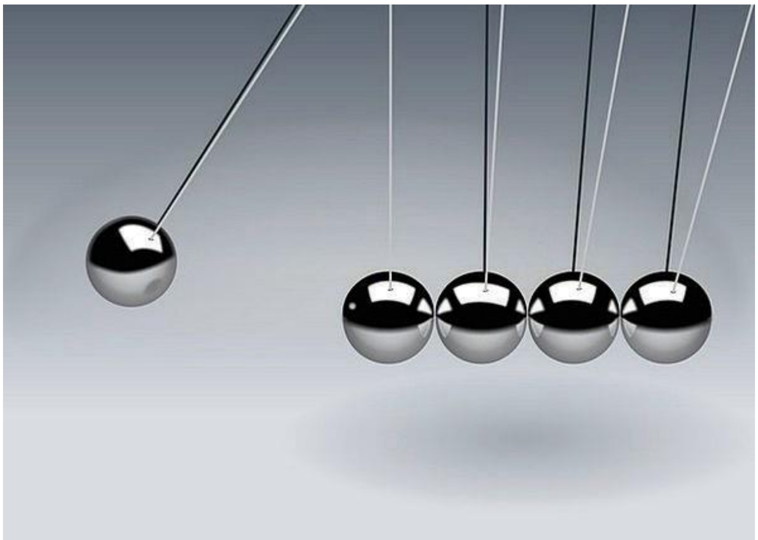
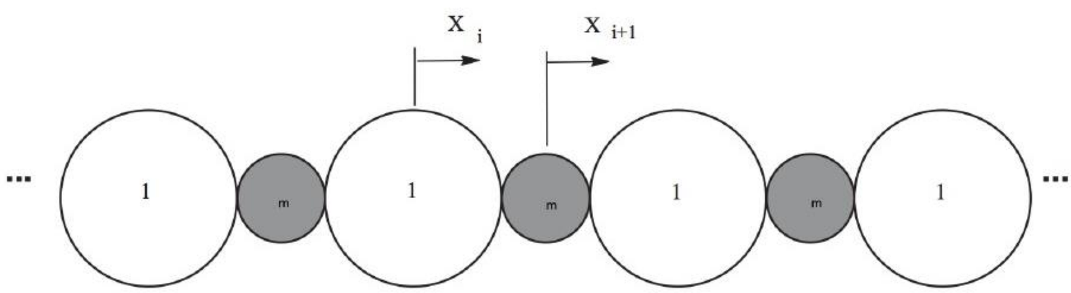
SIMULATION DE COLLISIONS DE BILLES

Simon Bigot, Yevann Randriamora
G4





Simulation de la collision de deux billes par le schéma d'Euler explicite :

On modélise la collision de deux billes identiques par le système :

$$\boxed{\begin{aligned} \frac{d^2x_1}{dt^2} &= -(x_1 - x_2)_+^{\frac{3}{2}} \quad (1) \\ \frac{d^2x_2}{dt^2} &= (x_1 - x_2)_+^{\frac{3}{2}} \quad (2) \end{aligned}}$$

Où $x_i(t)$ est le déplacement de la $i$ème bille par rapport à une position de référence (les billes sont tangentes lorsque $x_1 = x_2$).

$$(a)_+ = \max(a, 0)$$

La puissance $\frac{3}{2}$ vient du fait qu'on a des billes sphériques.

On note $v_i = \frac{dx_i}{dt}$ les vitesses des billes.

Question 1 :

On définit l'énergie mécanique du système en fonction des déplacements et des vitesses des billes de la manière suivante :

$$\boxed{H = \frac{v_1^2}{2} + \frac{v_2^2}{2} + \frac{2}{5}(x_1 - x_2)_+^{\frac{5}{2}} \quad (3)}$$

$\max(x, 0)^{\frac{5}{2}} \in \mathcal{C}^2$

Soit (x_1, x_2) solution de $(1) - (2)$:

$$\begin{aligned} \frac{dH}{dt} &= \frac{1}{2} \cdot \frac{dv_1^2}{dt} + \frac{1}{2} \cdot \frac{dv_2^2}{dt} + \frac{2}{5}\frac{d}{dt}(x_1 - x_2)_+^{\frac{5}{2}} \\ &= 2v_1 \frac{dv_1}{dt} + 2v_2 \frac{dv_2}{dt} + \frac{2}{5} + \left(\frac{5}{2}\frac{d}{dt}(x_1 - x_2)(x_1 - x_2)^{\frac{3}{2}}\right)_+ \\ &= 2v_1 \frac{dv_1}{dt} + 2v_2 \frac{dv_2}{dt} + v_1(x_1 - x_2)_+^{\frac{3}{2}} - v_2(x_1 - x_2)_+^{\frac{3}{2}} \\ &= 2v_1\left(\underbrace{\frac{dv_1}{dt} + v_1(x_1 - x_2)_+^{\frac{3}{2}}}_{0}\right) + 2v_2\left(\underbrace{\frac{dv_2}{dt} - v_2(x_1 - x_2)_+^{\frac{3}{2}}}_{0}\right) \\ &= 0 \end{aligned}$$

Conclusion :

Pour toute solution de $(1) - (2)$ on a $\frac{dH}{dt} = 0$.

