Informatique

Dérivation – Euler

Euler pour le rebond de balles en 2D

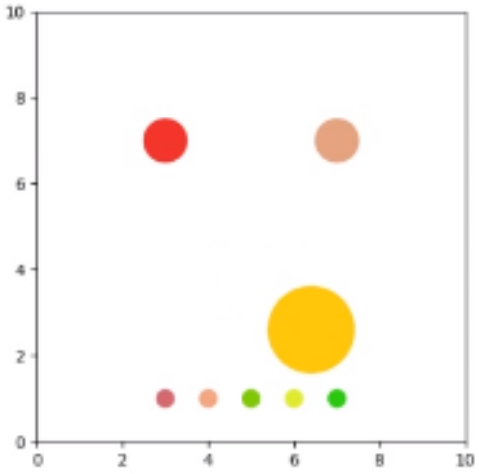
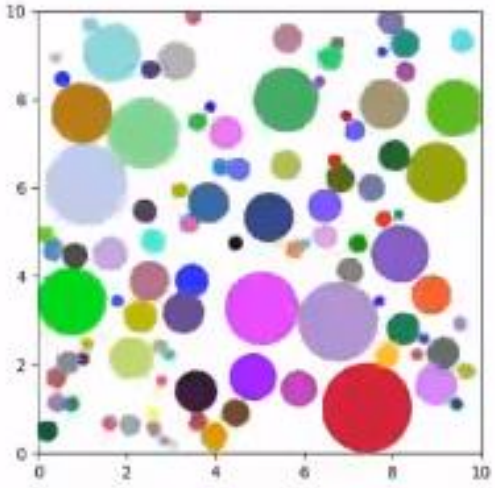
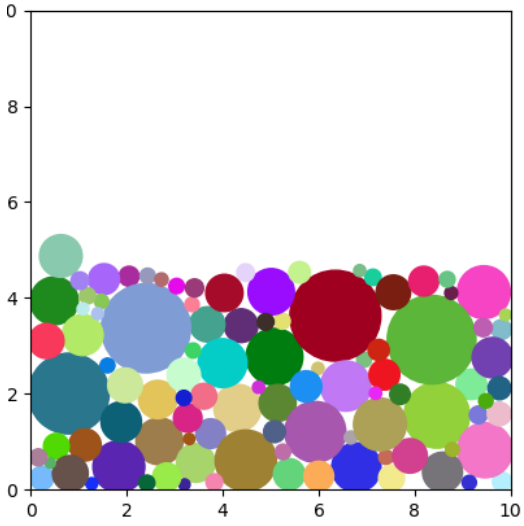
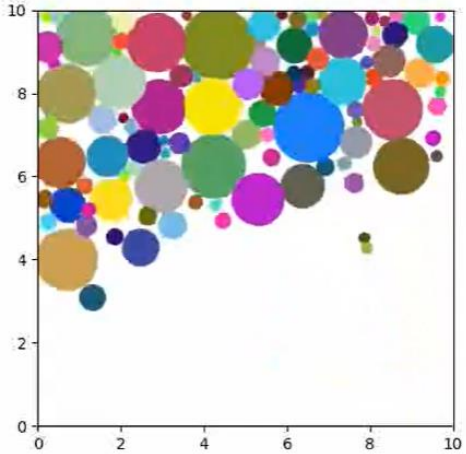
Contexte

En SII, la modélisation d'un système physique passe souvent par l'écriture d'équations différentielles issues du Principe Fondamental de la Dynamique. Résoudre ces équations analytiquement est rarement possible dès que le système devient complexe : c'est précisément là qu'intervient la simulation numérique.
Ce TP propose de construire pas à pas un simulateur de balles en 2D, en s'appuyant sur la méthode d'Euler. Cette méthode d'intégration numérique est au cœur des logiciels de simulation industriels (SolidWorks Motion ou Méca3D, Matlab/Simulink, etc.).
Le travail s'articule en trois étapes :

- Modélisation des efforts : traduire les contacts (balle/bord, balle/balle) en forces élastiques, comme on modéliserait un ressort de contact dans une liaison mécanique.
- Intégration numérique (Euler) : mettre à jour les accélérations, vitesses et positions à chaque pas de temps, à la manière d'un solveur dynamique.
- Visualisation : animer la simulation pour valider qualitativement le comportement du modèle, étape indispensable en bureau d'études avant tout prototypage physique.

Nous allons donc mettre en place un code simple permettant de simuler le mouvement de « balles » en tenant compte de chocs élastiques avec les contours d'un domaine rectangulaire et avec les autres balles. Nous ne tiendrons compte d'aucune dissipation énergétique et considérerons que la masse des balles est proportionnelle à leur rayon, autrement dit, ce sont plutôt des « palets » plats. Nous ne chercherons pas non plus à optimiser le code (en particulier sur les interactions entre balles). L'objectif est de vous proposer une méthode de résolution numérique simple qui permettra d'aller plus loin en TIPE par exemple.

