A review and implementation of *Advection, diffusion and delivery over a network* on python

Sergio Villamarin
Tulane University

Abstract: Different disciplines use some form of the advection-diffusion equation with delivery (ADED) to simulate transport processes. It is used on geophysical models to simulate landscape evolution driven by water and sediment, and in biology to simulate glucose delivery on blood vessels. I will present a review of the paper by [1], a discussion on the details and challenges of the python implementation, examples solving the heat equation on a single edge, and finally I will reproduce the example from figure 4 of [1].

Keywords: ***advection, diffussion, delivery, transport equation, python***

1 Introduction

The advection-diffusion equation with delivery (ADED), sometimes refereed simply as convection-diffusion equation, is used to model the distribution of some quantity of interest (e.g. heat, sediment volume, glucose concentration) being transported by a medium (a metal rod, a river's stream, blood on vessels) by means of advection (bulk displacement of the quantity by the medium), diffusion (spreading of the quantity driven by the differences in concentration in the medium) and delivery (filtration or infiltration of the quantity from outside the system).

In practice, solving transport equations numerically on a 2 or 3 dimensional domain can be computationally expensive since the numerical stability requires a relatively fine mesh [2–5], and also resource expensive as real life applications need boundary and initial conditions over a full grid. To make this situation feasible several simplifications are usually made including sampling and simulating only on regions of interest, assuming time, width, lenght, or elevation averaged equations.

Most of the systems where applications of ADED are used have a natural network-like structure (e.g. water and sediment on river basins, blood and glucose on blood vessels). Because of this, the authors of [1] propose to model these systems using a 1 dimensional network in such a way so that every edge of the network can be assumed to have some spatially constant properties at least over a small period of time and each edge solves a one dimensional ADED that is coupled through its boundary conditions to the whole network.

This report splits into 3 more sections. The methodology section further splits into two parts, the first one giving a description of the model from [1] and the theoretical tools used in the numerical solution, and the second one explaining how all of this was implemented in python along with what considerations were made for efficiency and architecture. A section where I discuss 2 examples, one solving the heat equation and another one reproducing the results from figure 4 of [1]. Finally, a discussion/future direction section where I talk about how the implementation can improve its efficiency, and how this model can be applied to sediment flux on a river basin system.

2 Methodology

2.1 ADED over a network

2.1.1 Setup

One can think of this model as a network of tubes, through which a fluid is moving, connected to each other, and we're interested in the time evolution of the concentration of a given resource in the fluid all over the network. The model in [1] starts with these assumptions about the edge connecting nodes i and j :

- The cross sectional area of the edge $S_{ij}(t)$ is piecewise constant.
- The length l_{ij} does not vary over time.
- The mean velocity along the edge $u_{ij}(t)$ is also piecewise constant and its sign reflects direction (positive from i to j).
- Resource is delivered outside the network at a rate R_{ij}.
- The dispersion coefficient $D_{i,j}(t)$ is piece-wise constant and may depend on the on the velocity of the fluid.

Having these, the fundamental 1 dimensional ADED they are trying to solve is:

$$\frac{\partial q_{ij}}{\partial t} = D_{ij}\frac{\partial^2 q_{ij}}{\partial x^2} - u_{ij}\frac{\partial q_{ij}}{\partial x} - R_{ij}q_{ij}$$

where q_{ij} is the amount of resource per unit length.

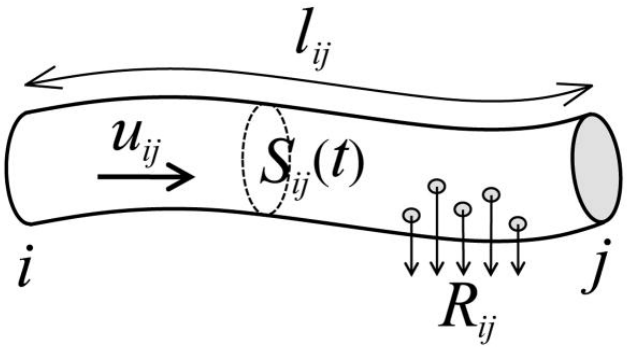

FIGURE 1. Taken from [1]. Properties of a single edge on a resource distribution network. Notice the diffusion coefficient D_{ij} is not represented here but is also being accounted