Question 2 :

$F: \mathbb{R}^4 \to \mathbb{R}^4$

$$Y = \begin{pmatrix} x_1 \\ x_2 \\ v_1 \\ v_2 \end{pmatrix} \to \begin{pmatrix} F_1(Y) \\ F_2(Y) \\ F_3(Y) \\ F_4(Y) \end{pmatrix}$$

$$\boxed{\frac{dY}{dt} = F(Y) \quad (4)}$$

$$\Leftrightarrow \begin{cases} \frac{dx_1}{dt} = F_1 \\ \frac{dx_2}{dt} = F_2 \\ \frac{dv_1}{dt} = F_3 \\ \frac{dv_2}{dt} = F_4 \end{cases} \Leftrightarrow \begin{cases} F_1 = v_1 \\ F_2 = v_2 \\ F_3 = -(x_1 - x_2)_+^{\frac{3}{2}} \\ F_4 = (x_1 - x_2)_+^{\frac{3}{2}} \end{cases}$$

Conclusion :

$F: \mathbb{R}^4 \to \mathbb{R}^4$

$$Y = \begin{pmatrix} x_1 \\ x_2 \\ v_1 \\ v_2 \end{pmatrix} \to \begin{pmatrix} v_1 \\ v_2 \\ -(x_1 - x_2)_+^{\frac{3}{2}} \\ (x_1 - x_2)_+^{\frac{3}{2}} \end{pmatrix}$$

Question 3 :

Fonction qui calcule par le schéma d'Euler explicite la solution de l'équation (4) pour une condition initiale $Y(0) = Y^{(0)}$, pour $t \in [0, T]$, avec un pas d'intégration $h = \frac{T}{N}$.

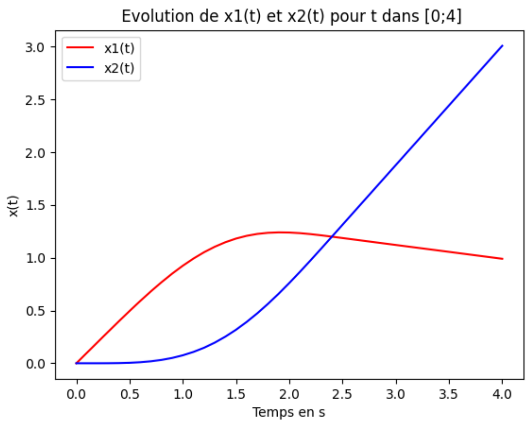
```
#! /usr/bin/env python3
import numpy as np
import matplotlib.pyplot as plt


def F(Yi):
    x1, x2, v1, v2 = Yi[0], Yi[1], Yi[2], Yi[3]
    Yip1 = np.array([v1, v2, -max(x1 - x2, 0)**(3/2), max(x1 - x2, 0)**(3/2)])
    return Yip1


def schema_euler_explicite(Y0, T, N):
    """
    Schéma d'euler explicite
    Renvoie une matrice de taille N+1
    T : temps maximal d'intégration
    N : nombre de points de discrétisation
    """
    #print("Y0 = ",Y0)
    Y = np.zeros((4, N+1))
    Y[:,0] = Y0
    #print("Y=",Y)
    h = T / N

    for k in range(1, N+1):
        Yip1 = F(Y[:, k - 1]) * h + Y[:,k - 1]
        Y[:,k] = Yip1
    #print(Y)
    return Y

def main():
    Y0 = np.array([0, 0, 1, 0])
    T = 4

    lN = [40, 400, 4000]

    for N in lN:
        fig = plt.figure(' positions pour {} points'.format(N))
        lT = np.linspace(0, 4, N+1)
        Y = schema_euler_explicite(Y0, T, N)
        # GRAPHE DES POSITIONS
        print("Erreur des vitesses = ",abs(Y[3, -1] - Y0[2]))
        plt.title(' Evolution de x1(t) et x2(t) pour t dans [0;4]')
        plt.plot(lT, Y[0,:], 'r', label ="x1(t)")
        plt.plot(lT, Y[1, :], 'b', label = "x2(t)")
        plt.xlabel("Temps en s")
        plt.ylabel("x(t)")
        #plt.legend(["x1(t), x2(t)"])
        plt.legend()
        plt.show()
        fig = plt.figure('vitesses pour {} points'.format(N))
        # GRAPHE DES VITESSES
        plt.title(' Evolution de v1(t) et v2(t) pour t dans [0;4]')
        plt.plot(lT, Y[2, :], 'r', label = "v1(t)")
```

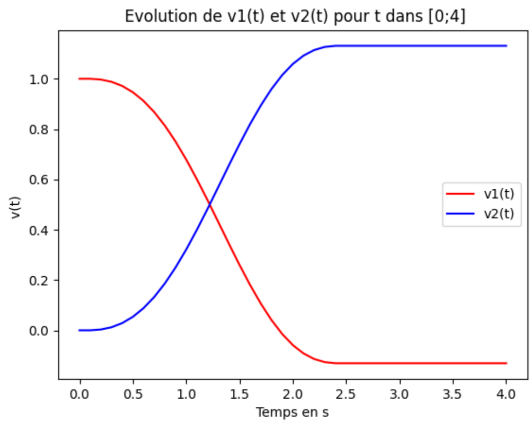
```
        plt.plot(lT, Y[3,:], 'b', label = "v2(t)")
        plt.xlabel("Temps en s")
        plt.ylabel("v(t) ")
        #plt.legend(["v1(t), v2(t)"])
        plt.legend()

        plt.show()

if __name__=="__main__":
    main()
```

Question 4 :

