Python-Based Tools for Modeling Transport in Porous Media Columns

Boyang Lu
Illinois Institute of Technology
Chicago, Illinois
blu6@hawk.iit.edu

David Lampert
Illinois Institute of Technology
Chicago, Illinois
Dlampert1@iit.edu

ABSTRACT

The fate and transport of dissolved constituents in porous media has important applications in the earth and environmental sciences and many engineering disciplines. Mathematical models are commonly applied to simulate the movement of substances in porous media using the advection-dispersion equation. Whereas computer programs based on numerical solutions are commonly employed to solve the governing equations for these problems, analytical solutions also exist for some important one-dimensional cases. These solutions are often still quite complex to apply in practice, and therefore computational tools are still needed to apply them to determine the concentrations of dissolved substances as a function of space and time. The Python Programming Language provides a variety of tools that enable implementation of analytical solutions into useful tools and facilitate their application to experimental data. Python provides an important but underutilized tool in environmental modeling courses. This article highlights the development of a series of Python-based computing tools that can be used to numerically compute the values of an analytical solution to the one-dimensional advection-dispersion equation. These tools are targeted to graduate and advanced undergraduate courses that teach environmental modeling and the application of Python for computing.

KEYWORDS

Python, Advection-Dispersion Model, Analytical Solution, Column Experiment, Columntracer, Dispersion Coefficient, Breakthrough Curve, Jupyter Notebook, Binder, Educational Computing Tools

1 INTRODUCTION

The fate and transport of dissolved constituents in porous media has many important applications in geology, environmental science, and engineering. Field and laboratory studies are often used to study the fate and transport of contaminants in porous media. These studies also require computational tools for interpretation of data and forecasting of pollutant migration into uncontaminated areas. Since many contaminants released to the environment are eventually trapped in soils and sediments, these media can contribute to the contamination to surface water and groundwater in the vicinity, depending on the contaminant characteristics and site geological properties.

Laboratory columns are widely used to study fate and transport in porous media such as soils and sediments. For example, McKenzie et al. [13] and Høisæter et al. [8] recently conducted column experiments to improve the understanding of per- and polyfluoroalkyl substances, an emerging class of pollutants, in unsaturated soils and groundwater. Perujo et al. [16] carried out a laboratory-scale column experiment to study the interaction between physical heterogeneity and microbial processes in subsurface sediments, and Westerhoff et al. [22] performed column tests for arsenate removal in iron oxide packed bed columns. The main purpose of column experiments is to investigate the transport and attenuation of a specific compound within a specific sediment or substrate [2]. Column experiments are flexible and simple to manage; therefore, it is possible to run a column experiment as part of an educational course. The boundary conditions, physical and chemical properties of the contaminants, media characteristics, and the type of the solvent can be controlled easily during preparation. The resulting data can provide a useful educational experience for students that are learning about fate and transport modeling.

The movement of dissolved constituents in porous media strongly depends on the fluid flow characteristics. In laboratory columns, it is reasonable to assume the flow is one-dimensional. Tracer studies using an inert substance that does not interact with the media are frequently used to assess fluid flow. The results of a tracer study provide data that can be used with an appropriate model to interpret the fluid movement, which can then be used to assess migration of other substances within the media.

Mathematical models based on advection (the movement of a dissolved substance with the bulk media) and dispersion (the dissipation of concentration gradients in the media due to differences in flow path lengths) are often used to simulate the fate and transport of dissolved substances. One-dimensional advection-dispersion models often provide excellent performance in explaining observed concentrations within laboratory columns used to study the movement of dissolved substances within porous media [1]. Students in the earth and environmental sciences and engineering disciplines require substantial training in computational science to apply these models. In addition to knowledge of the underlying physical and chemical processes, these students often also require training in the solution of differential equations and the development of computer programs to perform the calculations. The Python Programming Language provides a convenient platform for solving advection-dispersion problems, since it provides access to many applicable computational and visualization tools; however, limited educational tools are available to teach the applications of Python for environmental modeling.

The one-dimensional dispersion-advection model can be used to simulate the behavior of tracer transport in porous media. An analytical solution for the model has been developed in the Fortran programming language that is described in a report published by the U.S. Geological Survey (USGS), which includes three additional useful analytical solutions to 1-dimensional dispersion-advection equation in porous media, and more solutions to 2 and 3-dimensional situations [23]. Fortran is still used today for high performance computing, but it is difficult to implement for analytical

DOI: https://doi.org/10.22369/issn.2153-4136/14/1/2

solutions. High-level languages like Python provide many external libraries for specific needs such as root finding, minimization, and graphics that make it a more suitable alternative on modern computing platforms for problems such as the transport of a tracer in a porous medium.

