Project report on

Solving Material science and metallurgical problem using Python

Course : MM 644 – ICME

Submitted by,

Namrata Sarania (2102105016)

Group -1

Under the Guidance of

DR. SUMANTA SAMAL

DEPARTMENT OF METALLURGY ENGINEERING AND MATERIALS SCIENCE

INDIAN INSTITUTE OF TECHNOLOGY INDORE

APRIL 2022

CONTENTS

1.Introduction :

Navier-Stokes equations : These equations describe how the velocity, pressure, temperature, and density of a moving fluid are related. The Navier-Stokes equations consists of a time-dependent continuity equation for conservation of mass, three time-dependent conservation of momentum equations and a time-dependent conservation of energy equation.

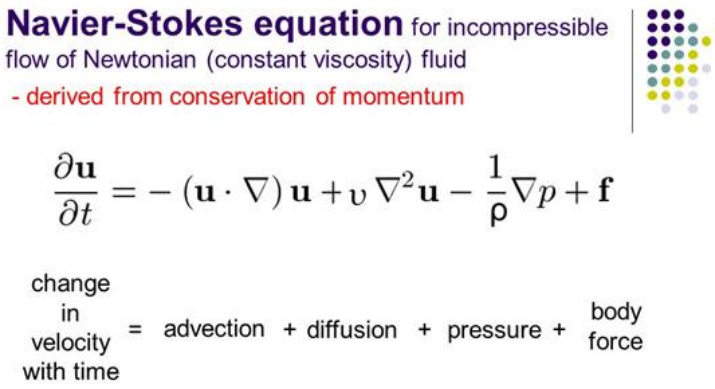

Image source : https://slideplayer.com/slide/9727750/

In this present work , we are basically working on the 2D diffusion which is a small part of Navier – stokes equations . Diffusion is a physical process that occurs in a flow of gas in which some property is transported by the random motion of the molecules of the gas.

2D diffusion is described as follows :

$$\frac{\partial u}{\partial t} = \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$

$$\frac{\partial v}{\partial t} = \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)$$

Image source : https://slideplayer.com/slide/9727750/

Verlet Integration : Verlet integration is a numerical method used to integrate Newton's equations of motion. It is frequently used to calculate trajectories of particles in molecular dynamics simulations and video games.

$$\vec{x}(t+\Delta t) = \vec{x}(t) + \vec{v}(t)\Delta t + \frac{\vec{a}(t)\Delta t^2}{2} + \frac{\vec{b}(t)\Delta t^3}{6} + \mathcal{O}(\Delta t^4)$$

$$\vec{x}(t-\Delta t) = \vec{x}(t) - \vec{v}(t)\Delta t + \frac{\vec{a}(t)\Delta t^2}{2} - \frac{\vec{b}(t)\Delta t^3}{6} + \mathcal{O}(\Delta t^4),$$

where $\vec{x}$ is the position, $\vec{v} = \dot{\vec{x}}$ the velocity, $\vec{a} = \ddot{\vec{x}}$ the acceleration and $\vec{b}$ the jerk (third derivative of the position with respect to the time) t.

https://resources.saylor.org/wwwresources/archived/site/wp-content/uploads/2011/06/MA221-6.1.pdf

Adding these two expansions gives,

$$\vec{x}(t+\Delta t) = 2\vec{x}(t) - \vec{x}(t-\Delta t) + \vec{a}(t)\Delta t^2 + \mathcal{O}(\Delta t^4).$$

We can see that the first and third-order terms from the Taylor expansion cancel out, thus making the Verlet integrator an order more accurate than integration by simple Taylor expansion alone.

2.Objectives

The main objective of this present work is to study different material science and metallurgical problem and solve them using Python. In this work I have taken two problems – one is related to Navier stokes equations and another one is plotting the trajectory of a Simple harmonic oscillator using Verlet integration .

What I did ?