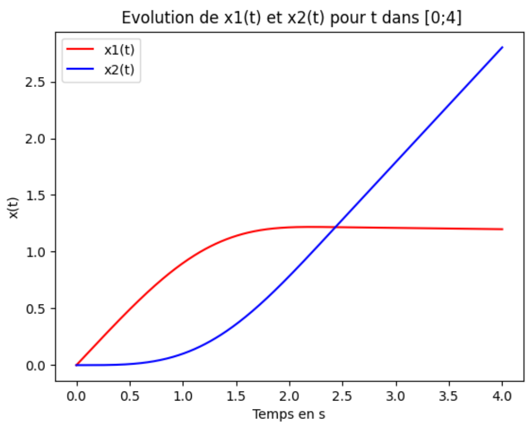
Calcul de la solution avec la condition initiale : $x_1(0) = x_2(0) = 0, v_1(0) = 1, v_2(0) = 0, t \in [0,4], h \in \{10^{-1}, 10^{-2}, 10^{-3}\}$

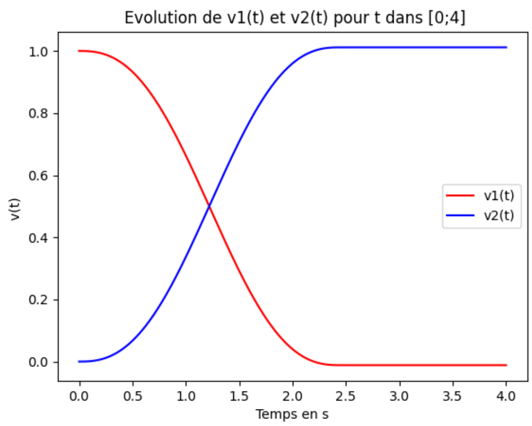
Cas $h = 10^{-1}$:



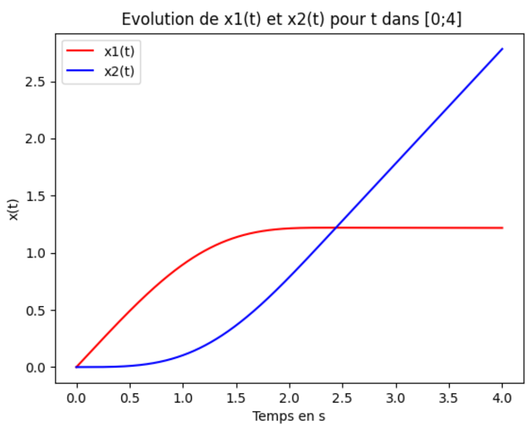


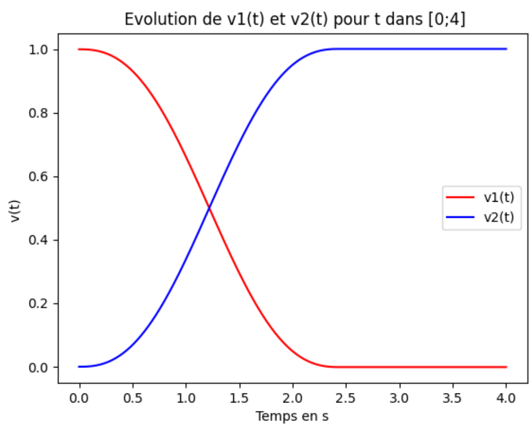
Cas $h = 10^{-2}$:





Cas $h = 10^{-3}$:





Vérification $v_{2,finale} > v_1(0)$:

```
[[ 0.00000000e+00  1.00000000e-02  2.00000000e-02 ...  1.19882055e+00
   1.19870482e+00  1.19858909e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00 ...  2.78117945e+00
   2.79129518e+00  2.80141091e+00]
 [ 1.00000000e+00  1.00000000e+00  9.99990000e-01 ... -1.15727257e-02
  -1.15727257e-02 -1.15727257e-02]
 [ 0.00000000e+00  0.00000000e+00  1.00000000e-05 ...  1.01157273e+00
   1.01157273e+00  1.01157273e+00]]
[[ 0.00000000e+00  1.00000000e-03  2.00000000e-03 ...  1.21735838e+00
   1.21735724e+00  1.21735610e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00 ...  2.78064162e+00
   2.78164276e+00  2.78264390e+00]
 [ 1.00000000e+00  1.00000000e+00  9.99999968e-01 ... -1.14305397e-03
  -1.14305397e-03 -1.14305397e-03]
 [ 0.00000000e+00  0.00000000e+00  3.16227766e-08 ...  1.00114305e+00
   1.00114305e+00  1.00114305e+00]]
randriamora@randriye  ~/Documents/COURS/ENSIMAG/1A/S2/MNB/TP_MNB  Q
```

Au-delà d'un certain temps, les vitesses prennent des valeurs constantes.
On observe $v_{2,finale} > v_1(0)$ ce qui est physiquement irréaliste.

Observation pour $v_{2,finale} - v_1(0)$ lorsque $h \to 0$:

```
Erreur des vitesses =  0.1310192986671248
Erreur des vitesses =  0.011572725732442946
Erreur des vitesses =  0.001143053973849284
```

Lorsque $h \to 0$ on observe que $v_{2,finale} - v_1(0)$ tend vers 0.