In this article, a new Python library "columntracer" and a suite of supplementary Jupyter Notebooks [11] illustrating its development and usage are presented. The software is written entirely in Python and is freely available online. Key features of columntracer include the ability to: (1) calculate the solute concentration at any point in time and space in a column, (2) plot concentration profiles and breakthrough curves, and (3) fit experimental data at the outlet to breakthrough curves to find dispersion coefficient. In experimental column studies, effluent concentrations are easily obtained, while the dispersion coefficients are a key unknown parameter. By using columntracer, dispersion coefficients can be determined with a few lines of code. This library and the associated Jupyter Notebooks serve as a potentially useful educational tool for students in environmental modeling classes. Students are able to learn how contaminants flow through the column, how different initial conditions affect the concentration profile across the column and lead to different final concentrations at outlet, and how to fit experimental data to obtain dispersion coefficient of the process. The Notebooks also demonstrate the potential power of using Python versus computing tools that are more familiar to environmental science and engineering students, such as spreadsheet programs.

This project was conducted as an individual special problem for three credit hours for the student lead author. The objectives of the project were to: (1) deepen student understanding of the modeling of fate and transport of contaminants in porous media, (2) improve student Python programming skills, including creating Python classes, utilizing modules, managing code on GitHub, and publishing the columntracer library, and (3) provide an alternative to the U.S. Geological Survey Fortran based program for solving the dispersion-advection equation by developing a Python implementation.

2 METHOD

2.1 Model Description

Consider a cylindrical column of length L with flow entering on one end and exiting on the other end. The velocity of the flow U is easily measured by monitoring the flow rate (volume that exits per unit time). The dispersion coefficient D represents the tendency of the concentration gradients to dissipate. Tracer experiments using conservative substances such as bromide are typically used, along with a model, to estimate this parameter. The solute concentration in the influent for a tracer has a constant concentration of C0, and eventually the concentration leaving the column will also have a concentration of C0, at which point the tracer is said to have achieved "breakthrough." Before breakthrough, the concentration in the effluent gradually increases from zero and to the influent concentration level. Figure 1 illustrates the model system for the case when $C_0 = 100$, $L = 30$, $U = 10$, and $D = 100$. The parameters that affect the output from the column for this model are listed in .

.

Table 1. List of model parameters.

Parameters	Description	Units
C_0	Solute influent concentration	mg/L
U	Flow velocity in column	cm/hr
D	Dispersion coefficient	cm^2/hr
L	Length of column	cm

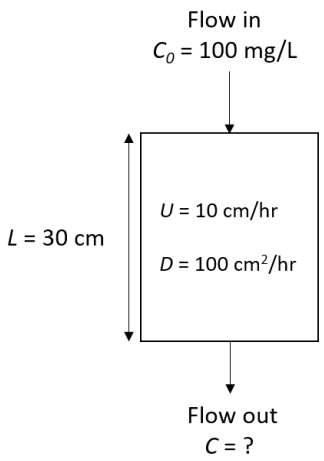

Figure 1. Schematic of Column Mode.

2.2 Advection and Dispersion Equation

2.2.1 Model and Boundary Conditions

Equation (2.1) shows the model used in the software, which is the one-dimensional advection-dispersion equation. The value of C is the concentration at time t and distance x from the inlet. The equation is based on a material balance within a differential element, and it assumes constant value of the parameters U and D.

$$\frac{dC}{dt} = D\frac{d^2C}{dx^2} - U\frac{dC}{dx} \tag{2.1}$$

Two boundary conditions and one initial condition are required to solve the equation. Assuming there is no tracer in the column at the start of the simulation, the initial condition is zero concentration, as shown in Equation (2.2). The boundary condition for the influent is flux-matching (i.e., the mass flow of the tracer into the column equals the mass flow inside the column at $x = 0$). The advective flux into the column matches the advective and dispersive fluxes at the start of the column in Equation (2.3). The Danckwerts' boundary condition used at the effluent assumes that the dispersion flux is negligible, meaning the derivative is zero, which is shown in Equation (2.4) [25].

$$C(x, t = 0) = 0 \tag{2.2}$$

$$UC_0 = UC(x = 0, t) - D\frac{dC(x = 0, t)}{dx} \tag{2.3}$$

$$\frac{dC(x = L, t)}{dx} = 0 \tag{2.4}$$

The equations can be non-dimensionalized using the dimensionless time t^*, distance x^*, and concentration C^*, which simplifies the mathematics as shown in Equations (2.5)-(2.7). In the dimensionless domain, the three parameters are reduced to just one parameter, defined as the Peclet number Pe in Equation (2.8), which represents the ratio of the importance of advection to dispersion processes in the column.

$$t^* = \frac{Dt}{L^2} \tag{2.5}$$

$$x^* = \frac{x}{L} \tag{2.6}$$

$$C^* = \frac{C}{C_0} \tag{2.7}$$

$$Pe = \frac{UL}{D} \tag{2.8}$$

The governing equation, initial condition, and boundary conditions become Equations (2.9)-(2.12), after normalization.