Notice that if there's no edge connecting nodes i, j then we might assume all parameters and q_{ij} to be zero. Now, to actually make this into a network system, they force consistent boundary conditions for all these equations. Assuming perfect mixing at the nodes, the concentration at the nodes i, j must satisfy:

$$c_i(t) = \frac{q_{ij}(0,t)}{S_{ij}(t)} \quad \text{and} \quad c_j(t) = \frac{q_{ij}(l_{ij},t)}{S_{ij}(t)}.$$

Now, for initial conditions, it is assumed we known $I_i(t)$, the net rate at which resource leaves node i. Applying Fick's law we can express this as

$$I_i(t) = \sum_j \left[u_{ij} q_{ij} - D_{ij} \frac{\partial q_{ij}}{\partial x} \right]_{x=0}$$

where we are adding the rate at which resource leaves node i through the edge i, j for every j connected to i.

2.1.2 *Solution of an initially empty network on Laplace space*

From now on they will assume that all piece-wise constant coefficients are constant and later explain how to solve the system of equations when these change over time. They consider the equivalent system in Laplace space and instead attempt to solve the system

$$-D_{ij} \frac{\partial^2 Q_{ij}}{\partial x^2} + u_{ij} \frac{\partial Q_{ij}}{\partial x} + (s + R_{ij}) Q_{ij} = q_{ij}(x, 0)$$

with boundary conditions

$$\Upsilon_i(s) = \sum_j \left[u_{ij} Q_{ij} - D_{ij} \frac{\partial Q_{ij}}{\partial x} \right]_{x=0}$$

where $\mathcal{L}(q_{ij}(x,t)) = Q_{ij}(x,s)$ and $\mathcal{L}(I_i(x,t)) = \Upsilon_i(x,s)$. The next big step towards finding the solution involves having a methodology to find the solution of the homogeneous Laplace system, namely when $q_{ij}(x,0) = 0$. In this case we have an homogeneous ODE with constant coefficients (as a function of x) and since D_{ij} and s are not both zero, then the solution of the system has the general solution of the system has the form

$$Q_{ij}(x,s) = A(s) e^{\lambda(s)} + B(s) e^{\bar{\lambda}(s)}.$$

Before we expand on the values of the coefficients and exponents, we introduce the quantity of resource in Laplace space at the end of each edge

$$X_{ij}(s) = Q_{ij}(0,s) = Q_{ji}(l_{ij}, s),$$

and the dimensionless ratios

$$g_{ij} = \frac{u_{ij} l_{ij}}{2D_{ij}} \quad \text{and} \quad h_{ij}(s) = \frac{\alpha_{ij}(s) l_{ij}}{2D_{ij}}$$

where $\alpha_{ij}(s) = \sqrt{u_{ij}^2 + 4D_{ij}(s + R_{ij})}$. Having these, the solution can be expressed as

$$Q_{ij}(x,s) = A e^{\frac{u_{ij} + \alpha_{ij}}{2D_{ij}} x} + B e^{\frac{u_{ij} - \alpha_{ij}}{2D_{ij}} x}$$

where

$$A = \frac{X_{ji} e^{-g_{ij}} - X_{ij} e^{-h_{ij}}}{2 \sinh(h_{ij})}$$

$$B = \frac{X_{ij} e^{h_{ij}} - X_{ji} e^{-g_{ij}}}{2 \sinh(h_{ij})}$$

We still need a way of connecting X_{ij} with our initial conditions Υ_i. To do this we recall the compatibility relationship of the concentration at the nodes and translate it into Laplace space, so $\mathcal{L}(c_i(t)) = C_i(s)$ and

$$C_i(s) = \frac{X_{ij}(s)}{S_{ij}} \quad \text{and} \quad C_j(s) = \frac{X_{ji}(s)}{S_{ij}}.$$