Simulation de la collision de n billes par un modèle linéarisé :

On considère l'équation différentielle linéaire dans $\mathbb{R}^n$:

$$\boxed{m_1 \frac{d^2 u_1}{dt^2} = u_2 - u_1 + f_1(t) \quad (5)}$$

$$2 \le i \le n-1 \; \boxed{m_i \frac{d^2 u_i}{dt^2} = u_{i+1} - 2u_i + u_{i-1} \quad (6)}$$

$$\boxed{m_n \frac{d^2 u_n}{dt^2} = u_{n-1} - 2u_n \quad (7)}$$

avec $m_{2p+1} = 1$ et $m_{2p} = m$, où $m \in]0,1]$ désigne un paramètre. Les solutions $u(t) = \left(u_1(t), \dots, u_n(t)\right)^T \in \mathbb{R}^n$ représentent (dans une approximation linéaire) les déplacements des billes dans une chaîne fortement comprimée aux deux extrémités. La première bille est soumise à une force extérieure $f_1(t)$ dont la valeur sera précisée plus loin.

On approche $(5)-(6)-(7)$ par le schéma d'Euler implicite avec un pas de discrétisation h, pour $t \in [0, T]$ avec $T = Nh$.
On note $u_i^{(k)}$ l'approximation numérique de $u_i(kh)$, $v_i^{(k)} = \frac{u_i^{(k)} - u_i^{(k-1)}}{h}$ celle de la vitesse $\frac{du_i}{dt}(kh)$, et $u^{(k)} = \left(u_1^{(k)}, \dots, u_n^{(k)}\right)^T \in \mathbb{R}^n$.

Le schéma s'écrit :

$$\boxed{m_1 \frac{u_1^{(k+1)} - 2u_1^{(k)} + u_1^{(k-1)}}{h^2} = u_2^{(k+1)} - u_1^{(k+1)} + f_1\big((k+1)h\big) \quad (8)}$$

$$2 \le i \le n-1 \; \boxed{m_i \frac{u_i^{(k+1)} - 2u_i^{(k)} + u_i^{(k-1)}}{h^2} = u_{i+1}^{(k+1)} - 2u_i^{(k+1)} + u_{i-1}^{(k+1)} \quad (9)}$$

$$\boxed{m_n \frac{u_n^{(k+1)} - 2u_n^{(k)} + u_n^{(k-1)}}{h^2} = u_{n-1}^{(k+1)} - 2u_n^{(k+1)} \quad (10)}$$

On note $M \in M_n(\mathbb{R})$ la matrice diagonale de coefficients $m_1, \dots, m_n$.

Question 5 :

Le schéma $(8)-(9)-(10)$ s'écrit :

$$\begin{cases} m_1 \dfrac{u_1^{(k+1)} - 2u_1^{(k)} + u_1^{(k-1)}}{h^2} = u_2^{(k+1)} - u_1^{(k+1)} + f_1\big((k+1)h\big) \\ m_i \dfrac{u_i^{(k+1)} - 2u_i^{(k)} + u_i^{(k-1)}}{h^2} = u_{i+1}^{(k+1)} - 2u_i^{(k+1)} + u_{i-1}^{(k+1)} \quad (2 \le i \le n-1) \\ m_n \dfrac{u_n^{(k+1)} - 2u_n^{(k)} + u_n^{(k-1)}}{h^2} = u_{n-1}^{(k+1)} - 2u_n^{(k+1)} \end{cases}$$

$$\Leftrightarrow \begin{cases} m_1\left(u_1^{(k+1)} - 2u_1^{(k)} + u_1^{(k-1)}\right) = h^2\left(u_2^{(k+1)} - u_1^{(k+1)} + f_1\big((k+1)h\big)\right) \\ m_i\left(u_i^{(k+1)} - 2u_i^{(k)} + u_i^{(k-1)}\right) = h^2\left(u_{i+1}^{(k+1)} - 2u_i^{(k+1)} + u_{i-1}^{(k+1)}\right) \quad (2 \le i \le n-1) \\ m_n\left(u_n^{(k+1)} - 2u_n^{(k)} + u_n^{(k-1)}\right) = h^2\left(u_{n-1}^{(k+1)} - 2u_n^{(k+1)}\right) \end{cases}$$

$$\Leftrightarrow \begin{cases} m_1 u_1^{(k+1)} + h^2\left(u_1^{(k+1)} - u_2^{(k+1)}\right) & = m_1\left(u_1^{(k)} + hv_1^{(k)}\right) + h^2 f_1\left((k+1)h\right) \\ m_i u_i^{(k+1)} + h^2\left(-u_{i-1}^{(k+1)} + 2u_i^{(k+1)} - u_{i+1}^{(k+1)}\right) & = m_i\left(u_i^{(k)} + hv_i^{(k)}\right) \\ m_n u_n^{(k+1)} + h^2\left(-u_{n-1}^{(k+1)} + 2u_n^{(k+1)}\right) & = m_n\left(u_n^{(k)} + hv_n^{(k)}\right) \end{cases}$$

$$\Leftrightarrow \underbrace{\begin{pmatrix} m_1 & 0 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & m_i & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & m_n \end{pmatrix}}_{M} \underbrace{\begin{pmatrix} u_1^{(k+1)} \\ \vdots \\ u_i^{(k+1)} \\ \vdots \\ u_n^{(k+1)} \end{pmatrix}}_{u^{(k+1)}} + h^2 \underbrace{\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}}_{D} \underbrace{\begin{pmatrix} u_1^{(k+1)} \\ \vdots \\ u_i^{(k+1)} \\ \vdots \\ u_n^{(k+1)} \end{pmatrix}}_{u^{(k+1)}} = \underbrace{\begin{pmatrix} m_1\left(u_1^{(k)} + hv_1^{(k)}\right) + h^2 f_1\left((k+1)h\right) \\ m_2\left(u_2^{(k)} + hv_2^{(k)}\right) \\ \vdots \\ m_i\left(u_i^{(k)} + hv_i^{(k)}\right) \\ \vdots \\ m_n\left(u_n^{(k)} + hv_n^{(k)}\right) \end{pmatrix}}_{b^{(k)}}$$

$$Mu^{(k+1)} + h^2 Du^{(k+1)} = b^{(k)}$$
$$Au^{(k+1)} = b^{(k)}$$

Conclusion :