$$\frac{dC^*}{dt^*} = \frac{d^2C^*}{dx^{*2}} - Pe\frac{dC^*}{dx^*} \tag{2.9}$$

$$C^*(x^*,\ t^* = 0) = 0 \tag{2.10}$$

$$C^*(x^* = 0,\ t^*) - \frac{1}{Pe}\frac{dC^*(x^* = 0,\ t^*)}{dx^*} = 1 \tag{2.11}$$

$$\frac{dC^*(x^* = 1, t^*)}{dx^*} = 0 \tag{2.12}$$

2.2.2 Analytical Solution to the Model

The dimensionless governing equation and auxiliary conditions (Equations (2.9)-(2.12)) are a boundary value problem that can be solved using separation of variables [18]. The USGS summarizes four analytical solutions for the 1-dimensional advection-dispersion equation, including the model problem shown in Equation (2.13, which is referred to as a "Finite System with Third-Type Source Boundary Condition" in the report. Only one analytical solution is included in columntracer, but the others could be added easily in the future or developed for other student projects. Furthermore, analytical solutions to 2 and 3-dimensional problems in different situations are also available [23]. The values of β_i are the eigenvalues of the boundary value problem, and the corresponding terms in the infinite series are the eigenfunctions [5]. The eigenvalues are determined by finding the roots of Equation (2.14), which is the characteristic equation of the boundary value problem.

$$C^*(x^*,\ t^*) = 1 - 2Pe \cdot e^{\left(\frac{Pe}{2}x^* - \frac{Pe^2}{4}t^*\right)} \cdot \sum_{n=1}^{\infty} \frac{\beta_i\left[\beta_i \cos(\beta_i x^*) + \frac{Pe}{2}\sin(\beta_i x^*)\right]}{\left[{\beta_i}^2 + \frac{Pe^2}{4} + Pe\right]\left[{\beta_i}^2 + \frac{Pe^2}{4}\right]} \cdot e^{-\beta_i^2 t^*} \tag{2.13}$$

$$\beta cot\beta - \frac{\beta^2}{Pe} + \frac{Pe}{4} = 0 \tag{2.14}$$

A sufficient number of eigenvalues must be estimated to perform the summation in Equation (2.13). The characteristic equation (2.14) has no exact solution, unlike some other characteristic equations commonly encountered in diffusion boundary value problems. The eigenvalues for a given column system with parameters U, D, and C_0 depend only on the Peclet number defined in Equation (2.8). The values for a given Pe can be determined by finding the roots of Equation (2.15). The function has a singularity at all integral multiples of π based on trigonometric relationships shown in Equation (2.16) and (2.17).

$$F(Pe, \beta) = \beta cot\beta - \frac{\beta^2}{Pe} + \frac{Pe}{4} \tag{2.15}$$

$$cot\beta = \frac{cos\beta}{sin\beta} \tag{2.16}$$

$$sin\beta = 0\ at\ \beta = 0, \pi, 2\pi, \ldots = n\pi \tag{2.17}$$

Between each singularity, the function has exactly one zero. Figure 2 shows the value of the function across the first ten singularities. It also shows the first few roots. In Figure 2, the value of the function, the singularities at every $n\pi$, and the location of the first eigenvalue near $\beta = 1.54$ can be seen. To use the model result, the first task is to identify the eigenvalue across each interval. Scientific Python (SciPy) has an optimization library with a variety of methods to determine the root of a function. For the model problem, Brent's method [3] can be used to solve the characteristic equation.

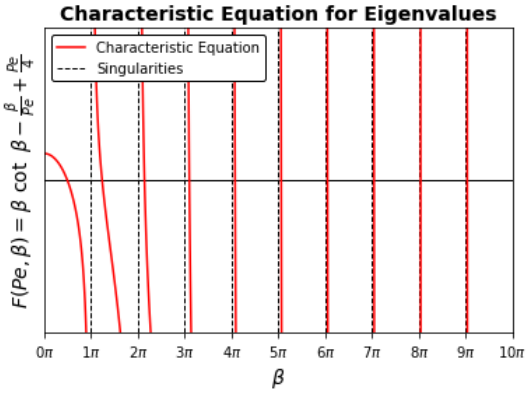

Figure 2. Characteristic Equation for Eigenvalues.