Finally, noting that

$$\Upsilon_i(s) = \sum_j \frac{\alpha_{ij}}{2}(B - A) + \frac{u_{ij}}{2}(A + B)$$

one can express the C_i's as a solution for a system of equations involving the Υ_i's, namely

$$M(s) \bar{C}(s) = \bar{\Upsilon}(s)$$

where

$$M_{ij}(s) = \begin{cases} \sum_k S_{ik} \left[\frac{u_{ik}}{2} + \frac{\alpha_{ik}}{2 \tanh(h_{ik})} \right] & \text{if } i = j, \\ -\frac{S_{ij} \alpha_{ij} e^{-g_{ij}}}{2 \sinh(h_{ik})} & \text{if } i \neq j. \end{cases}$$

The authors refer to M as the *propagation matrix*. Quoting the authors [1], *Multiplying $|M_{ij}(s)|$ by $C_j(s)$ gives us the Laplace transform of the current of resource flowing from j to i, so roughly speaking, $|M_{ij}(s)|$ represents the size of the volumetric current from j to i, over the time scale 1/s.*

2.1.3 Approach for the general solution

After this, they consider the solution when $q(x,0) \neq 0$. Since this is still a linear ODE one can apply the method of variation of parameters which yields a particular solution to the system as

$$f(x,s,q_{ij}(y,0)) = \frac{e^{(g_{ij}-h_{ij})\frac{x}{l_{ij}}}}{\alpha_{ij}} \int_0^x e^{(h_{ij}-g_{ij})\frac{y}{l_{ij}}} q_{ij}(y,0)\, dy - \frac{e^{(g_{ij}+h_{ij})\frac{x}{l_{ij}}}}{\alpha_{ij}} \int_0^x e^{(-h_{ij}-g_{ij})\frac{y}{l_{ij}}} q_{ij}(y,0)\, dy.$$

One very important property of f which will be used later is the fact that $f(x,s,q_1+q_2) = f(x,s,q_1)+f(x,s,q_2)$. Having a particular solution we can now build a general solution by adding it to the homogeneous system, that is

$$Q_{ij}(x,s) = Ae^{(g+h)\frac{x}{l}} + Be^{(g-h)\frac{x}{l}} + f$$

where now

$$A = \frac{X_{ji}e^{-g_{ij}} - X_{ij}e^{-h_{ij}}}{2\sinh(h_{ij})} + \frac{\beta_{ij}}{\alpha_{ij}}$$

$$B = \frac{X_{ij}e^{h_{ij}} - X_{ji}e^{-g_{ij}}}{2\sinh(h_{ij})} - \frac{\beta_{ij}}{\alpha_{ij}}$$

and

$$\beta_{ij}(s) = \frac{-\alpha_{ij}e^{-g_{ij}}}{2\sinh(h_{ij})} f(l_{ij},s,q_{ij}(y,0)).$$

Now, using the same idea as before they show that

$$M(s)\bar{C}(s) = \bar{\Upsilon}(s) + \bar{\beta}$$

where M is the same propagation matrix as in the homogeneous system and $\bar{\beta}_i = \left[\sum_j \beta_{ij}\right]$.

The next step consist on noting that all of the influence from the initial condition $q(x,0)$ on the concentration at the nodes is captured by $\bar{\beta}$ and using this to construct a solution from two pieces, one being $\hat{Q}_{ij} = \mathcal{L}(\hat{q}_{ij})$, a solution for the homogeneous system in Laplace space that assumes the boundary conditions are given by $\bar{\Upsilon}(s)+\bar{\beta}$ (an initially empty network pushing concentrations at the nodes equal to the ones given by the boundary conditions of the original system), and the second one $\tilde{q}_{ij} = q_{ij} - \hat{q}_{ij}$, which corresponds solving the same ADED with initial conditions $q_{ij}(x,0)$ but with absorbing boundary conditions $q_{ij}(0,t) = q_{ij}(l_{ij},t) = 0$ (a network initially filled just as the original system but where the boundaries pull the resource out instantly).

