A Minimal Tube-Network Model for Lake Star Formation

Adam Rybicki[1], Jakub Cwynar[1], Marek Spychała[1]
Supervisor: prof. dr hab. Piotr Szymczak[1]

[1]Faculty of Physics, University of Warsaw

June 2026

Abstract

This report presents a comprehensive numerical and theoretical investigation of the formation of lake stars using a minimal tube-network model. We thoroughly present the model alongside its assumptions about the system. Furthermore, we present the numerical implementation of this model. The results of the simulations, despite the apparent simplicity of the model, accurately represent the formation of real lake stars.

1 Introduction

Lake stars represent a striking self-organizing phenomenon observable on frozen lakes during early winter or spring thaws. They form when a sudden drop in temperature rapidly freezes a body of water while the ice layer is still relatively thin. Next, warmer temperatures rise and the ice layer warms. Thus, small holes in the ice form, allowing water to leak through the ice. Relatively warm water from the lower layers of a lake then flows upward through the hole in the ice sheet. Upon reaching the surface, water does not flow over a clean ice sheet, but rather through a fluid-saturated layer of ice slush mixed with snow.

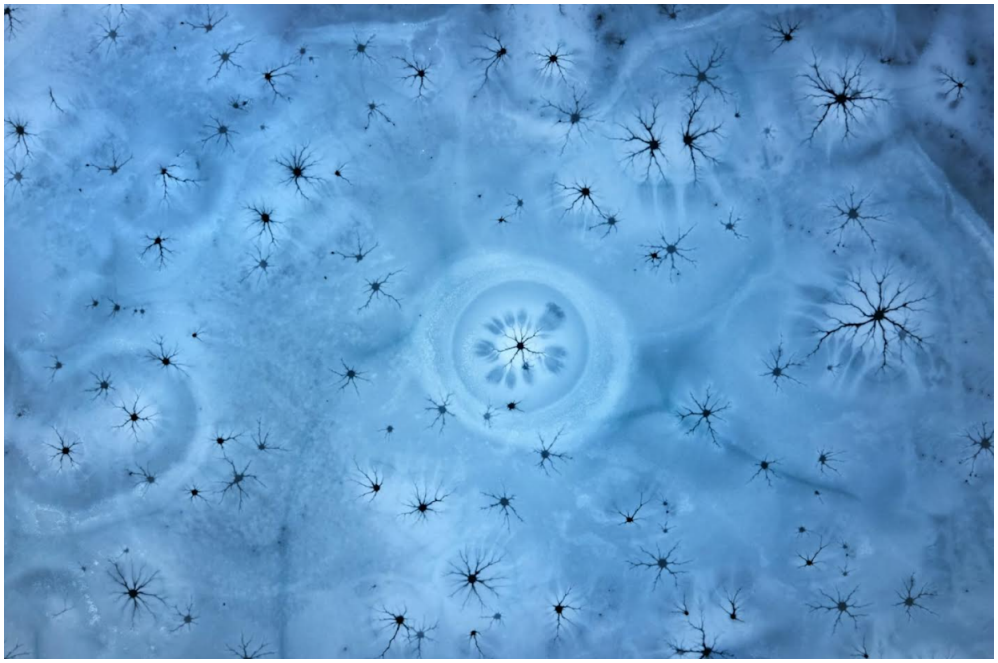

Figure 1: Ice sheet filled with lake stars

As the warm water flows radially outward through the slush, it continuously transfers heat to the surrounding ice matrix, causing the local boundaries of its flow paths to melt.

The physics of the system is governed by a highly non-linear positive feedback loop. As the path boundaries melt, the tube diameter rapidly increases, which drastically enhances its local hydraulic conductance. Due to this increase in conductance, a larger fraction of the global fluid flux is drawn into the already widened paths. The increased flow rates carry more thermal energy further into the channel, accelerating the melting rate relative to adjacent, unaltered pathways. This instability caused by the positive feedback loop breaks the initial radial symmetry of the system, focusing the flow into a limited number of paths. As these paths grow, they eventually branch out, forming star-like patterns.

We propose a minimal model in which the slush layer is represented as a two-dimensional network of nodes connected by cylindrical tubes of fixed length and time-dependent diameter. The water is assumed incompressible, the flow laminar, and the tube walls are held at the melting temperature T_m. Despite its simplicity, this model captures all the essential properties of the formation of lake stars.

