Nonlinear Oscillations

Table of contents

Giải hệ phương trình ODE

Nonlinear Oscillations

Lý thuyết

Xét bài toán vật lý gắn vật vào lò xo với lực đàn hồi là $F_k(x)$. Có thêm trường ngoài tác động lên vật theo phương chuyển động:

$$F_k(x) + F_{ext}(x,t) = m\frac{d^2x}{dt^2}.$$

Xét thế năng tới bậc 3 của x, có dáng điệu là:

$$V(x) \approx \frac{1}{2}kx^2 - \frac{1}{3}kax^3,$$

với $ax \ll 1$; $1 < ax < 2$.

Ta tìm được $F_k(x)$ là:

$$F_k(x) = -\nabla V = -kx + kax^2$$

Thuật toán

Ta đặt:

$$\begin{cases} y_1 &= x(t) \\ y_2 &= \dfrac{dy_1}{dt} = \dfrac{dx}{dt} = v \\ y_3 &= \dfrac{dy_2}{dt} = \dfrac{d^2x}{dt^2} = \dfrac{-kx + kax^2}{m} \end{cases} \tag{1}$$

với $k = 2, m = 1, \omega = \sqrt{\frac{k}{m}}$.

Ta viết lại (1), dưới dạng ma trận trong python:

$$\begin{cases} y_1 = x(t), \\ f_1(t, y_1) = y_2 = \dfrac{dy_1}{dt} = \dfrac{dx}{dt} = v(t), \\ f_2(t, y_2) = \dfrac{df_1}{dt} = y_3 = \dfrac{dy_2}{dt} = \dfrac{d^2x}{dt^2} = \dfrac{-kx + kax^2}{m}. \end{cases} \tag{2}$$

với f_1, f_2 là các thành phần ma trận của $|f\rangle$ trong python.

$$\begin{pmatrix} \dot{x} \\ \ddot{x} \end{pmatrix} = \begin{pmatrix} f_1(t, x(t)) \\ f_2(t, \dot{x}(t)) \end{pmatrix} \xrightarrow{\text{Runge-Kutta 4}} x, \dot{x}$$

Ta có điều kiện đầu cho $x, \dot{x}$ là 1,0

Source code

```
import numpy as np
from numpy import typing as npt, pi, sqrt, sin
```