2.1.4 Solving ADED with absorving boundary conditions

To solve the system with boundary conditions $q(0,t) = q(l,t) = 0$ the authors use a method similar to Fourier's approach to the heat equation after simplifying the system by means of an integrating factor. Although not mentioned on the paper, the idea is to find an integrating factor $w(x)$ to get rid of the first derivative term of the ADED, namely

$$\frac{\partial^2 q}{\partial x^2}w - \frac{u}{D}\frac{\partial q_{ij}}{\partial x}w = \frac{\partial^2(qw)}{\partial x^2} + Cqw$$

which yields $w = e^{-\frac{u}{2D}x}$ and $C = -\frac{u^2}{4D^2}$. Multiplying the original equation by $\frac{w}{D}$ we get

$$\frac{\partial(qw)}{\partial t} = \frac{\partial^2(qw)}{\partial x^2} - (R + \frac{u^2}{4D^2})(qw)$$

and so we instead solve this simplified system for $q' = qw$. Following Fourier's approach for finding non trivial solutions of this system by separating variables, if we assume that $q'(x,t) = X(x)T(t)$ then we solve the system

$$\lambda = \frac{T'(t)}{T(t)} = \frac{X''(x)}{X(x)} + R + \frac{u^2}{4D^2}$$

whose solution is of the form $q'(x,t) = A_m e^{\lambda_m t}\sin(\frac{m\pi x}{l})$ with $\lambda_m = -\left(m^2\frac{D\pi^2}{l^2} + \frac{u^2}{4D^2} + R\right)$ for some integer m. This means that the general solution of the system is given by

$$\tilde{q}_{ij}(x,t) = e^{\frac{u_{ij}}{2D_{ij}}x}\sum_{m=1}^{\infty} A_m e^{\lambda_{ij}^m t}\sin(\frac{m\pi x}{l_{ij}})$$

where

$$\lambda_m = -\left(m^2\frac{D_{ij}\pi^2}{l_{ij}^2} + \frac{u_{ij}^2}{4D_{ij}^2} + R_{ij}\right)$$

and

$$A_m = \frac{2}{l_{ij}}\int_0^{l_{ij}} e^{-\frac{u_{ij}}{2D_{ij}}x}\sin(\frac{m\pi x}{l_{ij}})q(x,0)\, dx.$$

2.1.5 Piece-wise constant approximations

The approach taken by the authors to implement a numerical solution of the problem involves a piece-wise approximation of the function $q_{ij}(x,0)$. We can write q_{ij} as a linear combination of simple functions, that is, if $\{[x_n, x_{n+1}]\}_n$ is a partition of $[0, l_{ij}]$, then

$$q_{ij}(x,0) \approx \sum_n q_{ij}(x_n,0)\mathbf{1}_{[x_n,x_{n+1}]}(x)$$

and we can use this sum and the additive property of f to approximate

$$\beta_{ij}(s) = \frac{-\alpha_{ij}e^{-g_{ij}}}{2\sinh(h_{ij})}\sum_n f(l_{ij},s,q_{ij}(x_n,0)\mathbf{1}_{[x_n,x_{n+1}]}(y)),$$

and also to approximate the coefficients A_m since we can compute a close form expression of the integrals defining f and A_m. As is usually the case the partition considered is uniform so that for every integer n such that $1 \leq n \leq N$ we have $x_n = \frac{(n-1)}{N}l_{ij}$.

To deal with the situation of piece-wise constant parameters for the network, one simply applies this methodology and solves for the evolution of the system up to the time where the parameters change and then feeds back the output as the initial condition of a new network with the new parameters, an iterates.