2 Mathematical formulation of the Model

We now discuss the proposed model in detail.

2.1 Flow on the network

The continuous slush layer is modeled as a disordered two-dimensional network, in which the nodes are randomly distributed, represented by a graph $G = (V, E)$, where V denotes the nodes and E represents the edges (tubes) connecting the nodes. Let an edge $e = (ij)$ connect nodes i and j. It is characterized by a fixed length l_e and a time-dependent diameter $d_e(\tau)$. Initially, all edges have the same diameter, while the lengths vary. The fluid is taken to be incompressible, and the flow is strictly laminar. The volumetric flux q_e within any given tube is determined by the Hagen-Poiseuille law:

$$q_e = -\frac{\pi d_e^4}{128\mu l_e}(P_i - P_j), \tag{1}$$

where μ is the dynamic viscosity and P_i is the pressure at node i. Since we want to simulate the model numerically, it is convenient to use dimensionless quantities. Thus we define:

$$D_e = \frac{d_e}{d_0}, \quad L_e = \frac{l_e}{l_0}, \quad Q_e = \frac{q_e}{q_0}, \tag{2}$$

where the reference diameter d_0,the reference length l_0 and the reference flow rate q_0 were used. Additionally, we can define dimensionless pressure:

$$\tilde{P}_i = \frac{\pi d_0^4}{128\mu l_0 q_0} P_i \tag{3}$$

Now, for an edge e, the flux is given by:

$$Q_e = \frac{D_e^4}{L_e}(\tilde{P}_i - \tilde{P}_j). \tag{4}$$

To satisfy mass conservation across the system, the net volumetric flux at every internal node i (not on the edge of the system) must satisfy Kirchhoff's law:

$$\sum_{e \ni i} Q_e = 0. \tag{5}$$

2.2 Heat transport along a tube

We now consider heat transport in the network. By $T_e(x)$ we denote bulk water temperature along the edge e and $x \in [0, l_e]$. Moreover, walls of the tubes are assumed to be held at a constant temperature, equal to the melting temperature ($T_{wall} = T_m$). Thermal transport is coupled to the network by dimensionless temperature excess:

$$\theta = \frac{T - T_m}{T_{lake} - T_m}, \tag{6}$$

where T_{lake} is the temperature of the water coming from the source. To balance advective heat transport against radial heat loss to the wall, the following relation must hold:

$$\rho_w c_{p,w} q_e \frac{dT_e}{dx} = -\pi d_e h_e (T_e - T_m), \tag{7}$$

where ρ_w is the water density, $c_{p,w}$ is the water heat capacity and h_e is the heat-transfer coefficient. To satisfy the requirement of laminar flow, we take:

$$h_e = \frac{\mathrm{Nu} k_w}{d_e}, \tag{8}$$

where thermal conductivity of water k_w and Nusselt number Nu were introduced. Now, the temperature decays exponentially along the direction of the flow.

$$\theta_e(X) = \theta_e^{in} \exp\left(-\frac{\mathrm{Da}_T}{|Q_e|} X\right), \tag{9}$$

where $X = x/l_0$ and $\mathrm{Da}_T = (\pi \mathrm{Nu} k_w l_0)/(\rho_w c_{p,w} q_0)$ is the thermal Damköhler number. This number measures the ratio of radial heat transfer to advective heat transport. Therefore, we have:

$$\theta_e^{out} = \theta_e^{in} \exp\left(-\frac{\mathrm{Da}_T L_e}{|Q_e|}\right) \tag{10}$$

2.3 Mixing at nodes

At each internal node where multiple tubes meet, instantaneous enthalpy mixing is assumed. The resulting mixed node temperature θ_j is found by mixing the incoming enthalpy fluxes (some tubes are incoming, some are outgoing). The incoming temperatures are weighted by their respective volumetric fluxes. With the sum running over all edges carrying flow into node j, we have:

$$\theta_j = \frac{\sum_{e \to j} |Q_e| \theta_e^{out}}{\sum_{e \to j} |Q_e|}. \tag{11}$$

This node temperature θ_j then serves as the inlet temperature $\theta_{e'}^{in}$ for all edges e' originating from node j.