Brent's method (also sometimes called the van Wijngaarden-Dekker-Brent method) is a root-finding algorithm which combines root bracketing, bisection, and inverse quadratic interpolation. It uses a Lagrange interpolating polynomial of degree 2. Brent [3] claims that this method always converges as long as the values of the function are computable within a given region containing a root. Given three points x_1, x_2, and x_3, Brent's method fits x as a quadratic function of y, then uses the interpolation formula described in Equation (2.18) [21].

$$x = \frac{[y - f(x_1)][y - f(x_2)]x_3}{[f(x_3) - f(x_1)][f(x_3) - f(x_2)]} + \frac{[y - f(x_2)][y - f(x_3)]x_1}{[f(x_1) - f(x_2)][f(x_1) - f(x_3)]} + \frac{[y - f(x_3)][y - f(x_1)]x_2}{[f(x_2) - f(x_3)][f(x_2) - f(x_1)]} \tag{2.18}$$

Subsequent root estimation is obtained by setting y = 0, giving

$$x = x_2 + \frac{P}{Q} \tag{2.19}$$

where P and Q are:

$$P = S[T(R - T)(x_3 - x_2) - (1 - R)(x_2 - x_1)] \tag{2.20}$$

$$Q = (T - 1)(R - 1)(S - 1) \tag{2.21}$$

with

$$R \equiv \frac{f(x_2)}{f(x_3)} \tag{2.22}$$

$$S \equiv \frac{f(x_2)}{f(x_1)} \tag{2.23}$$

$$T \equiv \frac{f(x_1)}{f(x_3)} \tag{2.24}$$

Following the determination of a suitable number of eigenvalues, the simulated concentration is computed at any point in the domain by summing the eigenvalues in Equation (2.13).

2.2.3 Dispersion Coefficient Fitting

One of the primary applications of Equation (2.13) is to determine the value of the dispersion coefficient in the media. The velocity and initial concentration can be measured relatively easily for a

given experiment. Determining the value of D requires an inverse parameter fitting, which typically requires running many simulations, assessing performance, and optimizing an objective function, such as minimizing error. A common approach for parameter estimation is to compare the model simulation with experimental results. Given a set of values for the effluent concentration at $x = L$ at various points in time, a series of values of the dispersion coefficient can be used to calculate the concentrations corresponding to specific data points of breakthrough curve.

After the simulated concentrations corresponding to each data point are calculated, the mean squared error (MSE) between the simulated data and the experimental data can be determined. The MSE is calculated using Equation (2.25), where $\widehat{C}_i$ is the simulated concentration, C_i is the measured concentration, n is the number of measurements. The goal of fitting process is to minimize the MSE.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\widehat{C}_i - C_i)^2 \tag{2.25}$$

Scientific Python (SciPy) offers several functions for minimization in its "optimize" module. Four different functions are available, depending on the nature of the application. The default function is fmin, which uses the downhill simplex algorithm, also known as the Nelder-Mead method [14]. The other options are fmin_powell, fmin_cg, and fmin_bfgs, which use Powell's method [17], the nonlinear conjugate gradient algorithm [15], and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [6], resp. The results of different functions were compared for performance as described in Section 3.3. One of the advantages of using Python is the ready availability of these tools for applications such as parameter fitting. Demonstrating these capabilities to students can help to close the gap and develop computational thinking skills for students with limited programming experience.

After the dispersion coefficient is determined, the coefficient of determination, R^2, can be calculated using Equation (2.26), where $\bar{C}_s$ and $\bar{C}_m$ represent the mean values of the model, $\widehat{C}_i$, and the observations, C_i, resp.

$$R^2 = \frac{(\sum_{i=1}^{i=n}(\widehat{C}_i - \bar{C}_s)\cdot(C_i - \bar{C}_m))^2}{\sum_{i=1}^{i=n}(\widehat{C}_i - \bar{C}_s)^2 \cdot \sum_{1}^{n}(C_i - \bar{C}_m)^2} \tag{2.26}$$

2.3 Software

2.3.1 Description

The model tools described in Section 2.2 have been compiled into a software library known as "columntracer." The columntracer library is written completely in Python [19]. Python is an interpreted high-level open-source programming language, with a design philosophy that emphasizes code readability. Python's user-friendly syntax and interpreted nature decrease the time requirements for new users (e.g., students in an environmental modeling course) to begin applying the software to problem solving. Python's extensibility and interpreted nature allow new users to perform complex tasks by integrating various libraries, thereby saving time [12]. The key third-party modules used in columntracer are Numeric Python (NumPy) for generating and calculating arrays and matrixes [7], Scientific Python (SciPy) for optimizing and solving equations [20], and the Math Plotting Library (Matplotlib) for plotting and visualization [9].

The Jupyter Notebook is an open-source web application that allows user to create and share documents that contain live code, equations, visualizations, and narrative texts [11]. Jupyter Notebooks have been generated to illustrate the computation procedure outlined in Section 2, show applications of the columntracer classes that enable rapid development of new models, and provide documentation of the source code that is available on GitHub [4]. In the documentation, examples are demonstrated with both code and narrative texts. An example data set for fitting the dispersion coefficient is also available in the repository that comes with the module. The data set was taken from a study that compared the performance of different models that were fit to experimental concentrations in a one-dimensional column [24].

The Binder project offers an easy place to share computing environment to everyone. It allows project creators to specify custom environments and share them with a single link [10]. With the link, users are able to get access to the project without downloading any required software or packages. On the columntracer GitHub page [4], Binder links are provided for interacting the Jupyter Notebooks, requiring only a web browser. With the help of Binder, it's easy to demonstrate the functions of columntracer in classroom or other educational situations that Python environment is not immediately accessible.

