**Simulation of DESI Solvent Flow and Analyte Transport on Fingerprint Ridges**

This document details a Python script simulating the dynamic interaction of a solvent spray with a patterned surface, mimicking Desorption Electrospray Ionization (DESI) Mass Spectrometry applied to fingerprint analysis. The simulation employs the Lattice Boltzmann Method (LBM) for fluid dynamics, coupled with a simplified model for analyte dissolution and diffusion, and includes a graphical user interface (GUI) for interactive control and visualization.

**1. Introduction**

Desorption Electrospray Ionization (DESI) is an ambient ionization technique used in mass spectrometry for analyzing latent fingerprints. It involves directing a charged solvent spray onto a surface, dissolving deposited residues (analytes), and carrying the solution into the mass spectrometer. The efficiency depends on the solvent's interaction with fingerprint ridges and valleys. Simulating solvent flow and analyte pick-up can optimize DESI parameters. This work presents a 2D LBM simulation coupled with a passive scalar model.

**2. Numerical Method: Lattice Boltzmann Method (LBM)**

The fluid flow is simulated using the LBM, a mesoscopic technique modeling fluid flow by tracking the movement and collision of probability distribution functions ($ f\_i $) of hypothetical particles on a discrete lattice.

**2.1. D2Q9 Model**

The simulation uses the two-dimensional nine-velocity (D2Q9) model. The lattice is a square grid, with particles moving in 9 directions. The discrete velocity vectors $ c\_i $ and their corresponding weights $ w\_i $ are defined as:

**Velocity Vectors:**

In the code, the velocity vectors $ c $ are explicitly given by:

c = np.array([  
 [0, 0], [1, 0], [0, 1], [-1, 0], [0, -1],  
 [1, 1], [-1, 1], [-1, -1], [1, -1]  
])

**Weights:**

In the code, the weights $ w $ are:

w = np.array([4/9] + [1/9]\*4 + [1/36]\*4)

The speed of sound squared in lattice units is $ c\_s^2 = \sum\_i w\_i c\_{i\alpha} c\_{i\beta} / (D \cdot \delta\_{\alpha\beta}) = 1/3 $, which is defined as c\_sqr in the code.

**2.2. Lattice Boltzmann Equation (BGK Model)**

The evolution of the particle distribution functions $ f\_i(x, t) $ is governed by the Lattice Boltzmann equation. Using the Bhatnagar-Gross-Krook (BGK) approximation for the collision term, the equation is:

where $ \Delta t $ is the time step (set to 1 in lattice units), $ \tau $ is the relaxation time, and $ f\_i^{eq} $ is the equilibrium distribution function. The collision frequency is $ \omega = 1/\tau $. The code uses $ \omega $ in the update step:

followed by the streaming step:

In the code, streaming is performed using np.roll after the collision update:

# Collision step  
f += omega \* (feq - f)  
# Streaming step  
for i, ci in enumerate(c):  
 f[i] = np.roll(f[i], ci, axis=(0, 1))

**2.3. Macroscopic Variables**

Macroscopic density () and velocity () are calculated from the moments of the distribution functions:

In the code, these are calculated as:

rho = np.sum(f, axis=0)  
u[^0] = np.sum(f \* c[:, 0, None, None], axis=0) / rho # u\_x  
u[^1] = np.sum(f \* c[:, 1, None, None], axis=0) / rho # u\_y

A small epsilon is used for $ \rho $ to prevent division by zero.