1. In the first problem , I solved the 2D second order diffusion (NS equations) at different time steps using Python and plotted the final velocity profile.
2. In the second problem, compared the Analytical solution and Verlet trajectory of a particle in motion of a simple harmonic oscillator (done the error analysis after the simulation).

3.Python Library :

Listed some of the python libraries with their functions which were used in this project work .

Python Library	Functions
numpy	numpy is a Python library used for working with arrays.
matplotlib	matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
mpl_toolkits.mplot3d	Generating 3D plots using the mplot3d toolkit.
set_xlim() , set_ylim()	function in axes module of matplotlib library is used to set the x-axis , y-axis view limits.
pyplot	pyplot is a collection of functions that make matplotlib work like MATLAB.

4.Problem Statement :

4.1. Problem -1 :Solving 2D second order diffusion (NS equations) at different time steps using Python

This analysis shows the final velocity profile for 2D diffusion when the initial conditions are a square wave and the boundary conditions are unity.

Input Data (variables) ,

nx = 31 (number of x spatial points)

ny = 31 (number of y spatial points)

nt = 17 (number of temporal points)

nu = .05

xmax =2 , ymax =2

Grid spacing,

dx = 2 / (nx - 1)

dy = 2 / (ny - 1)

sigma = .25

dt = sigma * dx * dy / nu

Initial conditions

x	i	y	j	u(x,y,t), v(x,y,t)
0	0	0	0	1
$0 < x \leq 0.5$	$0 < i \leq 5$	$0 < y \leq 0.5$	$0 < j \leq 5$	1
$0.5 < x \leq 1$	$5 < i \leq 10$	$0.5 < y \leq 1$	$5 < j \leq 10$	2
$1 < x < 2$	$10 < i < 20$	$1 < y < 2$	$10 < j < 20$	1
2	20	2	20	1

Boundary conditions

x=0 and x=2, y=0 and y=2

u[0,:,:] = u[nx-1,:,:] = u[:,0,:] = u[:,ny-1,:] = 1

v[0,:,:] = v[nx-1,:,:] = v[:,0,:] = v[:,ny-1,:] = 1

Design algorithm to solve the problem

(a) Space – time discretization

- i → index of grid in x
- j → index of grid in y
- n → index of grid in t

(b) Numerical scheme

- **FD** in time
- **CD** in space

(c) Discrete equation

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = \nu \frac{u_{i-1,j}^{n} - 2u_{i,j}^{n} + u_{i+1,j}^{n}}{\Delta x^2} + \nu \frac{u_{i,j-1}^{n} - 2u_{i,j}^{n} + u_{i,j+1}^{n}}{\Delta y^2}$$

$$\frac{v_{i,j}^{n+1} - v_{i,j}^{n}}{\Delta t} = \nu \frac{v_{i-1,j}^{n} - 2v_{i,j}^{n} + v_{i+1,j}^{n}}{\Delta x^2} + \nu \frac{v_{i,j-1}^{n} - 2v_{i,j}^{n} + v_{i,j+1}^{n}}{\Delta y^2}$$