2.3.2 Walk-through

Figure 3 shows a screenshot of the code for importing the columntracer package and performing a demo run with the default parameters.

```
from columntracer import ColumnTracer
```

```
%matplotlib inline

c = ColumnTracer(demo = True,
                 demo_plot = True,
                 demo_plot_save = False)
```

```
Default parameters for the demo are:
solute influent concentration C0 = 100 mg/L,
flow velocity in column U = 10 cm/h,
dispersion coeffiecient D = 100 cm2/h,
length of column L = 30 cm,
number of terms to use in series solution n = 1000.
```

Figure 3. Example Code for Import and Demo Run

Four methods are called during the demo run in the following sequence: (1) "characteristic_equation" that computes and plots the characteristic equation for a given *Pe* (3 in the demo), (2) "eigenvalues" that calculates the first *n* eigenvalues (1000 in the demo), (3) "concentration_profile" that calculates the concentrations across the column at various times (0.00001, 0.1, 0.5, 1, 2, 4, and 10 hours in the demo), and (4) "effluent_concentration" that calculates the concentration at the outlet of the column (0 to 12 hours in the demo).

By setting the parameter "demo_plot" to True, the software generates plots of the characteristic equation for eigenvalues (Figure 2), the column concentration profiles (Figure 11), and the column breakthrough curve (Figure 12), which can also be obtained by the code in Figure 4, Figure 6, and, Figure 8, respectively. The parameter "demo_plot_save" determines whether to save the plots to a local file, and "savefig_dpi" specifies the image quality (200 dots per inch, dpi).

The parameter "savefig" in Figure 4, Figure 6, and, Figure 8 controls the export of plots, and it is set to False by default. Setting savefig to True will save the plot to the working directory with a default file names of "characteristic_equation," "concentration_profile," and "breakthrough_curve," respectively. Users can also assign a string to the parameter to name the image files. If users

want an image with lower or higher quality, they can change the value of parameter "savefig_dpi."

```
%matplotlib inline

c = ColumnTracer(C0 = 100,
                 U = 10,
                 D = 100,
                 L = 30,
                 n = 1000)

c.characteristic_equation(plot = True,
                          savefig = False,
                          savefig_dpi = 200)
```

Figure 4. Example Code to Compute the Characteristic Equation.

```
c = ColumnTracer(C0 = 100,
                 U = 10,
                 D = 100,
                 L = 30,
                 n = 1000)

eff = c.get_concentration(time = 9,
                          x = 1)
print('The concentration is {:.2f} mg/L'.format(eff))
```

```
The concentration is 98.23 mg/L
```

Figure 5. Example Code for Calculating the Concentration at a Given Time and Position.

Columntracer can calculate the concentration at any given time and location in the column as shown in Figure 5. The *x* values range from 0 to 1, which is similar to the parameter "position" in Figure 6, indicating the location from the beginning to the end of the column. In Figure 5, the effluent concentration at 9 hours is calculated to be 98.23 mg/L. By calculating the concentration throughout the column at a given time, a concentration profile can be created as shown in Figure 6.

```
%matplotlib inline

c = ColumnTracer(C0 = 100,
                 U = 10,
                 D = 100,
                 L = 30,
                 n = 1000)

t = [0.00001, 0.1, 0.5, 1, 2, 4, 10]
pos = [0, 0.01, 0.02, 0.04, 0.06,
       0.08, 0.1, 0.15, 0.2, 0.25,
       0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
       0.85, 0.9, 0.95, 0.98, 0.99, 1]
c_profile = c.concentration_profile(times = t,
                                    positions = pos,
                                    plot = True,
                                    print_conc = False,
                                    savefig = False,
                                    savefig_dpi = 200)
```

Figure 6. Example Code for Calculating and Plotting Concentration Profiles.

In Figure 6, concentration profiles are calculated for t = 0.00001, 0.1, 0.5, 1, 2, 4, and 10 hours at different locations through the column, which are indicated by the variable "pos." The parameter "positions" must be provided as a list of values ranging from 0 to 1, and each value represents the ratio of the distance in the column to the total length of the column. This method returns a list of concentration lists that can be printed by setting parameter "print_conc" to True. Each concentration list corresponds to a time in the parameter "times," and each list has the same length as parameter "positions." By using the concentration stored in variable "c_profile," users can access the data and make plots using the Matplotlib package [9] as illustrated in Figure 7.

