Simulating Nanofluidics: an LLM Study

Aman Chawla

September 16, 2024

Abstract

In this note the authors present initial results that show the magnetic field and its power spectrum generated by a box of N particles with random velocities and a drift. The simulation is intended to mimic a chunk of fluid in motion, with the goal of determining its velocity using an NV center (Nitrogen Vacancy) which can sense the magnetic field.

In the paper [1], the authors present molecular dynamics simulations to mimic fluid flow and then sense the flow using an NV center via the magnetic field generated. Their analysis shows that the power spectrum of the measured magnetic field is related to the drift velocity of the fluid. In this note, we use an LLM to present us with a Python script (Algorithm 1-2) and simulation results (Figures 1 and 2) for this setting.

In future work, we will provide a specific bounding box shape and study flow in it using similar methodology. Our goal is ultimately to develop a nanopump that can generate nanoflows which can then be sensed using the mechanism simulated in this note.

References

[1] Daniel Cohen, Ramil Nigmatullin, Oded Kenneth, Fedor Jelezko, Maxim Khodas, and Alex Retzker. Utilising nv based quantum sensing for velocimetry at the nanoscale. *Scientific Reports*, 10(1):5298, 2020.

Algorithm 1 Python molecular dynamics.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq
# Constants and parameters
N = 30  # number of particles
L = 10.0  # size of the simulation box (cubic box with length L)
d = 2.0  # distance of NV center below the box (along z-axis)
mass = 1.0  # mass of particles (assuming uniform)
dt = 0.01  # time step for the simulation
n_steps = 100  # number of simulation steps
epsilon = 1.0  # Lennard-Jones potential parameter (depth of potential well)
sigma = 1.0  # Lennard-Jones potential parameter (finite distance at which inter
# Initialize particle positions and velocities
positions = np.random.rand(N, 3) * L
# positions in a 3D box (x, y, z)
velocities = np.random.randn(N, 3) * 0.1
# random initial velocities
# NV center position (below the box, center of x-y projection)
nv_center = np.array([L / 2, L / 2, -d])
# Define the Lennard-Jones potential
def lennard_jones(r, epsilon, sigma):
        return 4 * epsilon * ((sigma / r)**12 - (sigma / r)**6)
# Compute forces using the Lennard-Jones potential
def compute_forces(positions, epsilon, sigma, L):
        forces = np.zeros_like(positions)
        for i in range(N):
                for j in range(i + 1, N):
                        rij = positions[i] - positions[j]
# Apply minimum image convention for periodic boundary conditions
                        rij = rij - np.round(rij / L) * L
                        r = np.linalg.norm(rij)
                        if r > 0:  # To avoid division by zero
                                f = 24 * epsilon * (2 * (sigma / r)**12 - (sigma
                                forces[i] += f
                                forces[j] -= f
        return forces
```

Algorithm 2 Part 2 of the simulation algorithm.