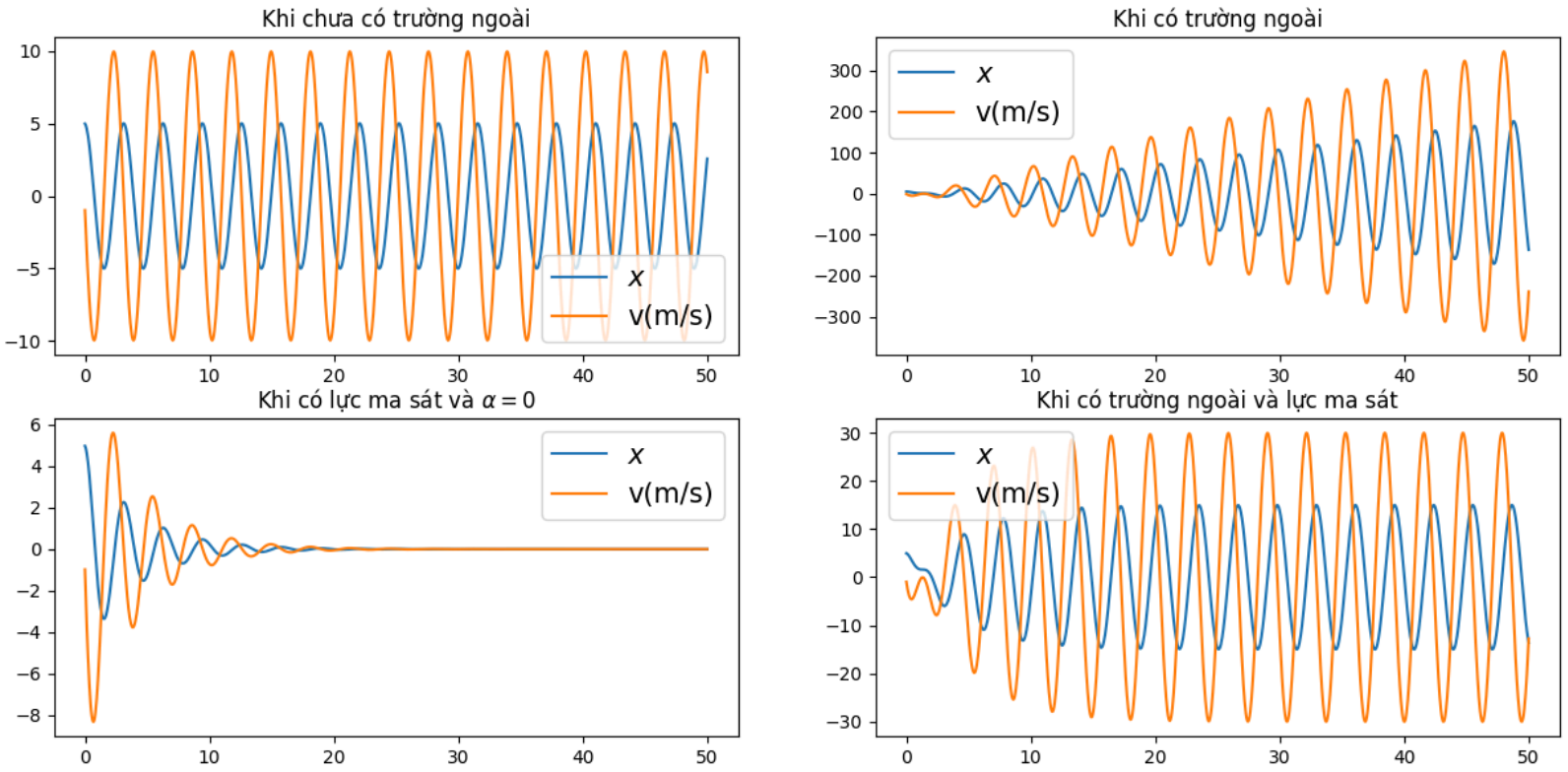
```
import matplotlib.pyplot as plt
import csv

alphax = 0.01
k = 2
m = 0.5
omega0 = sqrt(k / m)
b = 0.5 * m * omega0


def FArr(t: npt.NDArray, initInput: npt.NDArray) -> npt.NDArray:
    x, velocity = initInput

    F = np.zeros(2)
    F[0] = velocity
    F[1] = -(k * x - k * x * alphax) / m

    return F

def FArrExt(t: npt.NDArray, initInput: npt.NDArray) -> npt.NDArray:
    x, velocity = initInput

    F = np.zeros(2)
    F_Ext = 15 * sin(omega0 * t)
    F[0] = velocity
    F[1] = -(k * x - k * x * alphax) / m + F_Ext

    return F

def FArrVis(t: npt.NDArray, initInput: npt.NDArray) -> npt.NDArray:
    x, velocity = initInput

    F = np.zeros(2)
    F_Viscous = -b * velocity
    F[0] = velocity
    F[1] = -(k * x) / m + F_Viscous

    return F


def FArrExtVis(t: npt.NDArray, initInput: npt.NDArray) -> npt.NDArray:
    x, velocity = initInput

    F = np.zeros(2)
    F_Ext = 15 * sin(omega0 * t)
    F_Viscous = -b * velocity
    F[0] = velocity
    F[1] = -(k * x) / m + F_Ext + F_Viscous

    return F


def rk4(fArr: npt.NDArray, tn: npt.NDArray, yn: npt.NDArray, h: float) -> npt.NDArray:
    k1 = fArr(tn, yn)
    k2 = fArr(tn + 0.5 * h, yn + 0.5 * h * k1)
    k3 = fArr(tn + 0.5 * h, yn + 0.5 * h * k2)
    k4 = fArr(tn + h, yn + h * k3)

    return yn + h / 6 * (k1 + 2 * k2 + 2 * k3 + k4)


def savelog(file: str, N: int, solution: npt.NDArray):
    with open(file, "w", newline="") as writefile:
        header = ["n", "x", "velocity"]
        writer = csv.DictWriter(writefile, fieldnames=header)
        writer.writeheader()
        for i in range(N):
            writer.writerow({"n": i, "x": solution[i][0], "velocity": solution[i][1]})


def plot(
    file: str,
```

```
    t: npt.NDArray,
    solutionWOExtField: npt.NDArray,
    solutionWExtField: npt.NDArray,
    solutionWVis: npt.NDArray,
    solutionWExtFieldVis: npt.NDArray,
):
    x = {"xWOExtField": [], "xWExtField": [], "xWVis": [], "xWExtFieldVis": []}
    velocity = {"vWOExtField": [], "vWExtField": [], "vWVis": [], "vWExtFieldVis": []}