**2.4. Equilibrium Distribution Function**

The equilibrium distribution function for the D2Q9 model, based on a truncated Maxwell-Boltzmann expansion, is:

With $ c\_s^2 = 1/3 $ in lattice units, this simplifies to:

The code implements this as the equilibrium function:

def equilibrium(rho, u):  
 cu = np.einsum('ia,axy->ixy', c, u) # c\_i . u  
 usqr = u[^0]\*\*2 + u[^1]\*\*2 # u . u  
 feq = np.einsum('i,jk->ijk', w, rho) \* (1 + 3\*cu + 9/2\*cu\*\*2 - 3/2\*usqr[None, :, :])  
 return feq

**2.5. Kinematic Viscosity and Relaxation Time ()**

The kinematic viscosity () of the simulated fluid is directly related to the relaxation time $ \tau $ and lattice parameters:

In lattice units with $ c\_s^2 = 1/3 $ and $ \Delta t = 1 $:

The tau parameter, controlled by a GUI slider (tau\_scale), directly governs the fluid viscosity in the simulation. A stable simulation requires $ \tau > 0.5 $. The GUI limits $ \tau $ to the range $$0.55, 2.0].

To relate this lattice viscosity to real-world fluid viscosities, a scaling factor () is needed:

If we assume a specific scaling (e.g., choosing $ \Delta x\_{real} $ and $ \Delta t\_{real} $ such that $ \tau = 1.0 $ corresponds to the known viscosity of the 95:5 Methanol:Water mixture, which is approximately $ 0.736 \times 10^{-6} \ m^2/s $ at 25°C), we can find the approximate $ \tau $ values that correspond to the kinematic viscosities of other common solvents.

Table 1 shows the approximate kinematic viscosity for several solvents at 25°C and the corresponding $ \tau $ values in the simulation based on this assumed scaling.

**Table 1: Approximate Kinematic Viscosity and Corresponding Simulation $ \tau $ Values** (at 25°C, based on LBM scaling where $ \tau = 1.0 $ represents 95:5 MeOH:Water)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Solvent | Approx. $ \nu\_{real} \ (10^{-6} \ m^2/s) $ | Approx. $ \nu\_{lattice} $ | Approx. $ \tau $ | Notes |
| Hexane | 0.45 | 0.102 | 0.81 | Common non-polar solvent |
| Acetone | 0.45 | 0.102 | 0.81 | Common polar aprotic solvent |
| Acetonitrile | 0.46 | 0.105 | 0.82 | Common polar aprotic solvent |
| Methanol (Pure) | 0.73 | 0.166 | 1.00 |  |
| Methanol:Water (95:5) | 0.736 | 0.167 | 1.00 | The reference value for this scaling |
| Water | 0.89 | 0.202 | 1.11 |  |
| Ethanol | 1.46 | 0.332 | 1.50 |  |
| Isopropanol (IPA) | 2.41 | 0.548 | 2.14 | Slightly above slider max (2.0) |

By adjusting the tau slider in the GUI to these approximate values, the simulation can qualitatively represent the flow behavior of these different solvents relative to each other.

**3. Computational Domain and Geometry**

The simulation domain is a 2D Cartesian grid of size $ n\_x \times n\_y $ ($ n\_x = 300 $, $ n\_y = 100 $). Solid objects (fingerprint ridges) are represented by a binary mask (ridges\_rotated).

**3.1. Ridge Pattern Generation**

Several parametric functions (generate\_parallel\_ridges, generate\_wavy\_ridges, generate\_whorl\_ridges, generate\_arch\_ridges, generate\_loop\_ridges) are available to generate different initial ridge patterns, approximating typical fingerprint features. These functions create a binary NumPy array (1 for ridge, 0 for fluid) based on geometric formulas.

**3.2. Roughness**

The apply\_roughness function adds a simple form of geometric roughness to the generated ridge patterns by randomly flipping the state of boundary pixels based on a roughness\_factor. This provides a visual roughness to the solid boundaries but does not rigorously model surface texture effects on flow or dissolution at a sub-grid scale.

**3.3. Rotation**

The initial ridge pattern is rotated by a user-specified angle (rotation\_angle\_degrees) using the Pillow (PIL) library's image rotation function before being used to define the solid mask (ridges\_rotated) in the LBM domain.

**4. Boundary Conditions**

Appropriate boundary conditions are necessary to define the interaction of the fluid with the domain boundaries and solid objects.