```
# Compute magnetic field at NV center due to moving particles (Biot-Savart Law)
def compute_magnetic_field(positions, velocities, nv_center):
        mu_0 = 1.0  # permeability of free space (simplified for units)
        magnetic_field = np.zeros(3)
        for i in range(N):
                r = nv_center - positions[i]
                r_mag = np.linalg.norm(r)
                if r_mag > 0:  # avoid division by zero
                        cross_product = np.cross(velocities[i], r)
                        magnetic_field += mu_0 * cross_product / (r_mag**3)
return magnetic_field
# Main simulation loop
magnetic_fields = []
for step in range(n_steps):
        # Update forces
        forces = compute_forces(positions, epsilon, sigma, L)
# Update velocities and positions using Velocity Verlet algorithm
        velocities += forces * dt / (2 * mass)
        positions += velocities * dt
        velocities += forces * dt / (2 * mass)
        # Apply periodic boundary conditions
        positions = np.mod(positions, L)
        # Compute magnetic field at NV center
        magnetic_field = compute_magnetic_field(positions, velocities, nv_center
        magnetic_fields.append(magnetic_field)
# Convert magnetic field list to numpy array for further analysis
magnetic_fields = np.array(magnetic_fields)
# Compute the magnitude of the magnetic field over time
magnetic_magnitude = np.linalg.norm(magnetic_fields, axis=1)
# Plot the magnetic field magnitude over time plt.figure()
plt.plot(magnetic_magnitude) plt.xlabel('Time step') plt.ylabel('Magnetic field
# Compute and plot the power spectrum (Fourier Transform of the magnetic field m
magnetic_spectrum = fft(magnetic_magnitude)
freqs = fftfreq(n_steps, dt)
plt.figure()
plt.plot(freqs[:n_steps // 2], np.abs(magnetic_spectrum)[:n_steps // 2])
plt.xlabel('Frequency')
plt.ylabel('Power Spectrum (Magnetic Field)')
plt.title('Power Spectrum of Magnetic Field at NV Center') plt.show()
```

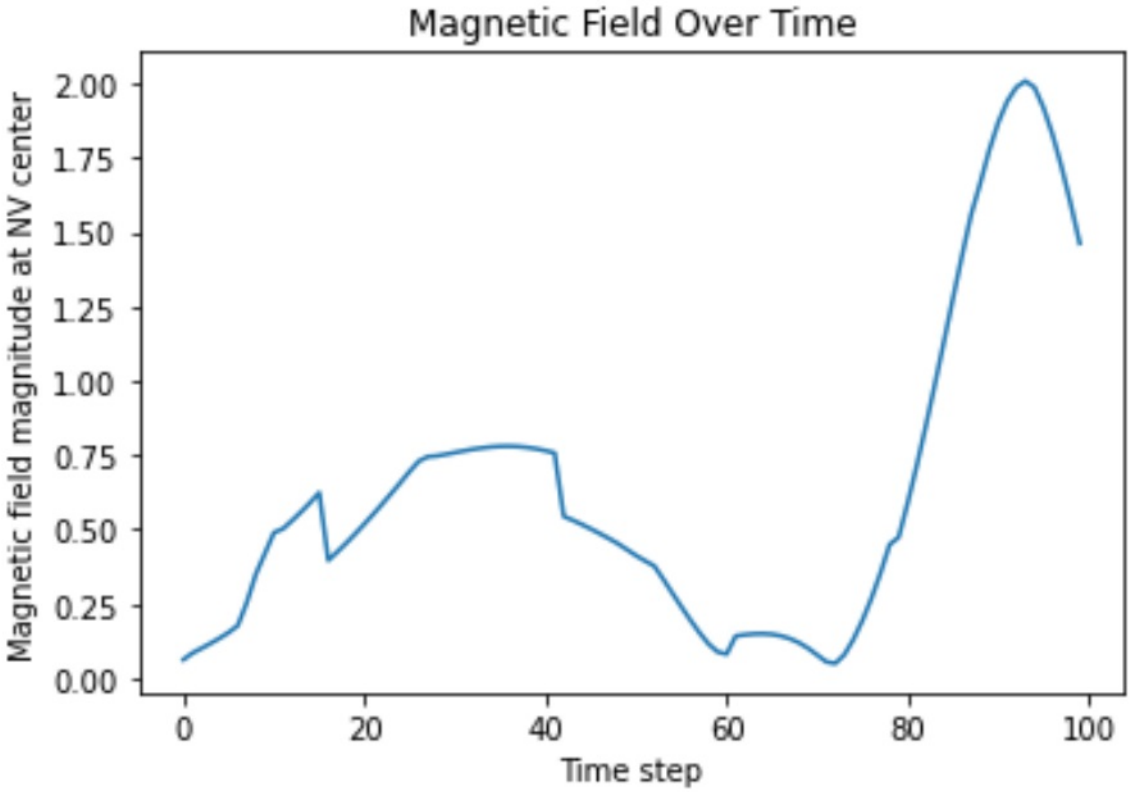

Figure 1: Variation of the sensed magnetic field with time. The vertical axis is magnetic field magnitude in unitless form.

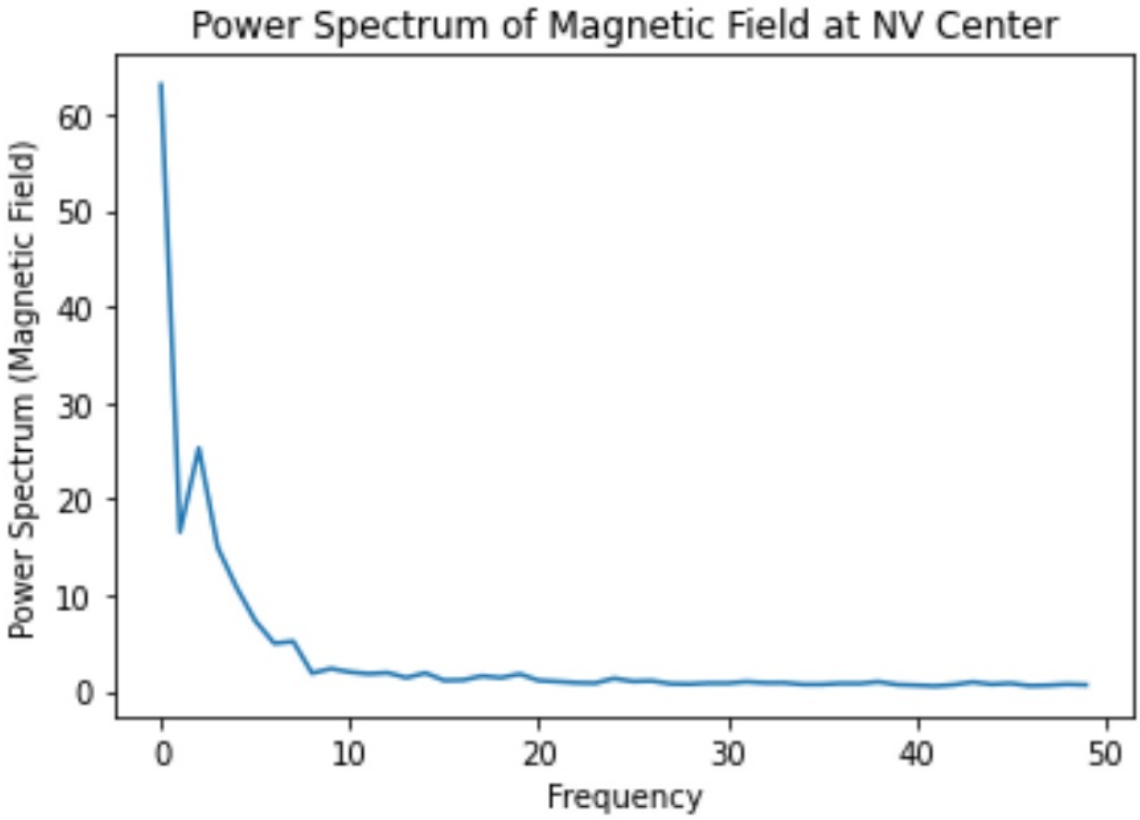

Figure 2: Power spectrum of the magnetic field at the NV center. The horizontal axis is frequency in unitless form.