Ainsi, le schéma $(8) - (9) - (10)$ s'écrit :

$$\boxed{Au^{(k+1)} = b^{(k)} \quad (11)}$$

Question 6 :

$A = M + h^2 D$

M et $h^2 D$ sont symétriques donc A est symétrique.

M est diagonale donc ses valeurs propres sont ses éléments diagonaux (les m_i), or $\forall i, m_i > 0$ donc M est définie positive (Une matrice $M \in M_n(\mathbb{R})$ symétrique est définie positive si et seulement si toutes ses valeurs propres sont > 0).

Soit $x = (x_1, \dots, x_n)^T \in \mathbb{R}^n$ non nul.

$$x^T D x = \begin{pmatrix} x_1 & \dots & x_i & \dots & x_n \end{pmatrix} \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{pmatrix}$$

$$= \begin{pmatrix} x_1 & \dots & x_i & \dots & x_n \end{pmatrix} \begin{pmatrix} x_1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ \vdots \\ -x_{i-1} + 2x_i - x_{i+1} \\ \vdots \\ -x_{n-1} + 2x_n \end{pmatrix}$$

$$= x_1(x_1 - x_2) + x_2(-x_1 + 2x_2 - x_3) + \dots + x_i(-x_{i-1} + 2x_i - x_{i+1}) + \dots + x_n(-x_{n-1} + 2x_n)$$

$$= x_1^2 - x_1 x_2 + \sum_{i=2}^{n-1} \left(2x_i^2 - x_{i-1}x_i - x_{i+1}x_i\right) - x_{n-1}x_n + 2x_n^2$$

$$= x_1^2 + 2\sum_{i=2}^{n} x_i^2 - \sum_{i=2}^{n} x_{i-1}x_i - \sum_{i=1}^{n-1} x_{i+1}x_i$$

$$= x_1^2 + 2\left(\sum_{i=2}^{n} x_i^2 - x_{i-1}x_i\right) = \underbrace{\sum_{i=2}^{n} (x_{i-1} - x_i)^2 + x_n^2}_{Voir\ récurrence} > 0$$

Récurrence :

$$\forall n \geq 2, P(n) : x_1^2 + 2\left(\sum_{i=2}^{n} x_i^2 - x_{i-1}x_i\right) = \sum_{i=2}^{n} (x_{i-1} - x_i)^2 + x_n^2$$

- **Initialisation ($n = 2$) :**

$$x_1^2 + 2\left(\sum_{i=2}^{2} x_i^2 - x_{i-1}x_i\right) = x_1^2 + 2(x_2^2 - x_1 x_2) = (x_1 - x_2)^2 + x_2^2$$

$$\sum_{i=2}^{2} (x_{i-1} - x_i)^2 + x_2^2 = (x_1 - x_2)^2 + x_2^2$$

Donc $P(2)$ est vraie.

- **Hérédité :** soit $k > 2$ tel que $P(n)$ soit vrai.

$$x_1^2 + 2\left(\sum_{i=2}^{n+1} x_i^2 - x_{i-1}x_i\right) = \underbrace{x_1^2 + 2\left(\sum_{i=2}^{n} x_i^2 - x_{i-1}x_i\right)}_{HR} + 2(x_{n+1}^2 - x_n x_{n+1})$$

$$= \sum_{i=2}^{n} (x_{i-1} - x_i)^2 + x_n^2 + 2x_{n+1}^2 - 2x_n x_{n+1}$$

$$= \sum_{i=2}^{n+1} (x_{i-1} - x_i)^2 + x_{n+1}^2$$

Donc $P(n+1)$ est vraie.

- **Conclusion :**

$$x_1^2 + 2\left(\sum_{i=2}^{n} x_i^2 - x_{i-1}x_i\right) = \sum_{i=2}^{n} (x_{i-1} - x_i)^2 + x_n^2$$

Donc $h^2 D$ est strictement définie positive.

Conclusion :

Ainsi M et $h^2 D$ sont strictement définies positives donc par somme de matrice strictement définies positives, A est symétrique définie positive.

Théorème (Cholesky) :

Soit A une matrice symétrique définie positive. Alors il existe une unique matrice triangulaire inférieure L telle que $A = LL^T$ et $L_{ii} > 0$ pour tout i.

Comme la matrice A de (11) est tridiagonale, seuls les coefficients diagonaux et sous-diagonaux de L sont non nuls. Ainsi, pour gagner en efficacité, on stockera la matrice L dans deux vecteurs (un de taille n pour les coefficients diagonaux et un de taille $n-1$ pour les coefficients sous-diagonaux).