    for n in range(len(t)):
        x["xWOExtField"].append(solutionWOExtField[n][0])
        velocity["vWOExtField"].append(solutionWOExtField[n][1])

        x["xWExtField"].append(solutionWExtField[n][0])
        velocity["vWExtField"].append(solutionWExtField[n][1])

        x["xWVis"].append(solutionWVis[n][0])
        velocity["vWVis"].append(solutionWVis[n][1])

        x["xWExtFieldVis"].append(solutionWExtFieldVis[n][0])
        velocity["vWExtFieldVis"].append(solutionWExtFieldVis[n][1])

    fig, ax = plt.subplots(2, 2, figsize=(15, 7))

    ax[0][0].plot(t, x["xWOExtField"])
    ax[0][0].plot(t, velocity["vWOExtField"])
    ax[0][0].legend([r"$x$", r"v(m/s)"], fontsize=15)
    ax[0][0].set_title(r"Khi chưa có trường ngoài", loc="center")

    ax[0][1].plot(t, x["xWExtField"])
    ax[0][1].plot(t, velocity["vWExtField"])
    ax[0][1].legend([r"$x$", r"v(m/s)"], fontsize=15)
    ax[0][1].set_title(r"Khi có trường ngoài", loc="center")

    ax[1][0].plot(t, x["xWVis"])
    ax[1][0].plot(t, velocity["vWVis"])
    ax[1][0].legend([r"$x$", r"v(m/s)"], fontsize=15)
    ax[1][0].set_title(r"Khi có lực ma sát và $\alpha = 0$", loc="center")

    ax[1][1].plot(t, x["xWExtFieldVis"])
    ax[1][1].plot(t, velocity["vWExtFieldVis"])
    ax[1][1].legend([r"$x$", r"v(m/s)"], fontsize=15)
    ax[1][1].set_title(r"Khi có trường ngoài và lực ma sát", loc="center")

    plt.savefig(file)
    plt.show()



def solve(F: npt.NDArray, N: int, t0: float, t1: float, h: float, solver: npt.NDArray) -> npt.NDArray:
    initInput = np.zeros(2)
    initInput[0] = 5  # Điều kiện đầu cho tại x = 1 so với vị trí cân bằng là x = 0
    initInput[1] = 0  # Điều kiện đầu cho vật tại x = 1 là lúc thả tay ra thì v = 0

    t = np.linspace(t0, t1, N)
    solution = []

    for i in range(N):
        initInput = solver(F, t[i], initInput, h)
        solution.append(initInput)

    return t, solution


def main():
    N = 1000
    t0, t1 = 0, 50
    h = (t1 - t0) / N
    fileWrite = "nonlinearOSC.txt"
    filePlot = "nonlinearOSC.pdf"
```