2.4 Edge-averaged diameter evolution from melting

Radial melting rate at position x along the tube is given by the local Stefan law:

$$\rho_{ice} L_{eff} \partial_t \left(\frac{d_e(x,t)}{2}\right) = h_e \left(T_e(x,t) - T_m\right) \tag{12}$$

When the fluid moves downstream along the tube, it cools down and consequentially the local melting rate is not uniform along the edge. Up until this point, however, we assigned each edge a single diameter $d_e(t) \neq d_e(x,t)$. We will continue with this assumption, instead computing the

edge-averaged growth rate by equating the total heat lost by the fluid along the edge with the latent heat required to increase the diameter of a cylindrical tube of a uniform diameter $d_e(t)$ over its full length l_e. Therefore, the total heat loss rate along edge e is:

$$\dot{Q}_e^{(heat)} = \rho_{ice} L_{eff} \frac{\pi d_e l_e}{2} \dot{d}_e, \tag{13}$$

where L_{eff} is the effective enthalpy cost of melting. In the simplest case, we can assume it to be equal to the latent heat of fusion: $L_{eff} = L_f$. Whenever the surrounding solid lies initially below T_m, to include the heating of ice from its initial temperature T_s to the melting point, we can instead use:

$$L_{eff} = L_f + c_{p,i}(T_m - T_s) \tag{14}$$

After edge-averaging, the growth law is:

$$\dot{d}_e = \frac{2\rho_w c_{p,w} |q_e|}{\rho_{ice} L_{eff} \pi d_e l_e} \left(T_e^{in} - T_e^{out} \right) \tag{15}$$

Using the exponential cooling law, we obtain in a dimensionless form:

$$\frac{dD_e}{d\tau} = \frac{|Q_e|\, \theta_e^{in}}{D_e L_e} \left[1 - \exp\left(-\frac{\mathrm{Da}_T L_e}{|Q_e|} \right) \right], \tag{16}$$

where the dimensionless time $\tau = t/t_0$ was introduced. It is based on the melting time scale $t_0 = (\pi d_0^2 l_0)/(2q_0 \mathrm{Ste})$, where:

$$\mathrm{Ste} = \frac{\rho_w c_{p,w} (T_{lake} - T_m)}{\rho_{ice} L_{eff}} \tag{17}$$

is the Stefan number.

3 Numerical implementation

The numerical implementation of the lake star model is carried out through a discrete-time simulation. At each time step, the system undergoes a sequence of operations that evolve the state of the network. Before the simulation itself, we set its parameters, which are the number of nodes n_{points}, the number of sinks n_{sinks}, the radius of the graph r, the initial diameter of the tubes D_0, the water temperature at the entrance θ_0, the thermal Damköhler number Da_T, the time of the simulation t, and the time step dt.

We begin by preparing the graph. All its nodes are located inside a circle of radius r and all the sinks are located uniformly on its edge. The radii and angles of all points in polar coordinates are generated randomly to ensure a uniform spatial distribution. Nodes are connected to each other using Delaunay triangulation.

The basis of the simulation is an iterative algorithm in which each time step can be divided into four steps:

1. **Finding flows**: The algorithm first solves the linear system for pressures in the nodes. Next, dimensionless volumetric flow rates, Q_e, are calculated for every edge. There are two boundary conditions determining the evolution of the system, both setting pressure equal to 0. The first one, constant in time, sets $P = 0$ at the edge of the system (namely all the sinks). The second boundary condition, which evolves together with the graph, imposes $P = 0$ at all the nodes occupied by water, thus expanding radially with time.

2. **Tube filling**: Using the calculated flows, the algorithm "fills" all the tubes by updating the volumes filled. The added volume equals the product of the time step and the corresponding flow rate.

3. **Temperature propagation and water mixing**: This step propagates the fluid temperature along the length of each tube. This heat transport is modeled using the exponential cooling law. Furthermore, at every internal node where multiple edges meet, the outgoing water temperature is determined. The temperatures of all outgoing tubes originating from that specific node are set to this temperature.

4. **Tube diameter evolution**: Finally, the tube diameters are updated using the edge-averaged melting equation.

Within this minimal model, the main morphological control parameter is the thermal Damköhler number Da_T . A small value of Da_T corresponds to weak cooling along the flow paths and tends to produce more diffuse melting, while a large value of Da_T promotes localized flow focusing and branched structures. On the other hand, the Stefan number Ste mainly sets the physical time scale of the evolution.