Question 7 :

$$L = \begin{pmatrix} d_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ sd_1 & \ddots & 0 & 0 & 0 & 0 & 0 \\ 0 & \ddots & \ddots & 0 & 0 & 0 & 0 \\ 0 & 0 & sd_{i-1} & d_i & 0 & 0 & 0 \\ 0 & 0 & 0 & sd_i & \ddots & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & sd_{n-1} & d_n \end{pmatrix}$$

$$A = LL^T = \begin{pmatrix} d_1^2 & sd_1.d_1 & 0 & 0 & 0 & 0 & 0 \\ sd_1.d_1 & sd_1^2 + d_2^2 & \ddots & 0 & 0 & 0 & 0 \\ 0 & sd_2.d_2 & \ddots & sd_{i-1}.d_i & 0 & 0 & 0 \\ 0 & 0 & \ddots & sd_{i-1}^2 + d_i^2 & \ddots & 0 & 0 \\ 0 & 0 & 0 & sd_i.d_i & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & \ddots & \ddots & sd_{n-1}.d_{n-1} \\ 0 & 0 & 0 & 0 & 0 & sd_{n-1}d_{n-1} & sd_{n-1}^2 + d_n^2 \end{pmatrix}$$

$$d_1 = \sqrt{a_{11}}$$

$$sd_1 = \frac{a_{12}}{d_1}$$

$$d_i = \sqrt{a_{ii} - sd_{i-1}^2}\ (\forall i \in [2, n])$$

$$sd_i = \frac{a_{i+1,i}}{d_i}\ (\forall i \in [2, n-1])$$

$$a_{ii} = diag[i] \to diag[i-1]\ en\ python$$

$$a_{i+1,i} = sous\_diag[i] \to sous\_diag[i-1]\ en\ python$$

```python
def factorise(diag, sous_diag):
    ldiag = []
    linf = []
    N = len(diag)
    ldiag.append(np.sqrt(diag[0]))
    linf.append( sous_diag[0] / ldiag[-1])
    for i in range(1, N - 1):
        ldiag.append( np.sqrt( diag[i] - linf[-1] ** 2) )
        linf.append(sous_diag[i] / ldiag[i])

    ldiag.append( np.sqrt( diag[ - 1] - sous_diag[-1] ** 2))
    return linf, ldiag
```

Une fois cette factorisation obtenue, la résolution d'un système $Au = b$ se décompose comme suit :

$$LL^T u = b \Leftrightarrow \begin{cases} Ly = b \\ L^T u = y \end{cases}$$

Question 8 :

$$L = \begin{pmatrix} d_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ sd_1 & \ddots & 0 & 0 & 0 & 0 & 0 \\ 0 & \ddots & \ddots & 0 & 0 & 0 & 0 \\ 0 & 0 & sd_{i-1} & d_i & 0 & 0 & 0 \\ 0 & 0 & 0 & sd_i & \ddots & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & sd_{n-1} & d_n \end{pmatrix}, y = \begin{pmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{pmatrix}, b = \begin{pmatrix} b_1 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{pmatrix}$$

$$Ly = b$$

$$\Leftrightarrow \begin{cases} b_1 = d_1 y_1 \\ b_i = sd_{i-1} y_{i-1} + d_i y_i \end{cases}$$

$$\Rightarrow \begin{cases} y_1 = \dfrac{b_1}{d_1} \\ y_i = \dfrac{b_i - sd_{i-1} y_{i-1}}{d_i} \end{cases} (car\ d_i \neq 0)$$

```python
def descente(linf, ldiag, b):
    y = [b[0] / ldiag[0]]
    N = len(ldiag)
    for i in range(1, N):
        y.append ( (b[i] - linf[i - 1] * y[-1]) / ldiag[i])
    return y
```

Question 9 :

$$L^T = \begin{pmatrix} d_1 & sd_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \ddots & \ddots & 0 & 0 & 0 & 0 \\ 0 & 0 & \ddots & sd_{i-1} & 0 & 0 & 0 \\ 0 & 0 & 0 & d_i & sd_i & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & \ddots & sd_{n-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & d_n \end{pmatrix}, u = \begin{pmatrix} u_1 \\ \vdots \\ u_i \\ \vdots \\ u_n \end{pmatrix}, y = \begin{pmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{pmatrix}$$

$$L^T u = y$$

$$\Leftrightarrow \begin{cases} y_1 = d_1 u_1 + sd_1 u_2 \\ y_i = d_i u_i + sd_i u_{i+1} \\ y_n = d_n u_n \end{cases}$$

$$\Rightarrow \begin{cases} u_n = \dfrac{y_n}{d_n} \\ u_i = \dfrac{y_{i-1} - d_{i-1} u_{i-1}}{sd_{i-1}} \end{cases}$$

```python
def remonte(linf, ldiag, y):
    u = [ y[-1] / ldiag[-1]]
    N = len(ldiag)
    for i in range(N - 1, 0, -1):
        u.append( (y[i - 1] - linf[i - 1] * u[-1]) / ldiag[i - 1])

    u.reverse()
    return u
```

Question 10 :

$Au^{(k+1)} = b^{(k)}$

$A = LL^T$

$LL^T u = b \Leftrightarrow \begin{cases} Ly = b \\ L^T u = y \end{cases}$

- factorise(diag, sous_diag) => pour obtenir L.
- descente(linf, ldiag) => pour obtenir y solution de $Ly = b$
- remonte(linf, ldiag, y) => pour obtenir u solution de $L^T u = y$

- compute_b(f1, u, h, m) => pour calculer $b^{(k)}$.
- compute_A(m, h, n) => pour calculer $b^{(k)}$.
- compute_speed(u, h) => pour calculer $v_i^{(k)}$ en fonction de h et $u^{(k)}$.