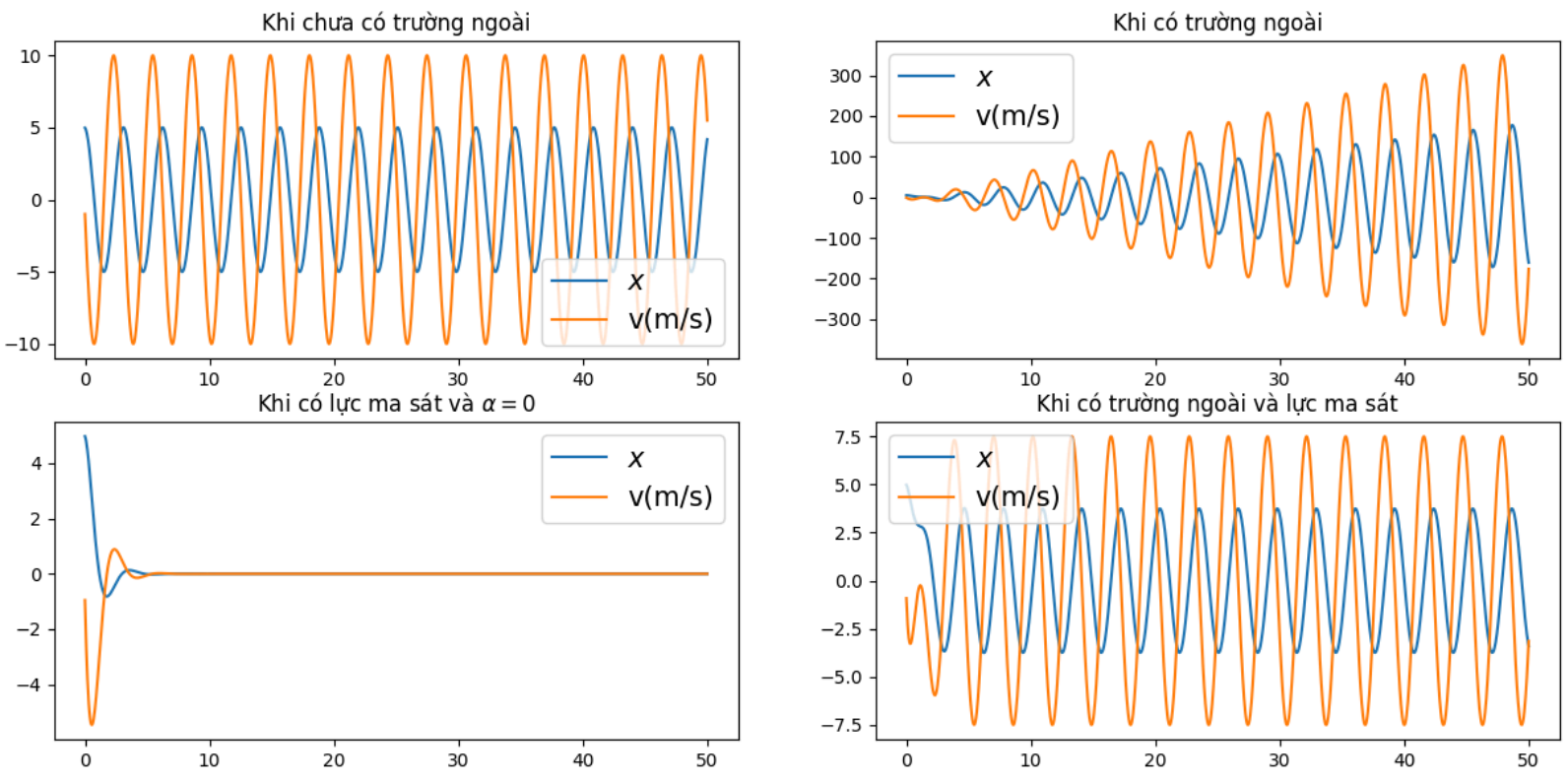
```
    t, solutionWOExtField = solve(FArr, N, t0, t1, h, rk4)
    t, solutionWExtField = solve(FArrExt, N, t0, t1, h, rk4)
    t, solutionWVis = solve(FArrVis, N, t0, t1, h, rk4)
    t, solutionWExtFieldVis = solve(FArrExtVis, N, t0, t1, h, rk4)

    plot(filePlot, t, solutionWOExtField, solutionWExtField, solutionWVis, solutionWExtFieldVis)
    savelog(fileWrite, N, solutionWOExtField)

if __name__ == "__main__":

    main()
```