4 Results

The numerical simulations of the minimal model successfully capture the emergence of spider-like structures characteristic of lake stars. By tracking the spatial and temporal evolution of the fluid temperature and tube diameters, we can demonstrate the formation of complex patterns. Simulations were conducted for networks with one or two water sources.

4.1 Single-source model

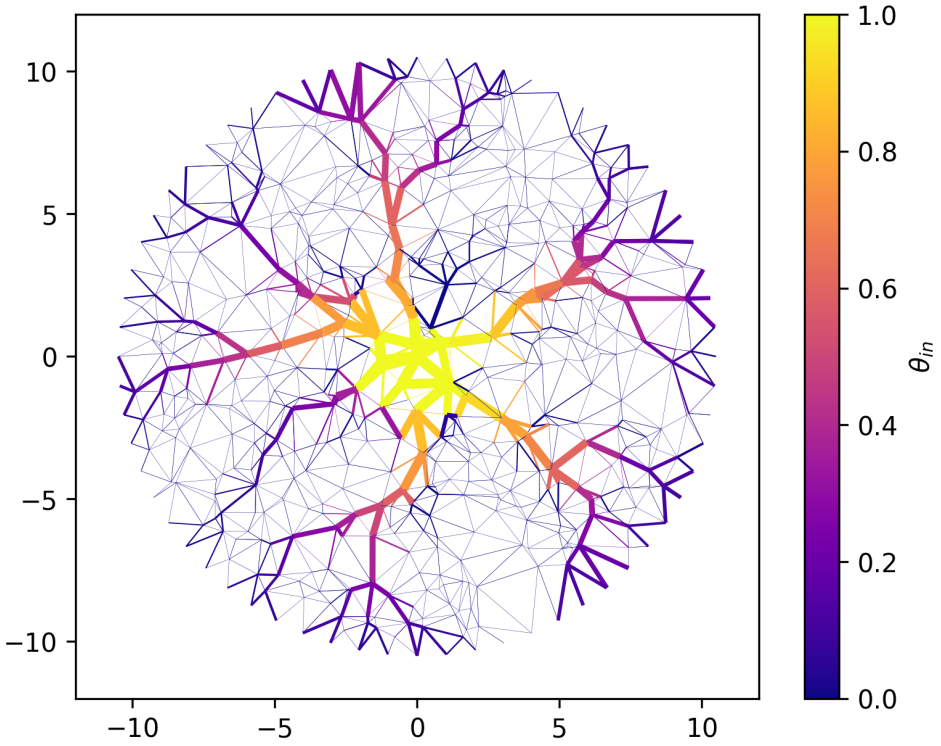

Figure 2: Numerical simulation of a lake star formation originating from a single central source on a graph with 512 nodes. The colors represent the dimensionless temperature field (θ), while the thickness of the lines corresponds to the expanded tube diameters D_e