For calculating and plotting a breakthrough curve such as the one shown in Figure 8, users must provide a time period for the solute transport, as well as the time interval, which determines how many data points are calculated. The parameter "time_start" is 0 by default, but can modified if a different starting time is desired. This method returns a list of concentrations that can be used for printing or plotting. Users can also choose to use automatic plotting by setting parameter "plot" to True, or create plots manually as shown in Figure 9.

```
import matplotlib.pyplot as plt
# plotting
fig, ax = plt.subplots()
ax.set_xlabel('Position in column (cm)',
              size = 12, weight = 'bold')
ax.set_ylabel('Concentration (mg/L)',
              size = 12, weight = 'bold')
ax.set_title('Column Concentration Profiles',
             size = 14, weight = 'bold')

for t, cs in zip(default_t, c_profile):
    ax.plot(default_pos, cs, label = 't = {:.1f}h'.format(t))
ax.legend(loc = 'right', bbox_to_anchor = (1.4, 0.5),
          fontsize = 12)
```

Figure 7. Example Code for Accessing Concentration Profile Data after Numerical Computation.

For calculating and plotting a breakthrough curve such as the one shown in Figure 8, users must provide a time period for the solute transport, as well as the time interval, which determines how many data points are calculated. The parameter "time_start" is 0 by default, but can modified if a different starting time is desired. This method returns a list of concentrations that can be used for printing or plotting. Users can also choose to use automatic plotting by setting parameter "plot" to True, or create plots manually as shown in Figure 9.

```
%matplotlib inline

c = ColumnTracer(C0 = 100,
                 U = 10,
                 D = 100,
                 L = 30,
                 n = 1000)
time_start = 0
time_end = 12
interval = 0.1
Cs = c.effluent_concentration(time_end = time_end,
                              interval = interval,
                              time_start = time_start,
                              plot = True,
                              print_conc = False,
                              savefig = False,
                              savefig_dpi = 200)
```

Figure 8. Example Code for Calculating the Effluent Concentration and Plotting the Breakthrough Curve.

In Figure 10, a csv file containing time and effluent concentration data is imported. The first 8 values in the file are also shown in the figure. The data are used to fit to a breakthrough curve, so that a dispersion coefficient can be determined. The data processing

in the figure is only for the example data set, which is a csv file with 2 columns: one for time and the other for the corresponding concentrations. The csv file is available in the columntracer module folder or can be accessed on GitHub repository [4]. The initial concentration, solute velocity, length of the column, and the initial guess of the dispersion coefficient are required for the dispersion coefficient fitting. Four algorithms are available for minimization, which are described in Section 3.3. Setting the parameter "algorithm" to None applies the default algorithm: the Nelder-Mead method.

```
import numpy as np
import matplotlib.pyplot as plt
# plotting
time = np.arange(time_start, time_end, interval)

fig, ax = plt.subplots()
ax.set_xlabel('Time (hr)',
              size = 12, weight = 'bold')
ax.set_ylabel('Concentration (mg/L)',
              size = 12, weight = 'bold')
ax.set_title('Column Breakthrough Curve',
             size = 14, weight = 'bold')
ax.plot(time, Cs, label = 'Breakthrough curve',
        ls = '-', c = 'r')
# plug flow line
xs = [0, L/U, L/U, time_end]
ys = [0, 0, C0, C0]
ax.plot(xs, ys,
        ls = '-.', lw = 1, c = 'b', label = 'Plug flow')
ax.legend()
```

Figure 9. Example Code for Accessing Effluent Concentration Data after Numerical Computation.

```
# use pandas to help read data from external csv file
import pandas as pd
import sys

# example data file is available in
# \Lib\site-packages\columntracer
# it's also available in the package repository:
# https://github.com/BYL4746/columntracer
path = sys.executable.split('python.exe')[0]
       + 'Lib\\site-packages\\columntracer\\'
data = pd.read_csv(path + 'data.csv')

# convert pandas DataFrame to lists
fit_t = data['time'].values.tolist()
fit_c = data['concentration'].values.tolist()

# fit D with known U
c = ColumnTracer(C0 = 1,
                 U = 34,
                 L = 650,
                 n = 1000)

result = c.fit_D(time = fit_t,
                 conc = fit_c,
                 algorithm = None,
                 initial_guess=175,
                 plot = True)
```

time	concentration
17.35043	0.102756522
17.5641	0.147771513
17.75641	0.239908041
18.39744	0.313172909
18.61111	0.363423502
18.84615	0.477547322
19.10256	0.625177876
19.33761	0.654484942

Figure 10. Example Code for Fitting Data to Breakthrough Curve to Fit the Dispersion Coefficient.

3 RESULTS

Examples of several model applications are provided in Jupyter Notebooks that describe the code and show plots of the output for educational purposes. A general description of the model and a detailed set of examples scripts for columntracer are provided with the source repository [4]. Jupyter Notebooks are recommended for educational applications, but other Python environments can also be used, including the Command Prompt or the IPython console. Alternative text editors and integrated development environments (IDE) such as PyCharm and Spyder can also be used to work with the code, particularly since columntracer is provided as a library on the Python Package Index (PyPI) [26].