Kết quả

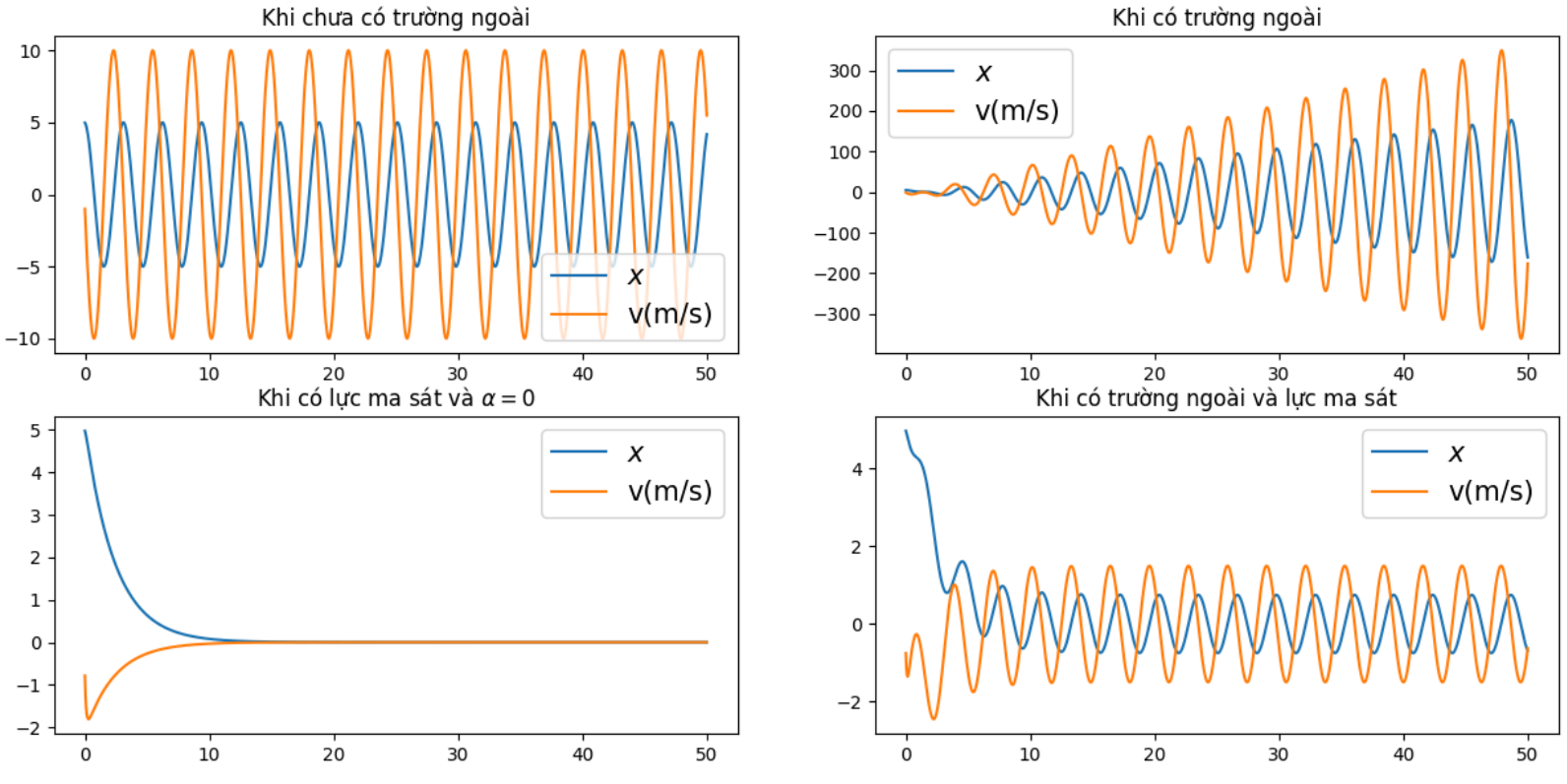
Với bộ tham số là $N = 1000, k = 2, m = 0.5, \omega_0 = \sqrt{\frac{k}{m}}, b = 0.5m\omega_0$ và $\alpha x \ll 1 \approx 0.001$.



Khi $b = 2m\omega_0$ và các thông số trên giữ nguyên.



Khi $b = 10m\omega_0$ và các thông số trên giữ nguyên.



> Có thể thấy hình vẽ trên có "dáng điệu" hợp lý ứng với mỗi b khác nhau. Khi có trường ngoài thì vận tốc biến đổi theo trường ngoài và tiếp tục tăng. Còn khi có lực ma sát, và xét trường hợp đơn giản nhất $\alpha = 0$, thì ta có thể thấy dao động bị tắt dần.