A faire une seul fois :

- factorise(diag, sous_diag) => pour obtenir L.
- Initialisation de $b^{(0)}$ (avec $u^{(0)}$ et $u^{(-1)}$)

A chaque itération :

- descente(linf, ldiag) => pour obtenir y solution de $Ly = b$
- remonte(linf, ldiag, y) => pour obtenir u solution de $L^T u = y$
- compute_b(f1, u, h, m) => pour calculer $b^{(k)}$.

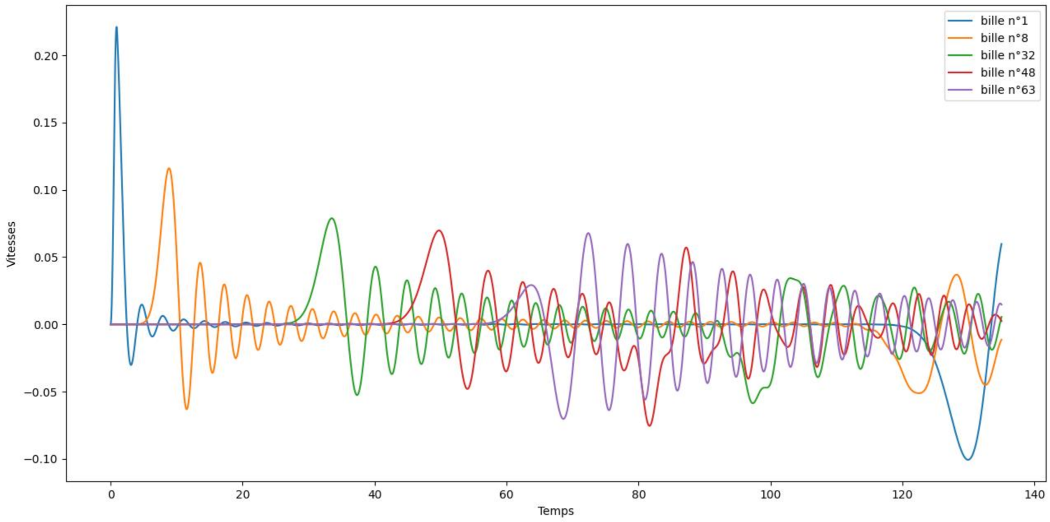
```python
def compute_b(f1, u, h, m, k):
    n = len(u[-1])
    b = [0 for _ in range(n)]
    uk = u[-1]
    ukm1 = u[-2]
    for i in range(n):

        if (i + 1) % 2 == 0:
            b[i] = m * (2 * uk[i] - ukm1[i])
        else:
            b[i] = (2 * uk[i] - ukm1[i])

    b[0] += (h ** 2) * f1((k+1) * h)
    return b

def compute_A(m, h, n):
    # CALCUL DIAG
    diag = [1 + h ** 2]
    for i in range(1, n):
        if (i + 1) % 2 == 0:
            diag.append(m + 2 * (h ** 2))
        else:
            diag.append( 1  + 2 * (h ** 2))
    # CALCUL LINF
    sous_diag = [ -(h ** 2) for _ in range(n - 1)]

    return sous_diag, diag

def compute_speed(u, h):
    n = len(u)
    v = []
    for i in range(1, n):
        v.append( (u[i] - u[i - 1]) / h)
    return v
```

```python
def sol(u0, um1, diag, sous_diag, h, m, f1, N):
    linf, ldiag = factorise(diag, sous_diag)
    #print("ldiag = ",ldiag)
    L = [um1, u0]
    b = compute_b(f1, L, h, m, 0)
    #print(b[1])
    for k in range(1, N + 1):

        y = descente(linf, ldiag, b)
        # if k == 1:
        #     print("y=",y)
        u = remonte(linf, ldiag, y)
        L.append(u)
        b = compute_b(f1, L, h,m, k)
    return L[2:]
```