4.1.1. Python Code

```
import numpy
from matplotlib import pyplot, cm
from mpl_toolkits.mplot3d import Axes3D ##library for 3d projection plots

###variable declarations
nx = 31
ny = 31
nt = 17
nu = .05
dx = 2 / (nx - 1)
dy = 2 / (ny - 1)
sigma = .25
dt = sigma * dx * dy / nu

x = numpy.linspace(0, 2, nx)
y = numpy.linspace(0, 2, ny)

u = numpy.ones((ny, nx))  # create a 1xn vector of 1's
un = numpy.ones((ny, nx))

###Assign initial conditions
# set hat function I.C. : u(.5<=x<=1 && .5<=y<=1 ) is 2
u[int(.5 / dy):int(1 / dy + 1),int(.5 / dx):int(1 / dx + 1)] = 2

fig = pyplot.figure(figsize=(11, 7), dpi=100)
ax = fig.gca(projection='3d')
X, Y = numpy.meshgrid(x, y)
surf = ax.plot_surface(X, Y, u, rstride=1, cstride=1, cmap=cm.viridis,
        linewidth=0, antialiased=False)
```

```
ax.set_xlim(0, 2)
ax.set_ylim(0, 2)
ax.set_zlim(1, 2.5)

ax.set_xlabel('$x$')
ax.set_ylabel('$y$');



###Run through nt timesteps
def diffuse(nt):
    u[int(.5 / dy):int(1 / dy + 1),int(.5 / dx):int(1 / dx + 1)] = 2

    for n in range(nt + 1):
        un = u.copy()
        u[1:-1, 1:-1] = (un[1:-1,1:-1] +
                         nu * dt / dx**2 *
                         (un[1:-1, 2:] - 2 * un[1:-1, 1:-1] + un[1:-1, 0:-2]) +
                         nu * dt / dy**2 *
                         (un[2:,1: -1] - 2 * un[1:-1, 1:-1] + un[0:-2, 1:-1]))
        u[0, :] = 1
        u[-1, :] = 1
        u[:, 0] = 1
        u[:, -1] = 1

    fig = pyplot.figure(figsize=(11, 7), dpi=100)
    ax = fig.gca(projection='3d')
    surf = ax.plot_surface(X, Y, u[:], rstride=1, cstride=1, cmap=cm.viridis,
        linewidth=0, antialiased=True)
    ax.set_zlim(1, 2.5)
    ax.set_xlabel('$x$')
    ax.set_ylabel('$y$');

