4]] = V[-2,:, [6,5,4]]**

**for i,cx,cy in zip(range(Nl),cxs,cys):**

**V[:,:, i] = np.roll(V[:,:, i], cx, axis = 1)**

**V[:,:, i] = np.roll(V[:,:, i], cy, axis = 0)**

**boundary = V[Cylindre,:]**

**# on inverse les vitesses**

**boundary = boundary[:, [0,5,6,7,8,1,2,3,4]]**

**rho = np.sum(V, 2) # on additionne les vitesses des noeuds**

**momentum\_x = np.sum(V\*cxs, 2) / rho**

**momentum\_y = np.sum(V\*cys, 2) / rho**

**V[Cylindre, :] = boundary**

**momentum\_x[Cylindre] = 0**

**momentum\_y[Cylindre] = 0**

**#collisions**

**V\_eq = np.zeros(V.shape)**

**for i,cx,cy,w in zip(range(Nl), cxs, cys, weights):**

**V\_eq[:,:,i] = rho\*w \* ( 1 + 3 \* (cx\*momentum\_x + cy\*momentum\_y) + 9 \* (cx\*momentum\_x + cy\*momentum\_y)\*\*2 / 2 - 3 \* (momentum\_x\*\*2 + momentum\_y\*\*2)/2 )**

**V = V -(V-V\_eq)/tau**

**#plot**

**# Plot the vector field**

**if time%50 == 0:**

**dfydx = momentum\_x[2:, 1:-1] - momentum\_x[:-2, 1:-1]**

**dfxdy = momentum\_y[1:-1, 2:] - momentum\_y[1:-1, :-2]**

**curl = dfydx - dfxdy**

**#plt.imshow(np.sqrt(momentum\_x\*\*2 + momentum\_y\*\*2))**

**plt.imshow(curl, cmap="bwr")**

**plt.streamplot(X, Y, momentum\_x, momentum\_y, density=0.5, linewidth=0.1, arrowsize=1, arrowstyle='->', color='green')**

**#plt.imshow(Cylindre)**

**plt.savefig("res/res\_"+str(time//50)+".png")**

**#plt.savefig("res/0res\_airless\_"+str(time//50)+".png")**

**plt.pause(0.01)**

**plt.cla()**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**