4.2 Two-source model

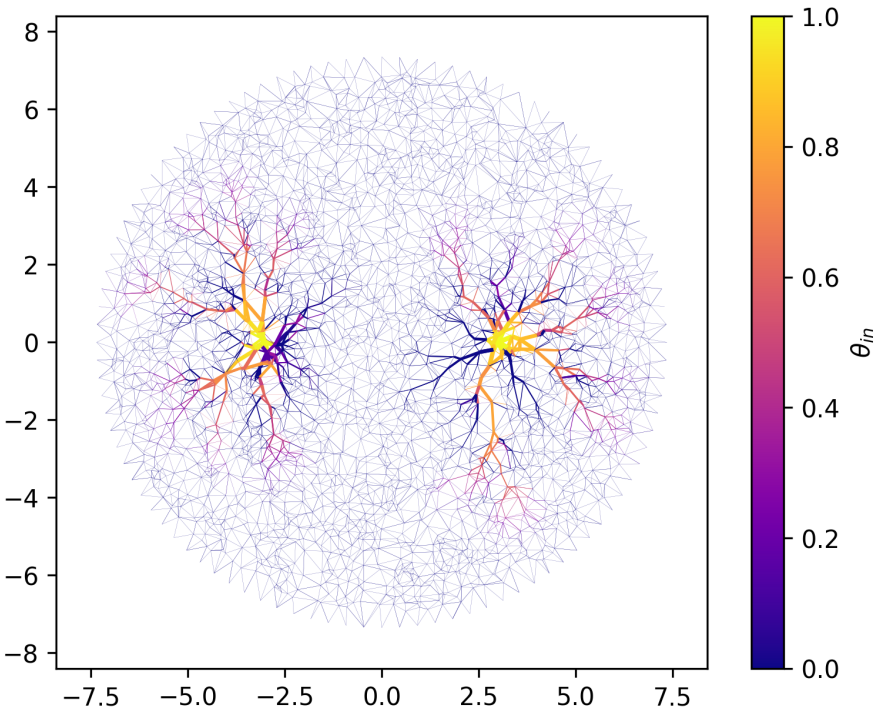

Figure 3: Evolution of the tube-network model featuring two distinct warm water sources on a graph with 3000 nodes. The interaction between the expanding thermal fronts alters the localized flow focusing, leading to asymmetric branch development.

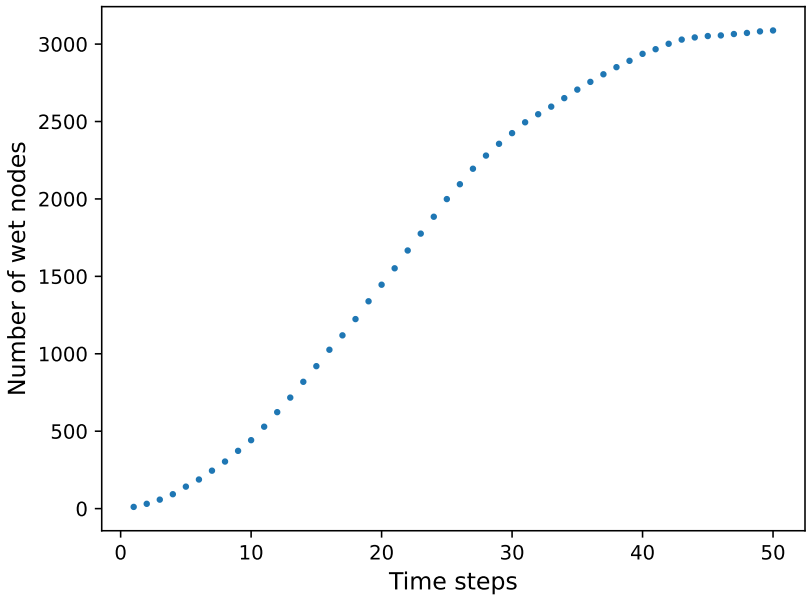

Figure 4: Evolution of the number of flooded (wet) nodes.

5 Summary

In summary, this report presents a comprehensive investigation into the formation of lake stars using a minimal, two-dimensional tube-network model. By coupling Hagen-Poiseuille fluid flow, advective heat transport, and localized tube melting governed by the Stefan law, we successfully captured the core physics driving this phenomenon. The underlying mechanism, a positive feedback loop where initially widened channels draw more warm water, thus accelerating their own melting, was clearly demonstrated in our numerical simulations.

Furthermore, our study explored both single-source and multiple-source configurations. The simulations revealed how initial radial symmetries are broken by flow focusing, leading to the characteristic spider-like branching structures observed in nature. In the multi-source scenario, we also observed complex interactions between expanding thermal fronts that further altered the localized flow dynamics. Despite simplifying assumptions, such as a fixed-length discrete network and purely laminar flow, the model proved highly effective in reproducing the complex morphological features of lake stars.

References

[1] A. Budek, P. Szymczak, *Network models of dissolution of porous media* `https://www.fuw.edu.pl/~piotrek/publications/network.pdf`

A Python Source Code

This appendix contains the Python implementation of the numerical model used to generate the simulation results presented in this report.