```

4.1.2 Result and discussion :

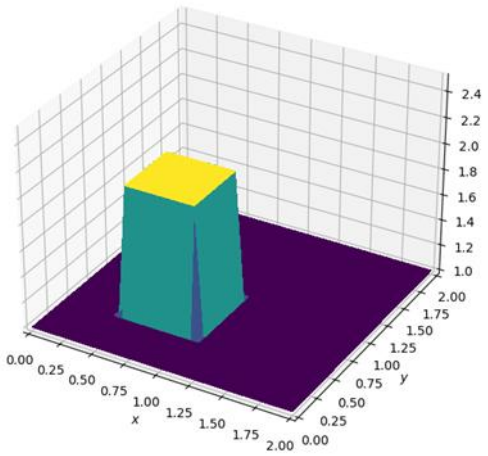

Fig 1. - Initial runfile

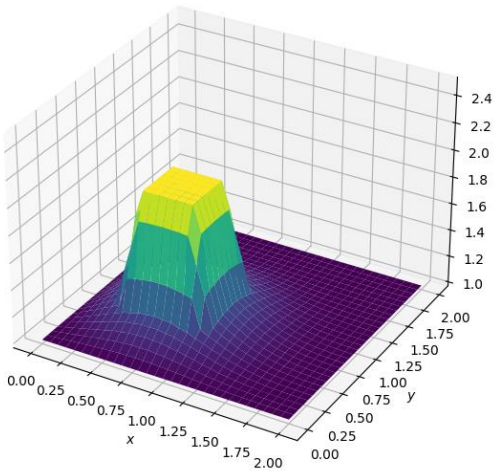

Fig 2. -diffusion at zero time steps

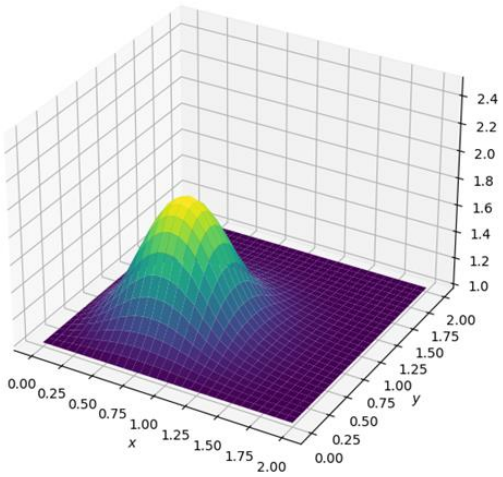

Fig 3. -diffusion at time steps =10

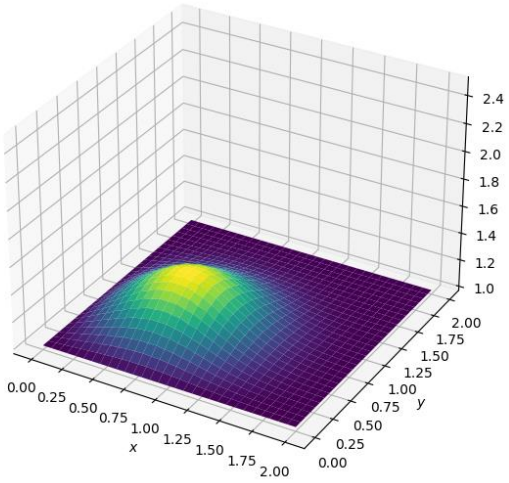

Fig 4. -diffusion at time steps =50

From the above figures we can conclude that we get flatter diffusion curve as we increase the number of time steps (Fig 3, Fig 4) . After the initial run file when we enter time steps and run the code again we get a smoother curve (Fig 2.) compared to the previous one (Fig 1.)

4.2 .Problem -2 :Verlet integration method to plot trajectories of a simple harmonic oscillator

Trajectory of a simple harmonic oscillator example is used in this present work. Comparing the analytical solution and verlet integration solution trajectories , and then performing some error analysis by changing the accuracy .

Constants used ,

Spring constant , k =1

Mass of the particle , m =1

Time step or accuracy , dt = 0.1

Simulation time , tm = 10

Initial conditions

x =[1]

v = [0]

time ,t =0

x and v are the initial position and velocity, respectively.

Verlet Algorithm

- Acceleration

  a = -(k/m) *x[i]

- Using Euler cromer method for initialization ,

  Velocity , v_next =v[i]+a*dt

  Position ,x_next =x[i] +v_next*dt

  Next step , using Verlet

  Position, x_next =2*x[i] –x[i-1] + a*dt**2

- Position update , x.append (x_next)
- Time update , t.append(t[i]+dt)
- Analytical solution ,

  omg=(k/m)**0.5

  xa= x[0]*cos(omg*array(t))

4.2.1. Python Code :

```
from numpy import cos, array
#%% Constants

k  = 1     # Spring constant
m  = 1     # Mass
dt = 0.1   # Accuracy
tm = 10    # Simulation time

#%% Initial conditions

x = [1]    # initial position
v = [0]    # initial velocity

# Time
t = [0]

```

```
#%% Verlet algorithm

i = 0

while t[-1] <= tm:

    # Acceleration
    a = -(k/m)* x[i]

    if i == 0:

        # Euler Cromer method

        # Velocity
        v_next = v[i] + a* dt

        # Position
        x_next = x[i] + v_next* dt

    else:

        # Position
        x_next = 2* x[i] - x[i-1] + a* dt**2

    # Position update
    x.append(x_next)

    # Time update
    t.append(t[i] + dt)

    # Loop update
    i = i + 1
```

```
#%% Analytical solution

omg = (k/ m)**0.5
xa  = x[0]* cos(omg* array(t))

#%% Plot
import matplotlib.pyplot as plt

plt.plot(t, xa, label = 'Analytical')
plt.plot(t, x, 'ro', label = 'Verlet')
plt.title('Verlet', fontweight = 'bold', fontsize = 16)
plt.xlabel('t', fontweight = 'bold', fontsize = 14)
plt.ylabel('X', fontweight = 'bold', fontsize = 14)
plt.grid(True)
plt.legend()
plt.show()
```