Iterative methods

Lý thuyết

Jacobian iterative method

Từ hệ phương trình tuyến tính:

$$\begin{cases} a_{11}x_1 + a12x_2 + \cdots + a_{1n}x_n = b_1 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n, \end{cases} \tag{3}$$

ta có thể xây dựng cho (3) thành dạng ma trận:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}. \tag{4}$$

Đặt:

$$\begin{aligned} A &= \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \\ &= \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix} - \begin{pmatrix} 0 & -a_{12} & \cdots & -a_{1n} \\ 0 & 0 & \cdots & -a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} - \begin{pmatrix} 0 & 0 & \cdots & 0 \\ -a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -a_{n1} & -a_{n2} & \cdots & 0 \end{pmatrix} \\ &= D - L - U, \end{aligned}$$

và

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \mathbf{x}, \qquad \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \mathbf{b}.$$

Ta viết lại (4) thành:

$$(D - L - U)\mathbf{x} = \mathbf{b} \Rightarrow D\mathbf{x} = (L + U)\mathbf{x} + \mathbf{b}.$$

Nếu tồn tại D^{-1} , ta xây dựng được ma trận x

$$\mathbf{x} = D^{-1}(L + U)\mathbf{x} + D^{-1}\mathbf{b}. \tag{5}$$

Thuật toán

Từ (5), ta có thể viết ma trận x dưới dạng tổng như sau:

$$\mathbf{x} = \sum_{i}^{n} x_i = \sum_{i}^{n} \frac{1}{a_{ii}} \left(\sum_{j \neq i}^{n} -a_{ij}x_j + b_i \right),$$

với $i = 1, 2, \ldots, n$ và n là số chiều của ma trận A.