2.1.6 *Inverting from Laplace space*

The suggested approach by the authors to invert the solution from the homogeneous Laplace system into the time domain is to use the Gaver-Stehfest algorithm (GS algorithm) but the formulas presented by them are missing a terms, so we refer to [6]. Gaver originally introduced the sequence of approximations

$$\hat{f}_n(x) = \frac{\ln 2}{x} \sum_{i=0}^{n} n \binom{2n}{n} \binom{n}{i} (-1)^i F((n+i)\frac{\ln 2}{x})$$

where $F = \mathcal{L}(f)$, while additionally showing that

$$\hat{f}_n(x) = \int_0^\infty p_n(u) f\left(\frac{xu}{\ln 2}\right)\, du$$

and that $p_n > 0$ integrates to 1, and so, considering p_n as the density function of some random variable U_n Gaver computed

$$E[U_n] = \ln 2 + O(n^{-1}) \quad Var(U_n) = O(n^{-1})$$

which shows that U_n converge in distribution to $\ln 2$, therefore, for any continuous function f and $x > 0$

$$\hat{f}_n(x) = E\left[f\left(\frac{xU_n}{\ln 2}\right)\right] \longrightarrow f(x)$$

as $n \to \infty$. This rate of convergence was later improved by Stehfest by considering a linear combination of Gaver's approximations

$$f_n(x) = \sum_{k=1}^{n} (-1)^{n+k} \frac{k^n}{n!} \binom{n}{k} \hat{f}_k(x)$$

to remove leading error coefficients, making if so that $f_n(x) = f(x) + O(n^{-n+1/2})$.

This error estimation relies on smoothness and integrability assumptions, but it is highlighted by the authors in [1] that empirical evidence shows that the relative error is approximately bounded by $10^{-0.45n}$.

2.2 Python implementation

2.2.1 *The main 3 classes*

I decided to split the functionality of the python module `pde_network` into three objects:

- `InitialConditions`, a public class that contains the initial and boundary conditions for a given network.
- `Solver`, a public class that focuses on the methods and functions from [1] needed to solve the ADED over a network.
- `_f_tools`, a pseudo private class with additional tools not included on [1].

This design is intuitive as it encourages the user to provide an `InitialConditions` object with the set of initial and boundary conditions for a given network, and then use a `Solver` object to solve for the time evolution of the system.

2.2.2 *InitialConditions*

For this and subsequent sections we let n be the number of nodes, Ω the number of grid points on Laplace space, N the number of points for a spatial resolution of $\frac{1}{N-1}$.

Since we will potentially store large amounts of data, we will use `numpy` arrays with double (float64) precision to efficiently store most numerical data. Double precision is required according to the GS-algorithm to have a relative error of about 3 decimal digits when using $\Omega = 10$ grid points on Laplace space.

One creates an `InitialConditions` object by providing:

- Initial conditions $q_{ij}(x, 0)$ as a $n \times n \times N$ numpy matrix in which each ij entry has the initial conditions at N uniformly spaced grid points from 0 to l_{ij}
- Several $n \times n$ numpy matrices corresponding to the coefficients $l_{ij}, R_{ij}, D_{ij}, u_{ij}$ and S_{ij}.
- A python list of n functions for the functions $I_i(t)$.

This way, a regular call of the class constructor looks like

```
ic=InitialConditions(qx, l, R, D, u, S, I)
```

In most situations the network produces a sparse adjacency matrix since most nodes are not connected to each other. To address the impact of this on performance while traversing the network, everytime an `InitialConditions` object is created, it defines an adjacency list for every node i. This allows to efficiently access the entries when doing computations, which additionally prevents methods' and functions' code from becoming messy and clustered or dealing with ValueErrors from logarithms and divisions by zero.

2.2.3 *Solver and _f_tools*

A solver instance takes an `InitialConditions` object as a parameter and has an internal instance of `_f_tools`. The solver class contains several public and pseudo private functions necessary to solve the ADED system. Among them it contains functions to compute the propagation matrix $M(s)$, the β_{ij} matrix, a function to solve the homogeneous Laplace system with homogeneous conditions and another one for non-homogeneous conditions $\hat{q}_{ij}$, a function to compute $\tilde{q}_{ij}$, and another two functions to calculate the total amount of resource for both $\int \hat{q}_{ij}$ and $\int \tilde{q}_{ij}$.