Voici des illustrations en vidéo de ce que vous pourrez simuler avec le code développé dans ce sujet :

Résultat de ce TD	Simulation de 100 balles sans viscosité ni gravité
	
Vidéo	Vidéo
Simulation de 100 balles avec viscosité et gravité	Simulation de 100 balles avec viscosité et gravité tournante
	
Vidéo	Vidéo

Code élèves

Vous avez à disposition un code élèves dans lequel sont présents deux codes. Voici le premier :

```
from random import random as rd
from math import pi

class new_balle():

    def __init__(self,x,y,r):
        self.r = r                  # Rayon de la balle
        self.m = pi*r**2            # Masse de la balle
        self.c = [rd(),rd(),rd()] # Couleur de la balle
        self.x = x                  # Position x
        self.y = y                  # Position y
        self.lx = [x]               # Liste positions x
        self.ly = [y]               # Liste positions y
        self.vx = 0                 # Vitesse x
        self.vy = 0                 # Vitesse y
        self.ax = 0                 # Accélération x
        self.ay = 0                 # Accélération y
        self.fx = 0                 # Forces sur la basse x
        self.fy = 0                 # Forces sur la basse y

b1 = new_balle(5,4,1)
b2 = new_balle(3,7,0.5)
b3 = new_balle(7,7,0.5)
b4 = new_balle(3,1,0.2)
b5 = new_balle(4,1,0.2)
b6 = new_balle(5,1,0.2)
b7 = new_balle(6,1,0.2)
b8 = new_balle(7,1,0.2)
balles = [b1,b2,b3,b4,b5,b6,b7,b8] # Variable globale pour la suite
```

Ce code permet de créer des balles en utilisant l'un des avantages de la programmation objet, ce qui simplifiera grandement l'écriture de toute la suite. En effet, en créant une classe « **`new_balle`** », il est possible de créer autant de balles que l'on veut en appelant « `bi = new_balle(x,y,r)` » avec x et y sa position initiale, et r son rayon. Chacune possèdera des paramètres à mettre à jour en y accédant très simplement en écrivant par exemple « `bi.x += 2` » pour la déplacer de 2 m sur $\vec{x}$ ou « `bi.vx = 2` » pour lui imposer une vitesse selon $\vec{x}$ de 2 m/s.

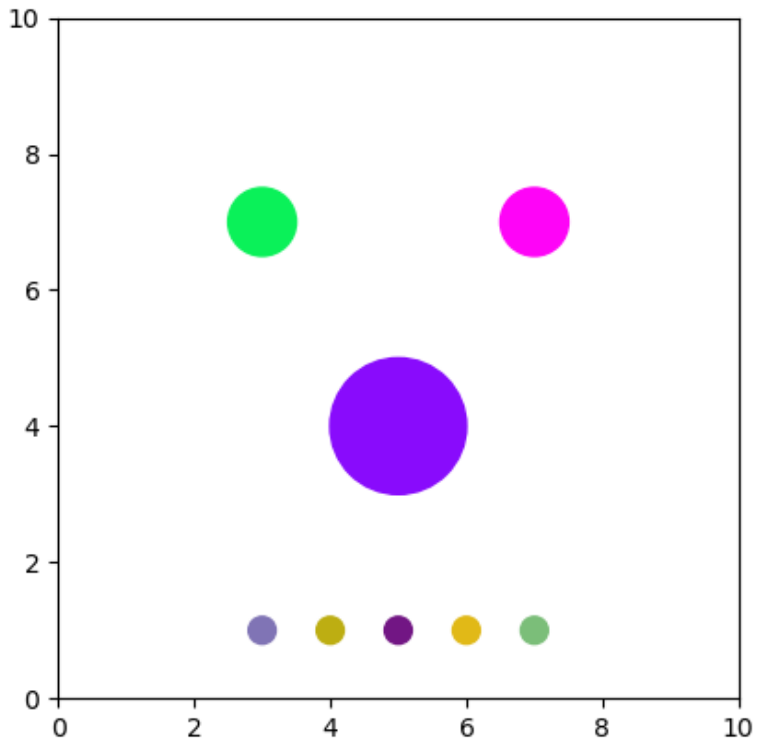
On propose ensuite le code suivant, affichant sur la figure 1 le domaine d'étude défini par la position de ses bords gauche, bas, haut et droit (`g,b,h,d = 0,0,10,10`) et toutes les balles :

```
import matplotlib.pyplot as plt
plt.close('all')

def affiche(fig,balles):
    plt.figure(fig)
    plt.clf()
    plt.xlim(g,d)
    plt.ylim(b,h)
    plt.axis('scaled')
    for balle in balles:
        Circle = plt.Circle((balle.x,balle.y),balle.r,color=balle.c)
        plt.gca().add_artist(Circle)
    plt.show()
    plt.pause(0.01)

g,b,h,d = 0,0,10,10 # Variables globales pour la suite
affiche(1,balles)
```