Source code cho Jacobian & Gaussian - Seidel

```
import numpy as np
from numpy import abs
import csv
from numpy import typing as npt

AConst = [
    [10, -1, 2, 0],
    [-1, 11, -2, 3],
    [2, -1, 10, -1],
    [0, 3, -1, 8],
]

BConst = [6, 25, -11, 15]

def FArr(dim: int, x: npt.NDArray) -> npt.NDArray:  ## Đưa mảng vào bằng tay
    F = np.zeros(dim)
    F[0] = 1 / 10 * x[1] - 1 / 5 * x[2] + 6 / 10
    F[1] = 1 / 11 * x[0] + 2 / 11 * x[2] - 3 / 11 * x[3] + 25 / 11
    F[2] = -2 / 10 * x[0] + 1 / 10 * x[1] + 1 / 10 * x[3] - 11 / 10
    F[3] = -3 / 8 * x[1] + 1 / 8 * x[2] + 15 / 8

    return F

def FArrMatrixJacobian(dim: int, x: npt.NDArray) -> npt.NDArray:  ## Đưa mảng vào bằng ma trận sử dụng pp Jacobian
    F = np.zeros(dim)
    for i in range(dim):
        sumAij = 0
        for j in range(dim):
            if j != i:
                sumAij += -AConst[i][j] * x[j]

        F[i] = 1 / AConst[i][i] * (sumAij + BConst[i])

    return F

def FArrMatrixGauss(dim: int, x: npt.NDArray) -> npt.NDArray:  ## Đưa mảng vào bằng ma trận sử dụng pp Gaussian-
Seidel
    F = np.zeros(dim)
    for i in range(dim):
        sumAij_1 = 0
        sumAij_2 = 0
        for j in range(i):
            sumAij_1 += -AConst[i][j] * F[j]
        for j in range(i + 1, dim):
            sumAij_2 += -AConst[i][j] * x[j]

        F[i] = (BConst[i] + sumAij_1 + sumAij_2) / AConst[i][i]

    return F

def Jacobian(dim: int, F: npt.NDArray, N: int) -> list:
    x = np.zeros(4)
    listX = [x]  ### Lưu giá trị vào mảng để kiểm soát
```

```
    for i in range(1, N):
        x = F(dim, x)
        listX.append(x)
        if abs(max(listX[i]) - max(listX[i - 1])) / max(listX[i]) <= 1e-3:
            break

    return listX, i

def saveLog(file: str, N: int, xByhand: list, xMatrix: list, xGauss: list):
    with open(file, "w", newline="") as writefile:
        header = [f"{"k":^11}", f"{"Jacobian By Hand":^49}", f"{"Jacobian Matrix":^49}", f"{"Gaussian Matrix":^49}"]
        writer = csv.DictWriter(writefile, fieldnames=header, delimiter="|")
        writer.writeheader()

        for i in range(1, N + 1):
            writer.writerow(
                {
                    f"{"k":^11}": f"{i:^11}",
                    f"{"Jacobian By Hand":^49}": xByhand[i] if i < len(xByhand) else f"{"":<49}",
                    f"{"Jacobian Matrix":^49}": xMatrix[i] if i < len(xMatrix) else f"{"":<49}",
                    f"{"Gaussian Matrix":^49}": xGauss[i] if i < len(xGauss) else f"{"":<49}",
                }
            )

def main():
    dim = 4
    N = 40
    fileLog = "Jacobi&GaussSeidel.txt"

    xByhand, i = Jacobian(dim, FArr, N)
    xMatrix, i = Jacobian(dim, FArrMatrixJacobian, N)
    xGauss, i = Jacobian(dim, FArrMatrixGauss, N)
    saveLog(fileLog, N, xByhand, xMatrix, xGauss)

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

Cấu trúc giàn

Lý thuyết

Tìm ma trận A

$$A = \begin{pmatrix} -1 & 0 & 0 & \frac{\sqrt{2}}{2} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{3}}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{\sqrt{3}}{2} & -1 \\ 0 & -1 & 0 & \frac{\sqrt{2}}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{\sqrt{2}}{2} & 0 & -1 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & \frac{1}{2} & 0 \end{pmatrix}$$

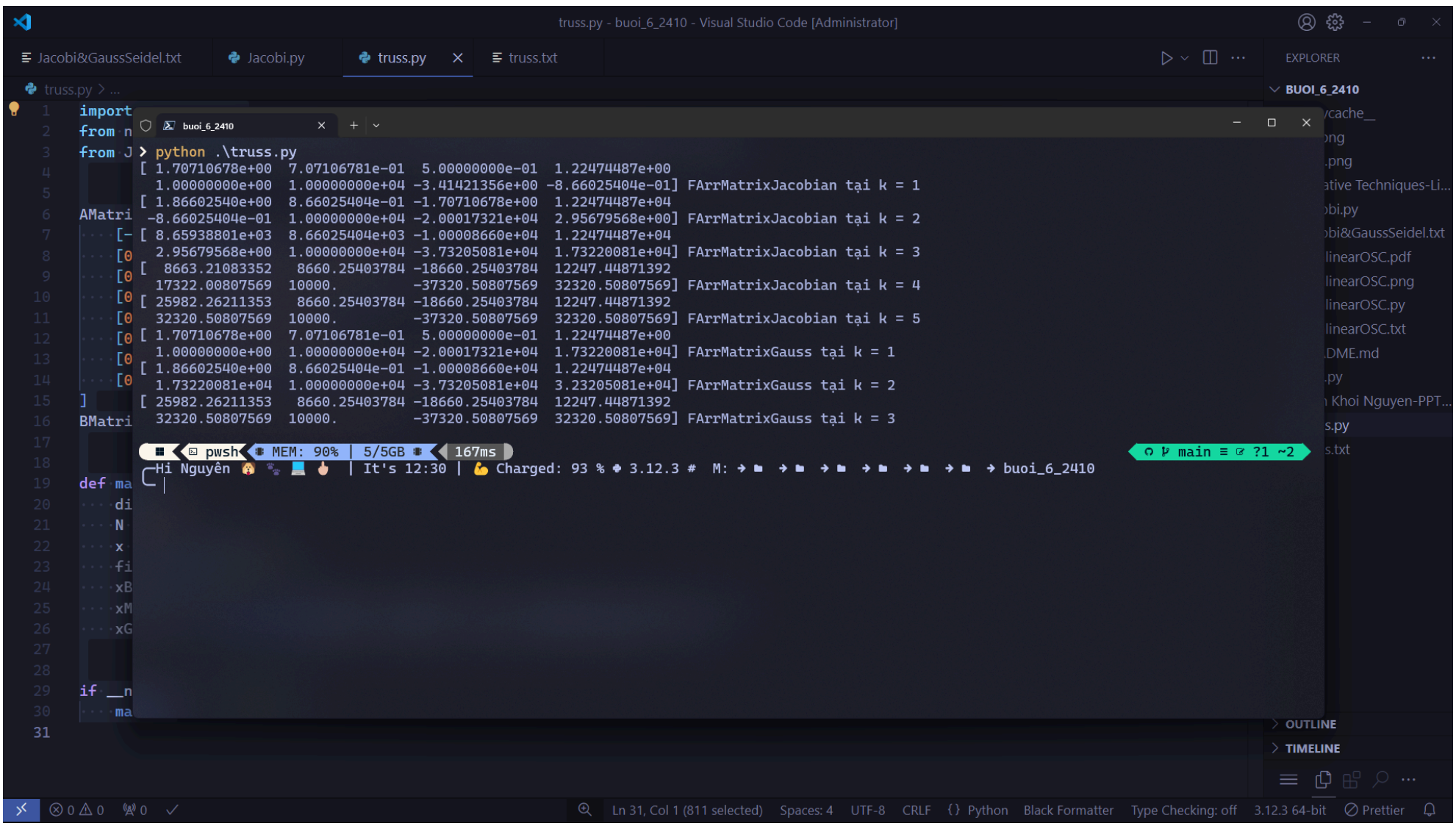
Ta có thể thấy nếu để dạng ma trận như này thì phương pháp Jacobian sẽ không giải được bởi vì không thỏa mãn được điều kiện tồn tại D^{-1} trong đó D là ma trận đường chéo. Ta phải xây dựng lại ma trận A sao cho thành phần ma trận trên đường chéo khác $không$.

$$A = \begin{pmatrix} -1 & 0 & 0 & \frac{\sqrt{2}}{2} & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & \frac{\sqrt{2}}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{3}}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{\sqrt{2}}{2} & 0 & -1 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{\sqrt{3}}{2} & -1 \end{pmatrix}$$

Source code