3.1 Concentration Profiles

After the eigenvalues for a given parameter set (*Pe*) have been determined, the concentration can be evaluated at any point in time and space with the same approach described in Section 2.2.2. Figure 11 shows the evolution in the concentration profile over time throughout the column for the example case where initial concentration $C_0 = 100$ mg/L, the column length $L = 30$ cm, the solute velocity $U = 10$ cm/hr, and the dispersion coefficient $D = 100$ cm^2/hr. The number of eigenvalues used for the example was $n = 1000$. At the beginning of the simulation, the concentration is zero everywhere, as expected, while at the end, the concentration has equilibrated with the influent concentration throughout the column. Between these extremes, the concentration gradually increases throughout the column.

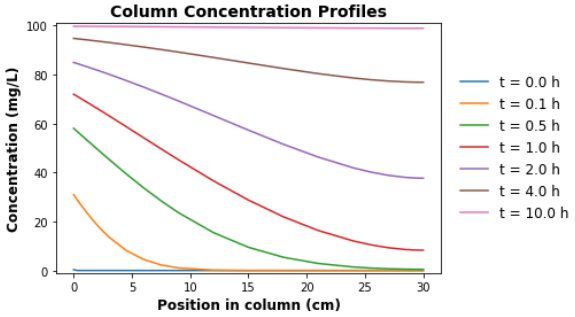

Figure 11. Column Concentration Profiles for t = 0, 0.1, 0.5, 1.0, 2.0, 4.0, 10 hours.

3.2 Breakthrough Curve

The concentration at the outlet is of primary interest for tracer studies, since it can be compared to observed data. A high-resolution time series of concentrations can easily be obtained by evaluating the function at the outlet ($x = L$), and the breakthrough curve is shown in Figure 12. The dotted line in blue indicates the breakthrough with $D = 0$, which is known as "plug flow," since the fluid flow paths in this case are all the same causing flow in a "plug" motion.

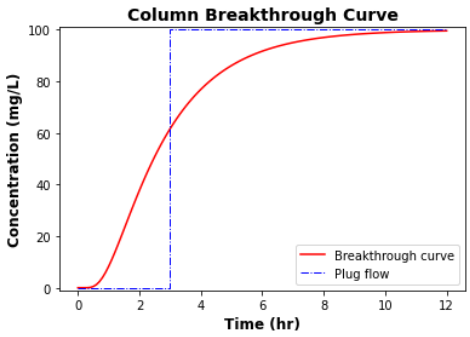

Figure 12. Column Breakthrough Curve.

3.3 Dispersion Model Fitting

Experimental data for a column were obtained from Xiong et al. [24] and are provided with the columntracer repository on GitHub

[4]. The column length in the study was 650 cm, and the velocity was 34 cm/hr. The concentrations in this publication were non-dimensionalized, meaning they represent the dimensionless C^* (the ratio of the effluent to the influent concentration), so the values range between 0 and 1. By choosing the default minimization method from SciPy, fmin, which uses Nelder-Mead method, and with an initial guess of 175 cm^2/hr, the fitted dispersion coefficient is 193.9 cm^2/hr, with mean squared error (MSE) of 0.0888 and R^2 of 0.964. Figure 13 shows the raw data and the breakthrough curve based on fitted dispersion coefficient. In Xiong et al. [25], the coefficient was fitted to be 74 cm^2/hr with root mean square error (RMSE) of 0.0313 and an R^2 of 0.9935. The relatively higher MSE may be caused by the inaccuracy and low quantity of the experimental data obtained from the figure. Because porosity η was not considered in the software, the adjusted dispersion coefficient is calculated to be 77.56 cm^2/hr using Equation (3.1), assuming $\eta = 0.4$, which is in good agreement with the value determined from the original analysis.

$$D_{adj} = \eta D \tag{3.1}$$

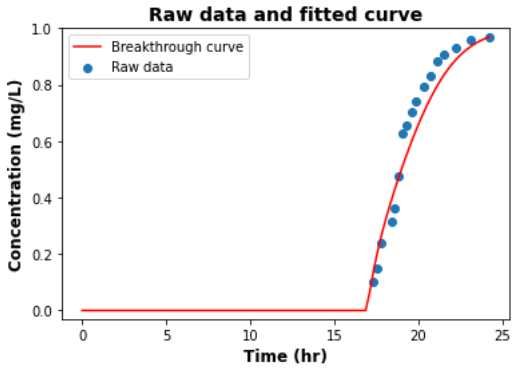

Figure 13. Fitted Breakthrough Curve and Experimental Data.

The other three minimization methods from SciPy were also tested. The results are listed in Table 2, which compare the time consumption to determine D for the different algorithms. For the sample dataset, both the Nelder-Mead method and Powell's method achieved satisfactory MSEs, whereas the MSEs for the nonlinear conjugate gradient algorithm and BFGS algorithm were less satisfactory. Students can easily try different algorithms to find the most suitable approach for their own data sets in an educational environment.