4.2.2. Result and discussion

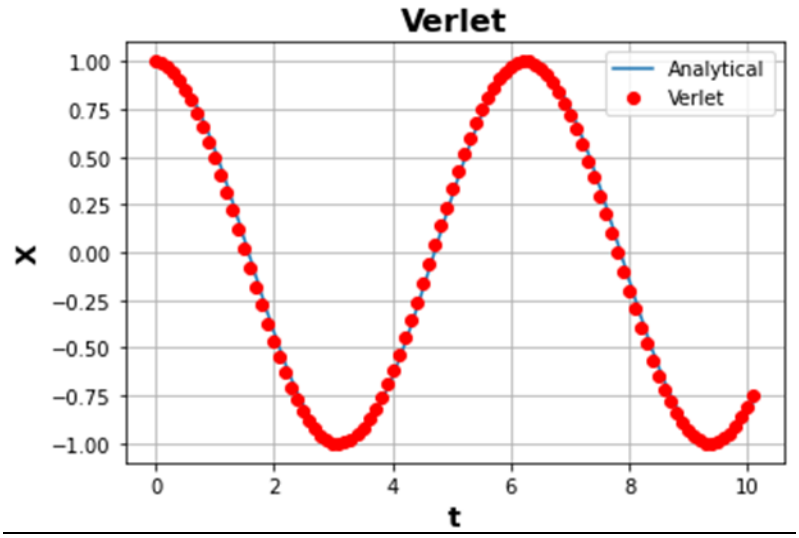

Fig 5. -Trajectory at dt =0.1

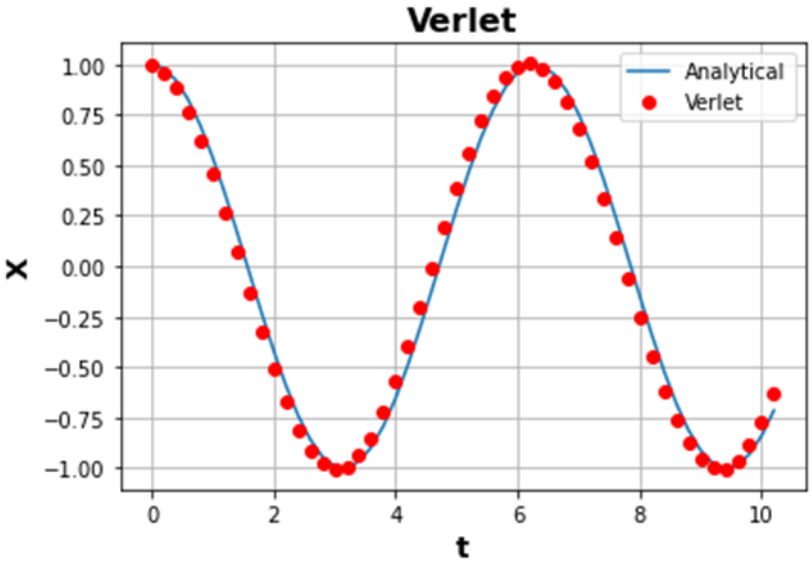

Fig 6. -Trajectory at dt =0.2

From the above figures we can conclude that , when the dt is small the trajectories of Analytical and Verlet are approximately same i.e. very less amount is there (Fig 5) . On the other hand , when we increase the accuracy dt more and more then we have observed the Verlet trajectory deviates from the analytical one i.e. error increases when we increase the dt (Fig 6).

5.References :

- Dr. Sumanta Samal class notes (MM 644 / Lec -15)
- Dr. Sumanta Samal class notes (MM 644 / Lec -10)
- http://www.thevisualroom.com/2D_linear_diffusion.html
- https://slideplayer.com/slide/5310307/
- https://resources.saylor.org/wwwresources/archived/site/wp-content/uploads/2011/06/MA221-6.1.pdf