We note that the computation of the GS-algorithm requires only of the evaluation of the Laplace transform at Ω non uniform grid points in Laplace space. This is the reason why we need the functions $I_i(t)$ instead of numerical values. To compute Laplace transform we use the sympy module and find the weights and roots of Laguerre polynomials for Gauss-Laguerre quadratures on the functions $I_i'(t) = I_i(t/s)/s$, where to avoid recomputing these on each iteration, they are saved on the instance of `_f_tools` in each `Solver` object.

A corrected implementation of the GS-algorithm on `_f_tools` was adapted to the specific use of the solver class as implementations from other libraries like scypy and mpmath don't take this type of input and both the performance of adapting the input or the method itself where worse when attempted.

Finally, we use scipy sparse solver to solve for the corresponding sparse system of equations involving the propagation matrix M.

3 Examples

3.1 A simple heat equation

We consider a network with a single edge and no advection ($u = 0$) or delivery ($R = 0$). The only process driving changes in the amount of resource is diffussion, which we set to one ($D = 1$), the length of the edge $l = \pi$, and cross sectional area $S = 1$. The ADED becomes a heat equation in this case

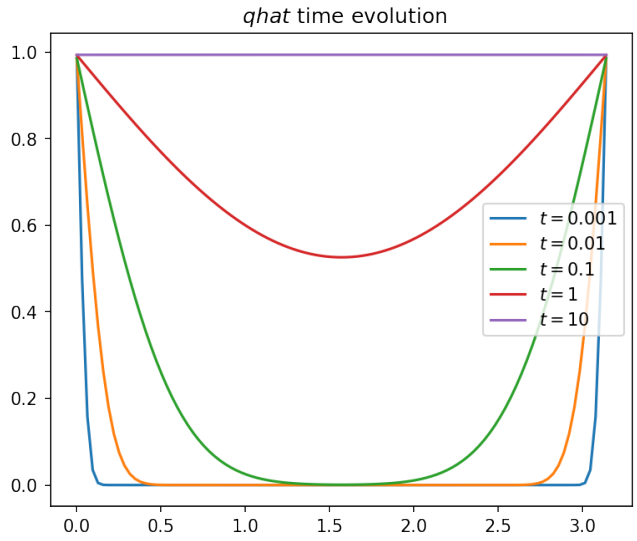

FIGURE 2. Time evolution of $\hat{q}$. An initially empty network with forced boundary concentrations at the nodes (equal to 1). Resource gradually diffuses from the nodes to the center of the edge.

$$\frac{\partial q}{\partial t} = \frac{\partial^2 q}{\partial x^2}$$

and we impose the initial and boundary conditions

$$q(x,0) = 1 + \sin(x), \quad I_0(t) = I_1(t) = -e^{-t}.$$

The exact solution of this system is $q(x,t) = 1 + e^{-t}\sin(x)$, we solve this numerically using the `pde_network` module with a spatial grid of 101 points, $\Omega = 10$ grid points in Laplace space, and solve for $t = 0.001, 0.01, 0.1, 1, 10$.

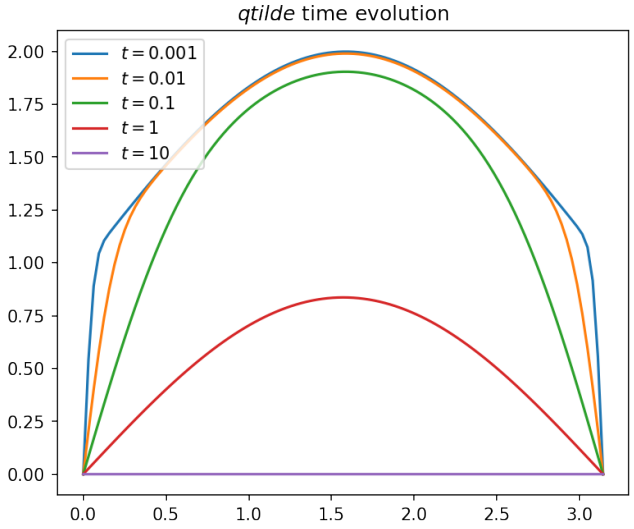

FIGURE 3. Time evolution of $\tilde{q}$. A network equal initial conditions $1 + \sin(x)$ and absorbing boundary conditions. Resource leaks on the boundary depleting the initial distribution of resource.