A l'exécution du code élèves complet, vous obtiendrez la figure suivante (aux couleurs près) :



Forces sur les balles

Pour tenir compte des chocs élastiques des balles avec les bords et avec les autres balles, nous allons calculer un écrasement e puis imposer une force élastique pour chaque contact proportionnelle à l'écrasement sous la forme :

$$F = \pm ke$$

Pour toutes les balles b_i, avec g, b, d et h les limites du domaine, on peut facilement exprimer l'écrasement avec le bord :

- Gauche : $e_g = f\big((b_i.x - b_i.r) - g\big)$ et $F_g = ke_g\vec{x}$
- Bas : $e_d = f\big(d - (b_i.x + b_i.r)\big)$ et $F_d = -ke_d\vec{x}$
- Droit : $e_b = f\big((b_i.y - b_i.r) - b\big)$ et $F_b = ke_b\vec{y}$
- Haut : $e_h = f\big(h - (b_i.y + b_i.r)\big)$ et $F_h = -ke_h\vec{y}$

Soient deux balles b_i et b_j et la fonction $f(x) = \begin{cases} 0\ si\ x \geq 0 \\ -x\ si\ x < 0 \end{cases}$

Lorsque deux balles entrent en contact, la distance entre leurs centres vaut : $d_{min} = b_i.r + b_j.r$

La distance entre les centres des balles à tout instant vaut :

$$d_{ij} = \sqrt{\big(b_j.x - b_i.x\big)^2 + \big(b_j.y - b_i.y\big)^2} = \sqrt{{d_{ijx}}^2 + {d_{ijy}}^2}$$

On peut alors exprimer l'écrasement entre deux balles ainsi : $e_{ij} = f(d - d_{min})$

En appelant $\overrightarrow{n_{ij}} = \big(n_{ijx}, n_{ijy}\big)$ le vecteur directeur unitaire de b_i vers b_j, on a : $\begin{cases} n_{ijx} = \frac{d_{ijx}}{d_{ij}} \\ n_{ijy} = \frac{d_{ijy}}{d_{ij}} \end{cases}$

Soit finalement l'action de b_j sur b_i : $F_{j\to i} = -ke_{ij}\overrightarrow{n_{ij}}$

Question 1: Créer la fonction f(x)

Pour chaque balle b_i, les efforts $b_i.fx$ et $b_i.fy$ seront réinitialisées à 0 à chaque itération (pas de temps) et actualisées par les fonctions d'efforts proposées dans la suite.

Question 2: Créer une fonction bords(balle) qui ajoute aux composantes fx et fy de la balle les actions des 4 bords

Question 3: Créer une fonction contacts(balle) qui ajoute aux composantes fx et fy de la balle les actions de toutes les autres balles

Remarque : On veillera à exclure la balle concernée qui a une distance nulle avec elle-même.

Question 4: Créer une fonction forces(balle) réinitialisant fx et fy, et appelant les fonctions précédentes afin de tenir compte de toutes les actions extérieures sur la balle concernée

Résolution d'Euler

On rappelle ce que donne le PFD en référentiel Galiléen appliqué à chaque balle b_i:

$$\begin{cases} b_i.fx = b_i.m * b_i.ax \\ b_i.fy = b_i.m * b_i.ay \end{cases}$$

Par ailleurs : $\begin{cases} \frac{db_i.x}{dt} = b_i.vx \\ \frac{db_i.y}{dt} = b_i.vy \end{cases}$

On appelle dt le pas de temps de la simulation qui sera considéré comme une variable globale constante dans la suite.

Question 5: Créer une fonction maj_avx(balle) qui met à jour pour la balle concernée ax&ay, puis vx,&vy, puis x&y et enfin stocke x dans lx et y dans ly

Simulation

On propose l'initialisation suivante :

```
k = 1000 # variable globale
tf = 20
t = 0
i = 0
lt = [t]
dt = 0.001 # Variable globale
b1.vx = 2
b1.vy = -2
```

On pourra par la suite faire évoluer les paramètres choisis.

Question 6: Mettre en place un code de simulation qui simule le mouvement des balles pendant 10 secondes sans affichage

Il nous reste à créer l'animation de la simulation, les coordonnées des balles b étant stockées pour chaque pas de temps dans les listes $b.lx$ et $b.ly$. Pour avoir un rendu intéressant, il ne faut pas afficher toutes les images mais seulement, par exemple, une image toutes les 50 images de simulation, soit une image toutes les 0.05 s. Pour créer l'affichage, il suffit de mettre à jour les valeurs $b.x$ et $b.y$ de toutes les balles à un pas de temps, puis à appeler la fonction d'affichage précédente.

Question 7: Proposer le code permettant d'animer la simulation

Vous pourrez maintenant ajouter une fonction de gravité, une fonction de force de dissipation visqueuse (etc.) pour voir ce que cela donne, ajouter des balles… Amusez-vous bien !