**4.1. Solid Boundaries (Ridges)**

Solid grid nodes, defined by the ridges\_rotated mask, are treated as no-slip boundaries. The simulation applies a simplified approach where the macroscopic velocity $ u $ at these nodes is explicitly set to zero after the LBM streaming and macroscopic variable calculation steps:

u[:, ridges\_rotated] = 0

A more standard and theoretically consistent LBM method for no-slip walls is the full bounce-back scheme, which reflects incoming particle distribution functions.

**4.2. Inflow Boundary (Left, x=0)**

The left boundary ($ x = 0 $) simulates the DESI spray source. A macroscopic velocity profile is imposed directly at this boundary:

u[0, 0, y] = ux\_spray # Fixed x-component  
u[1, 0, y] = uy\_spray\_base \* profile[y] # y-component based on profile

The spray velocity components (ux\_spray, uy\_spray\_base) and the shape of the profile (Gaussian or Uniform) are determined by the user-controlled parameters: spray angle, tip-to-surface distance, and spray width factor. This method of directly setting macroscopic velocity at the boundary is a simplification compared to standard LBM inflow conditions (e.g., Zou-He), which operate on the distribution functions.

**4.3. Outflow Boundary (Right, x=nx-1)**

The right boundary ($ x = n\_x - 1 $) allows fluid to exit the domain. A simple extrapolation boundary condition is applied to the distribution functions entering the boundary from the left:

f[3, nx - 1, :] = f[3, nx - 2, :] # Distribution coming from the west ([-1, 0])  
f[6, nx - 1, :] = f[6, nx - 2, :] # Distribution coming from the northwest ([-1, 1])  
f[7, nx - 1, :] = f[7, nx - 2, :] # Distribution coming from the southwest ([-1, -1])

This copies the values of the incoming distribution functions from the adjacent inner layer of grid nodes. While simple, this method can sometimes lead to reflections or numerical instability depending on the flow conditions.

**5. Analyte Dissolution and Transport Model**

A simplified model is included to simulate how analytes initially located on the solid ridges dissolve into the solvent and spread. This is treated as a passive scalar transport problem, currently including dissolution and diffusion, but not advection (transport by the fluid velocity field).

**5.1. Analyte Fields**

Two 2D NumPy arrays track the analyte:

* analyte\_on\_surface ($ n\_x, n\_y $): Stores the amount of analyte remaining on each solid ridge pixel. Initialized with initial\_surface\_analyte\_amount on ridges\_rotated pixels and 0 elsewhere.
* analyte\_concentration ($ n\_x, n\_y $): Stores the concentration of dissolved analyte in the fluid phase. Initialized to 0 everywhere. This field is effectively treated as 0 inside solid pixels during calculations and boundary conditions.

**5.2. Dissolution**

Dissolution is modeled as a source term for analyte\_concentration in fluid cells adjacent to solid ridge cells that still have analyte on their surface (analyte\_on\_surface > 0). In each time step, for every active ridge cell ($ r\_x, r\_y $):

1. The amount dissolved is limited by the dissolution\_rate\_constant or the remaining analyte\_on\_surface.
2. This amount is transferred from analyte\_on\_surface[rx, ry] to the analyte\_concentration in all adjacent fluid cells, divided equally among them. This process represents mass transfer from the solid surface into the bulk fluid driven by a simple rate law, neglecting potential effects like solubility limits or mass transfer limitations in the fluid boundary layer.

**5.3. Diffusion**

Diffusion of the dissolved analyte in the fluid is modeled using a simple finite difference approximation of Fick's Second Law (), assuming lattice units $ \Delta t = \Delta x = \Delta y = 1 x, y $), the concentration update is approximately:

where $ D $ is the diffusion coefficient (diffusion\_coefficient\_C). The Laplacian () is calculated using a 5-point stencil. A reflection method is used to approximate the no-flux boundary condition () at solid (ridge) walls; if a neighbor is solid, its concentration is treated as equal to the fluid cell's concentration in the stencil calculation.

This diffusion step is performed in the update\_analyte function after the dissolution step.