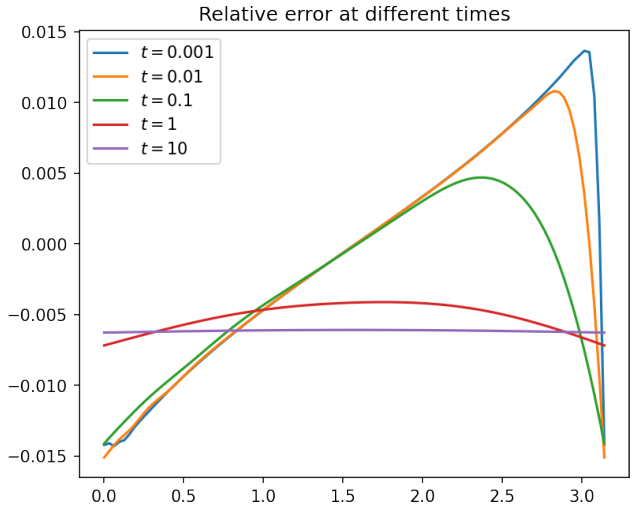

FIGURE 4. Relative error $(q - (\hat{q} + \tilde{q}))/q$ below 2%.

3.2 Figure 4 from [1]

We start by considering a network of blood vessels with three nodes $1, 2, n$ and two edges $(1, 2)$ and $(2, n)$ where a constant volume of fluid $F = 0.002mm^3/s$ is entering through node 1 and exiting at node 2. To reproduce the example from figure 4 we define the Total rate of resource consumption in the

edge $(1, 2)$ as

$$C_{tot} = I_1(t) + I_2(t).$$

Notice that this corresponds to the amount of glucose that remains on the edge at time t, and since we have already reached steady state, then this must correspond to how much is being transported outside of the network (delivery). The molecular diffusion coefficient of glucose $D_m = 6.7 \cdot 10^{-4} mm^2/s$, the lengths $l_{1,2} = 1$ and $l_{2,n} = \infty$, the density of transporters $\rho = 0.05 mm/s$, and the cross sectional areas $S_{1,2} = S_{2,n}$.

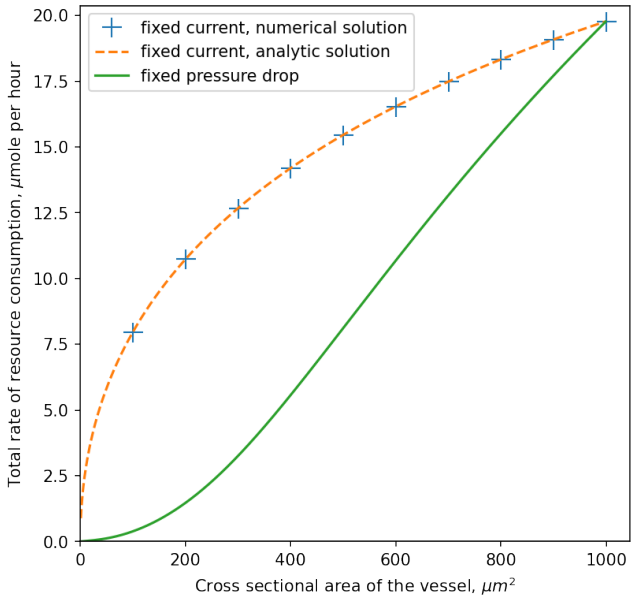

FIGURE 5. Total rate of glucose consumption after reaching steady state as a function of cross sectional area.