```python
import numpy as np
from numpy import sqrt
from Jacobi import FArrMatrixGauss, FArrMatrixJacobian, solveLoop, saveLog

AMatrix = [
    [-1, 0, 0, sqrt(2) / 2, 1, 0, 0, 0],
    [0, -1, 0, sqrt(2) / 2, 0, 0, 0, 0],
    [0, 0, -1, 0, 0, 0, 1 / 2, 0],
    [0, 0, 0, -sqrt(2) / 2, 0, sqrt(3) / 2, 0, 0],
    [0, 0, 0, 0, -1, 0, 0, 1],
    [0, 0, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, -sqrt(2) / 2, 0, -1, -1 / 2, 0],
    [0, 0, 0, 0, 0, 0, -sqrt(3) / 2, -1],
]
BMatrix = [0, 0, 0, 0, 0, 1e4, 0, 0]

def main():
    dim = 8
    N = 100
    x = np.full(dim, 1)
    fileLog = "truss.txt"
    xByhand = []
    xMatrix, i = solveLoop(AMatrix, BMatrix, dim, FArrMatrixJacobian, N, x)
    xGauss, i = solveLoop(AMatrix, BMatrix, dim, FArrMatrixGauss, N, x)

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

Kết quả



So sánh với kết quả có được từ Mathematica thì thấy được số hạng đầu bị sai, hiện tại chưa fix được mặc dù input ma trận giống nhau.

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Out[61]=

$$\left\{15\,000 \times \left(1+\sqrt{3}\right),\ 5000\sqrt{3},\ -5000 \times \left(2+\sqrt{3}\right),\ 5000\sqrt{6},\right.$$
$$\left.5000 \times \left(3+2\sqrt{3}\right),\ 10\,000,\ -10\,000 \times \left(2+\sqrt{3}\right),\ 5000 \times \left(3+2\sqrt{3}\right)\right\}$$

Out[62]//MatrixForm=

$$\begin{pmatrix} 15\,000 \times \left(1+\sqrt{3}\right) \\ 5000\sqrt{3} \\ -5000 \times \left(2+\sqrt{3}\right) \\ 5000\sqrt{6} \\ 5000 \times \left(3+2\sqrt{3}\right) \\ 10\,000 \\ -10\,000 \times \left(2+\sqrt{3}\right) \\ 5000 \times \left(3+2\sqrt{3}\right) \end{pmatrix}$$