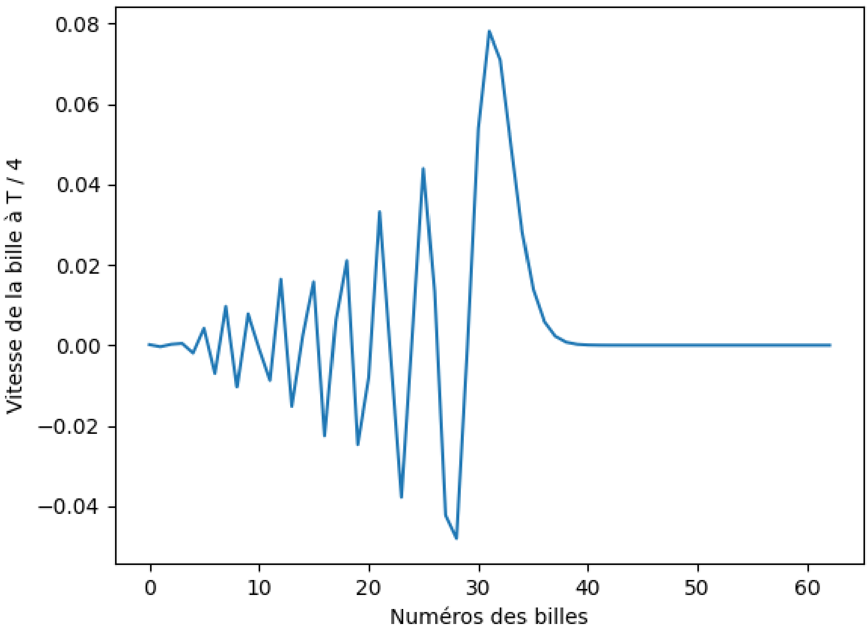
Question 11 :

```python
def f1(t):
    if t >= 0 and t <= 1 / 2 :
        return t
    elif t >= 1 / 2 and t <= 1:
        return 1 - t
    else:
        return 0

def compute_speed2(u, um1,  h):
    n = len(u)
    v = []
    for i in range( n):
        v.append( (u[i] - um1[i]) / h)
    return v

def main():
    m = 1
    n = 63
    T = 135
    h = 10**(-3)
    N = int(T / h)
    lt = [ k * h for k in range(N - 1)]
    u0 = [ 0 for _ in range(N)]
    um1 = [ 0 for _ in range(N)]
    sous_diag, diag = compute_A(m, h, n)
    U = sol(u0, um1, diag, sous_diag, h, m, f1, N)

    L1, L8, L32, L48, L63 = [], [], [], [], []
    for u in U:
        L1.append(u[0])
        L8.append(u[7])
        L32.append(u[31])
        L48.append(u[47])
        L63.append(u[62])

    V1 = compute_speed(L1, h)
    V8 = compute_speed(L8, h)
    V32 = compute_speed(L32, h)
    V48 = compute_speed(L48, h)
    V63 = compute_speed(L63, h)
```

```python
    fig = plt.figure()
    plt.plot(lt, V1)
    plt.plot(lt, V8)
    plt.plot(lt, V32)
    plt.plot(lt, V48)
    plt.plot(lt, V63)
    plt.legend(["bille n°1", "bille n°8", "bille n°32", "bille n°48", "bille n°63"])
    plt.xlabel("Temps")
    plt.ylabel("Vitesses")
    plt.show()

    ind = lt.index(T / 4)
    Lu = U[ind]
    Lum1 = U[ind -1]
    Lv = compute_speed2(Lu, Lum1,  h)

    fig = plt.figure()
    plt.plot(range(len(Lv)), Lv)
    plt.xlabel("Numéros des billes")
    plt.ylabel("Vitesse de la bille à T / 4")
    plt.show()

main()
```



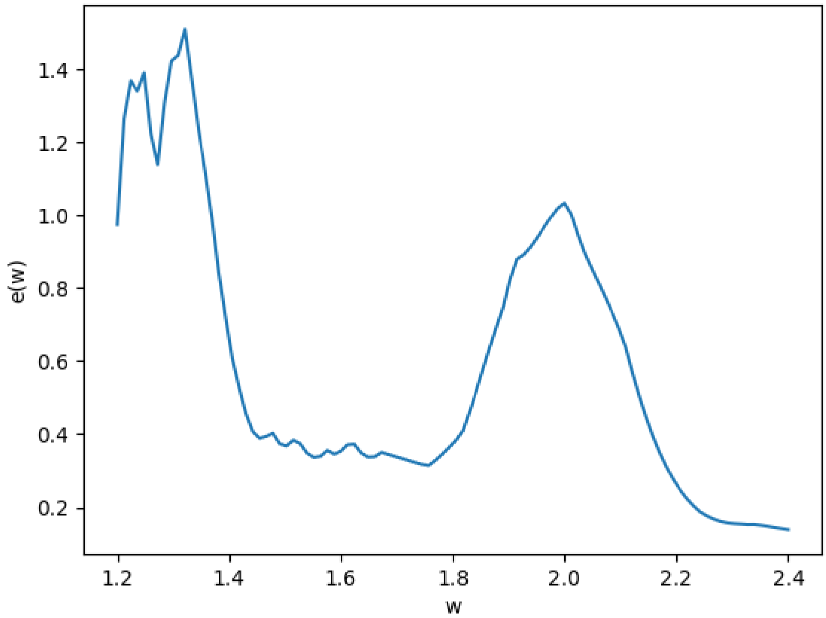
<u>Vitesses des billes en fonctions du temps</u>



<u>Vitesse au temps T/4 pour toutes les billes</u>

Question 12 :

```python
def main():
    m = 0.6
    n = 31
    T = 95
    h = 5 * (10 ** (-3))
    N = int(T / h)
    lt = [ k * h for k in range(N - 1)]
    u0 = [ 0 for _ in range(N)]
    um1 = [ 0 for _ in range(N)]
    sous_diag, diag = compute_A(m, h, n)


    lw = np.linspace(1.2, 2.4, 100)
    fig = plt.figure()
    ew = []
    for w in lw:
        U = sol(u0, um1, diag, sous_diag, h, m, f1, N, w)
        b1 = []
        bn = []
        for i in range(len(U)):
            b1.append( U[i][0])
            bn.append( U[i][-1])
        v1 = compute_speed(b1, h)
        vn = compute_speed(bn, h)
        max1 = max(v1)
        maxn = max(vn)
        ew.append(maxn / max1)
    plt.plot(lw, ew)
    plt.xlabel("w")
    plt.ylabel("e(w)")
    plt.show()

main()
```



Rapport $e(\omega)$ en fonction de ω

On observe un amortissement de la vitesse de la dernière bille par rapport à la vitesse de la première bille lorsque le paramètre de forçage augmente.

Le pic est là où il y a f_1 max.