The first curve is obtained by getting the medium velocity from the relationship with current and cross sectional area $F = uS$, similarly we obtain the local delivery rate from the cross sectional area and the density of transporters $R = \rho/\sqrt{S}$, next we use Taylor's dispersion coefficient on a cylindrical tube for a small Reynolds number to find the diffusion coefficient $D = D_m + u^2 \frac{r^2}{48 D_m}$ with $r = \sqrt{S/\pi}$ and finally, we compute this for S from 100 to 1000 in multiples of 100. The six points are obtained from the numerical solution and the curve is obtained from the analytical solution provided on [1].

The second curve is obtained from fixing a constant pressure drop instead of the current. This means we apply the Hagen-Poiseuille equation [7] to solve for the velocity of the current using the cross sectional area while fixing Δp as

$$u = \frac{\Delta p}{8\pi\mu l} S$$

where $\mu = 0.007 \cdot 10^{-1} gr/mm \cdot s$ is the kinematic viscosity of water at body temperature, and the fixed pressure drop is set so that the two curves meet at $S = 1000$.

4 Discussion/Future work

I would like to improve on the speed of the algorithm as it currently takes several seconds to run even small networks. One approach could be the use of a custom sparse solver as suggested by the authors [1] using a BiCGStab algorithm with custom initial guess and subsequent iterations. this would also prevent the sparse solver casting which currently takes quite some time. The application of `pandas` data frames for efficient vector operations was considered but since we access data from more than one dimension, the conclusion was that it would be detrimental for storage while possibly not making a difference in speed.

While we are still storing several 0 parameters and a vector of functions, we have quick access from the adjacency matrix and the actual storage limitation comes from the $n \times n \times N \times \Omega$ matrices needed to computed the GS algorithm. The storage on this front could be improved to $n \times n \times N$ if we could manage to calculate the values of the GS-algorithm at a given point in space without having to iterate over all the other ones.

I plan to use this method to find the distribution of sediment on a river network. The type of data for this application is well suited as USGS monitoring stations are placed at several different locations and they are constantly tracking water discharge and sediment discharge, flow velocity, and cross sectional area. Groundwater and infiltration studies will be a good source for information on the delivery of the network, and the relevant scales for diffusion of sediment flux are well studied.

4.1 Reproducibility

The code for this project is currently part of a github repository and the python files needed to run this are located at https://github.com/sergiogyoz/PDE$_N etwork/tree/main/Module$. To be able to run these you need to install all the dependencies used. To reproduce the graphs of the first example you can run the file `test_heat_equation.py`. To reproduce the graph from figure 4 of [1] run the file `test_example_fig4.py`.

1. L. L. M. Heaton, et al., Advection, diffusion, and delivery over a network, Phys. Rev. E 86 (2012) 021905, 10.1103/PhysRevE.86.021905

2. E. Isaacson and H. Keller, Analysis of Numerical Methods, Dover Books on Mathematics (Dover Publications, 2012), URL https://books.google.com/books?id=CqPDAgAAQBAJ.

3. K. Wang and H. Wang, Stability and error estimates of a new high-order compact ADI method for the unsteady 3D convection–diffusion equation, Applied Mathematics and Computation 331 (2018) 140, https://doi.org/10.1016/j.amc.2018.02.053

4. M. Saqib, S. Hasnain, and D. S. Mashat, Computational solutions of three-dimensional advection-diffusion equation using fourth order time efficient alternating direction implicit scheme, AIP Advances 7 (2017) 085306, 10.1063/1.4996341

5. A. R. Appadu, J. K. Djoko, and H. H. Gidey, Comparative study of some numerical methods to solve a 3D advection-diffusion equation, AIP Conference Proceedings 1863 (2017) 030009, 10.1063/1.4992162

6. A. Kuznetsov, On the Convergence of the Gaver–Stehfest Algorithm, SIAM Journal on Numerical Analysis 51 (2013) 2984, 10.1137/13091974X

7. A. Ostadfar, Chapter 1 - Fluid Mechanics and Biofluids Principles, In A. Ostadfar, ed., Biofluid Mechanics, pp. 1–60 (Academic Press, 2016), https://doi.org/10.1016/B978-0-12-802408-9.00001-6, URL https://www.sciencedirect.com/science/article/pii/B9780128024089000016.