Listing 1: Python implementation of the tube-network model

```
import matplotlib.pyplot as plt
import numpy as np
from numba import njit, prange
import os
import imageio
from matplotlib.patches import Circle
from matplotlib.animation import FuncAnimation, PillowWriter
from numba.typed import List
import matplotlib.cm as cm
import matplotlib.colors as colors
from mpl_toolkits.axes_grid1.inset_locator import inset_axes
from scipy.spatial import Delaunay
from scipy.sparse.linalg import spsolve, cg
from scipy.sparse import csc_matrix
from matplotlib.collections import LineCollection
from scipy.sparse.linalg import lsqr
from scipy.sparse import diags, csr_matrix
from scipy.sparse.linalg import spilu, LinearOperator

def create_graph(npoints,nsinks,radius,D0,theta0,holes,source_str):
    holes_num = len(holes)
    nodes = np.zeros((npoints+nsinks+holes_num,2))
    nodes[:holes_num] = holes
    angles = 2*np.pi*np.random.rand(npoints)
    radii = radius*np.sqrt(np.random.rand(npoints))
    nodes[holes_num:npoints+holes_num] = np.column_stack((radii*np.cos(
        angles),radii*np.sin(angles)))
```

```
    nodes[npoints+holes_num:] = 1.05*radius*np.array([[np.cos(i*2*np.pi/
        nsinks),np.sin(i*2*np.pi/nsinks)] for i in range(nsinks)])

    simplices = Delaunay(nodes).simplices

    size_nodes = np.shape(nodes)[0]
    where_holes = np.arange(holes_num)
    source = []
    for i in range(size_nodes):
        if i in where_holes:
            source.append(source_str)
        elif i<npoints+holes_num:
            source.append(0)
        else:
            source.append(-1)
    source = np.array(source)


    iterations = [[0,1],[0,2],[1,2]]

    edges = []
    for e in simplices:
        for iteration in iterations:
            if e[iteration[0]]<npoints+holes_num-1 or e[iteration[1]]<
                npoints+holes_num-1:
                 edges.append([e[iteration[0]],e[iteration[1]]])

    edges = np.array(edges)
    edges = np.unique(edges,axis=0)
    edges = edges

    size_edges = np.shape(edges)[0]

    incidence = np.zeros((size_nodes,size_edges))
    for i in range(size_edges):
        edge = edges[i]
        if np.linalg.norm(nodes[edge[0]])>np.linalg.norm(nodes[edge[1]])
            :
            incidence[edge[0],i] = 1
            incidence[edge[1],i] = -1
        else:
            incidence[edge[0],i] = -1
            incidence[edge[1],i] = 1

    lengths = np.linalg.norm(nodes[edges[:,0]]-nodes[edges[:,1]],axis=1)
    diameters = D0*np.ones(size_edges)
    thetas_in = np.zeros(size_edges)
    idx = []
    for i in where_holes:
        idx += list(np.where(incidence[i,:]!=0)[0])
    thetas_in[idx] = theta0

    no_outlet = []
    for i in range(len(edges)):
        if np.max(edges[i])<=npoints+holes_num-1:
            no_outlet.append(True)
        else:
            no_outlet.append(False)
```

```
82      no_outlet = np.array(no_outlet)
83
84      return nodes,edges,idx,size_nodes,size_edges,no_outlet,lengths,
            diameters,thetas_in,incidence,source
85
86  def find_flow_pressure(base,Delta,D,S,row_idx,size_nodes,wet_nodes):
87      M = base.copy()
88      M.data *= (D**4)[row_idx]
89      A = Delta @ M
90      P = np.zeros(size_nodes)
91      wet_idx = np.array(sorted(wet_nodes))
92      P[wet_idx], _ = cg(A[wet_idx][:,wet_idx],S[wet_idx],x0=None,rtol=1e
            -12)
93      return M @ P, P
94
95  def find_theta_in(inlets,outlets_T,thetas_in,F_abs,decay):
96      theta_out = thetas_in*decay
97      num = inlets @ (F_abs*theta_out)
98      den = inlets @ F_abs
99      factor = np.divide(num,den,out=np.zeros_like(num),where=den!=0)
100     return outlets_T @ factor
101
102 def D_evolve(thetas_in,F_abs,D,L_inv_array,dt,decay,k_melt=1.0):
103     melt = k_melt*F_abs*thetas_in*(1-decay)*L_inv_array
104     D_new = D+dt*melt
105     return D_new
106
107 def plot_all(radius,nodes,incidence,diameters,thetas_in,abs_flow):
108     cmap1 = plt.get_cmap('plasma')
109     norm1 = colors.Normalize(vmin=np.min(thetas_in),vmax=np.max(
            thetas_in))