**5.4. Analyte Boundary Conditions**

Boundary conditions are applied to the analyte\_concentration field after the dissolution and diffusion steps:

* Inflow ($ x = 0 $): analyte\_concentration[0, :] = 0 (zero concentration in incoming spray).
* Outflow ($ x = n\_x - 1 $): analyte\_concentration[nx-1, :] = analyte\_concentration[nx-2, :] (simple extrapolation, zero-gradient approximation).
* Solid Boundaries (Ridges): analyte\_concentration[ridges\_rotated] = 0 (concentration is zero inside solid regions).

**6. Implementation Details**

The script is written in Python using numpy for numerical calculations, matplotlib for plotting, tkinter for the GUI, and PIL for image rotation. The update\_flow function orchestrates the simulation loop, performing LBM steps and calling update\_analyte for each time step advanced by the GUI slider.

**7. Graphical User Interface (GUI)**

The Tkinter GUI provides interactive control over simulation parameters:

* **Simulation Parameters Frame**: Controls for Ridge Type, Rotation Angle, and Fluid Relaxation Time (/Viscosity).
* **Spray Parameters Frame**: Controls for Sprayer Angle, Tip to Surface Distance, Spray Width Factor, and Spray Profile Type.
* **Analyte Simulation Controls Frame**: Controls for Initial Analyte Amount on Ridges, Dissolution Rate Constant, and Diffusion Coefficient.
* **Schematic Window**: Visualizes the DESI emitter setup based on spray parameters.
* **Show Ridge/Analyte Overlay**: Checkbox to toggle overlays on the plots showing ridge locations and where analyte remains on the surface.
* **Time Step Slider**: Advances the simulation time.
* **Initialize/Restart Simulation Button**: Resets and starts the simulation with current GUI settings.
* **Download Plot PNG Button**: Saves the current visualization.

**8. Visualization**

The plot\_flow function generates a Matplotlib figure embedded in the GUI with two subplots (Figure 1):

* **Left Subplot**: Displays the x-component of the solvent velocity ($ u\_x $) using a 'jet' colormap.
* **Right Subplot**: Displays the analyte concentration in the fluid using a 'hot\_r' colormap, with a color scale maximum (vmax) relative to the initial analyte amount for consistent visualization.

Both plots can optionally overlay the locations of solid ridges and the locations where analyte still remains on the surface, indicated by colored dots.

**Figure 1: Example visualization output from the simulation. The left panel shows the solvent flow velocity magnitude/component, and the right panel shows the dissolved analyte concentration. (Placeholder for figure generated by the code).**

**9. Limitations and Future Work**

The current simulation provides a simplified 2D model with several approximations:

* **2D Domain**: Fingerprint topography and DESI spray are inherently 3D.
* **Single-Phase Fluid**: DESI involves liquid droplets in a gas stream; surface wetting and spreading are crucial aspects not fully captured by a single-phase model.
* **Simplified Boundary Conditions**: Inflow, outflow, and solid boundary conditions are approximations of standard LBM methods.
* **Simplified Analyte Model**: Advective transport by the fluid velocity is currently not included, which is a primary mechanism for analyte relocation. Dissolution is a simple rate model. Solubility limits are not included.
* **No Evaporation**: Solvent evaporation, which concentrates analytes, is not modeled.
* **No Gas Flow**: The nebulizing gas stream is not explicitly simulated.

Future work could involve extending the model to 3D, implementing multi-phase LBM for droplet and wetting dynamics, using more accurate boundary conditions, incorporating advective transport for the analyte, adding evaporation, and modeling the gas phase. Performance optimization via Numba or GPU computing would also be necessary for more complex models or larger grids.

**10. Conclusion**

This simulation provides a foundation for exploring DESI solvent-surface interactions on patterned substrates using the Lattice Boltzmann Method. By coupling LBM fluid dynamics with a simplified model for analyte dissolution and diffusion, it allows for interactive visualization of key transport processes influenced by geometry.