110
111     vmin_val = max(1e-5,np.min(abs_flow[abs_flow>0]))
112     cmap2 = plt.get_cmap('viridis')
113     norm2 = colors.LogNorm(vmin=vmin_val,vmax=np.max(abs_flow))
114
115     fig, ax = plt.subplots()
116     ax.set_xlim(-1.2*radius,1.2*radius)
117     ax.set_ylim(-1.2*radius,1.2*radius)
118     tails = np.argmax(incidence == -1,axis=0)
119     heads = np.argmax(incidence ==  1,axis=0)
120     start = nodes[tails]
121     end = nodes[heads]
122     segments = np.stack([start,end],axis=1)
123     lc = LineCollection(segments,linewidths=0.1*diameters/np.min(
            diameters),colors=cmap1(norm1(thetas_in)))
124     ax.add_collection(lc)
125     sm1 = cm.ScalarMappable(norm=norm1, cmap=cmap1)
126     plt.colorbar(sm1,ax=ax,label=r'$\theta_{in}$')
127     ax.set_aspect('equal')
128     plt.show()
129
130     fig, ax = plt.subplots()
131     ax.set_xlim(-1.2*radius,1.2*radius)
132     ax.set_ylim(-1.2*radius,1.2*radius)
133     tails = np.argmax(incidence == -1,axis=0)
134     heads = np.argmax(incidence ==  1,axis=0)
135     start = nodes[tails]
```

```
136     end = nodes[heads]
137     segments = np.stack([start,end],axis=1)
138     lc = LineCollection(segments,linewidths=0.1*diameters/np.min(
            diameters),colors=cmap2(norm2(abs_flow)))
139     ax.add_collection(lc)
140     sm2 = cm.ScalarMappable(norm=norm2, cmap=cmap2)
141     plt.colorbar(sm2,ax=ax,label=r'Flow')
142     ax.set_aspect('equal')
143     plt.show()
144
145
146 npoints = 3000
147 nsinks = 100
148 radius = 7
149 D0 = 0.1
150 theta0 = 1
151 P_in = 10000
152 source_str = 50
153 holes = np.array([[3,0],[-3,0]])
154
155 Da = 5
156 time = 50
157 dt = 0.01
158 steps = int(time/dt)
159 Das = np.linspace(0.1,2,5)
160 Das = [1.0]
161
162 for Da in Das:
163     print(f'Da={Da}')
164     (nodes,edges,idx,size_nodes,size_edges,no_outlet,
165      lengths,diameters,thetas_in,incidence,source) = create_graph(
             npoints,nsinks,radius,D0,theta0,holes,source_str)
166
167     current_volumes = np.zeros(size_edges)
168     wet_nodes = set([0,1])
169
170     incidence = csr_matrix(incidence)
171     incidence_array = incidence.toarray()
172     incidence_T = incidence.T
173     lengths_inv_sparse = diags(1.0/lengths)
174     lengths_inv_array = 1.0/lengths
175     base = (lengths_inv_sparse @ incidence_T).tocsr()
176     row_counts = np.diff(base.indptr)
177     row_idx = np.repeat(np.arange(base.shape[0]),row_counts)
178
179     for step in range(steps):
180         if step%5==0 and step!=0:
181             print(step)
182             plot_all(radius,nodes,incidence_array,diameters,thetas_in,
                    flow_abs)
183
184         flow, pressure = find_flow_pressure(base,incidence,diameters,
                source,
185                                                 row_idx,size_nodes,
                                                        wet_nodes)
186         flow_abs = np.abs(flow)
187         directions = -incidence_array*np.sign(flow)
188         inlets = csr_matrix((directions+1)//2)
```

```
189         outlets_T = csr_matrix((1-directions)//2).T
190 
191         safe_flow = flow_abs+(flow_abs == 0)*1e-15
192         decay = np.exp(-Da*lengths/safe_flow)
193         thetas_in = find_theta_in(inlets,outlets_T,thetas_in,flow_abs,
                decay)
194         thetas_in[idx] = theta0
195         diameters = D_evolve(thetas_in,flow_abs,diameters,
                lengths_inv_array,dt,
196                              decay,k_melt=10)
197 
198         current_volumes += flow_abs*dt
199         max_volumes = np.pi*((diameters/2)**2)*lengths
200 
201         for i in range(size_edges):
202             node_a, node_b = edges[i]
203             if (node_a in wet_nodes or node_b in wet_nodes) and
                    current_volumes[i] >= max_volumes[i]:
204                 wet_nodes.add(node_a)
205                 wet_nodes.add(node_b)
206         print(step,len(wet_nodes))
207 
208     plot_all(radius,nodes,incidence_array,diameters,thetas_in,flow_abs)
```