Table 2. Fitted Dispersion Coefficient and MSE Using Different Minimization Algorithms.

Function	Algorithm	*D* (cm^2/hr)	MSE (-)	Time (s)
fmin	Nelder-Mead method	193.9	0.089	4.73
fmin_powell	Powell's method	171.2	0.088	19.1
fmin_cg	Nonlinear conjugate gradient algorithm	175	0.18	7.7
fmin_bfgs	BFGS algorithm	175	0.19	9.6

3.4 Verification, Validation, and Accreditation

The USGS report provides an example (Sample Problem 2) in section titled "Finite System with Third-Type Source Boundary Condition" that includes detailed computational results shown in attachment 4 [23]. The input data from this sample problem were used for validation of the columntracer library. The input parameters include: the initial concentration $C_0 = 1$ mg/L, the column length $L = 12$ inches (30.48 cm), the solute velocity $U = 0.6$ inch/hr (1.524 cm/hr), and the dispersion coefficient $D = 0.6$ $inch^2$/hr (3.87096 cm^2/hr). The concentration profiles are shown in Figure 14, in which the lines represent the simulated data by columntracer and markers represent the data provided in the USGS report. The detailed results are presented in U.S. Customary units in APPENDIX in the Appendix and show perfect consistency with the results in the report.

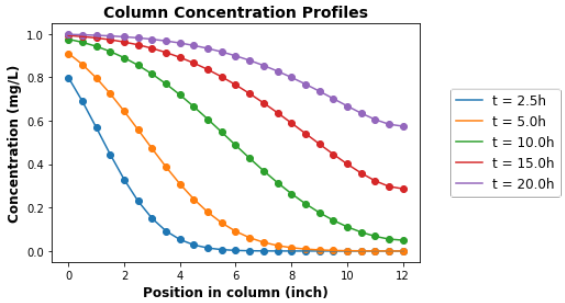

Figure 14. Concentration Profiles for Sample Problem 2.

The simulation results in Table 3 and the concentration profile plot are provided in the USGS report on pages 219 and 31, respectively. Both Figure 14 and Table 3 show a perfect reproduction of the results from the USGS program using the columntracer simulation. The time consumption is about 0.03 seconds, which is over a hundred times faster than the FORTRAN program described in the report. A Jupyter Notebook is available showing these results in a folder called "validation" on GitHub repository for validation [4]. The results provide a comparison of the utility of the modern interactive Python notebook and a compiled and less user-friendly FORTRAN program from 30 years ago.

The source code, data, and Jupyter Notebooks can be found on GitHub repository [4]. The columntracer library is downloadable from PyPI by using command "pip install columntracer" in Command Prompt. With the installation of columntracer and a clone of the repository, users are able to verify the algorithms of the program and use the Jupyter Notebooks to validate the results of the model that are provided in the repository. The source code was installed on new machines and used to validate the results shown in this article. The results are shown with both MSE and R2 to ensure that the model correctly predict the transport process. With the help of the Binder link provided in the source repository, the Jupyter Notebooks can be run through a server online without installing any software locally. The examples provided throughout this exercise, and additional examples that were run using different parameter values all validate the model. The expected behavior that is observed in physical experiments is reproduced from the model results.

The information in this article is useful for both undergraduate and graduate students in environment-related majors, teachers who teach courses involving fluid transport in porous media, and researchers who perform column experiments. The information provided can help students both be trained with Python programming ability and learn modeling of fate and transport of dissolved constituents.

4 CONCLUSION

Column experiments are useful for studying fate and transport of solutes through porous media. A new open-source software tool,

columntracer, has been developed to help user better understand the column experiment. The software provides solutions to advection-dispersion equation as well as the visualization of the solutions, which includes plotting the characteristic equation, concentration profiles, and the effluent breakthrough curve. The software can also be used to fit the dispersion coefficient using experimental effluent data by minimizing the mean-squared error. The columntracer library can be a useful tool for research, but it is also appropriate for educational purposes. Students in environmental modeling courses could use the software to learn about solute transport, Python scripting, NumPy, SciPy, Matplotlib, and Jupyter Notebook by using the software and the supplementary Notebooks. The code and Notebooks are open-source and freely available online [4].

5 REFLECTION

By completing the project, I managed to learn how to define and modify a Python class by manipulating attributes and functions, as well as by implementing third-party libraries including NumPy, SciPy, Matplotlib, Jupyter, and Binder. In addition to programming skills, I became acquainted with the advection-dispersion model, and its analytical solution solved by separation of variables. I also learned how to perform parameter fitting using optimization within the Python environment. During the acquirement of these skills, there were several challenges that I faced. Debugging was one of the hardest, because sometimes a typo could result in a break-down or an unexpected outcome. Finding an appropriate method for the parameter fitting was also challenging, because there were numerous approaches available. On the whole, the project improved both my programming ability and specialized knowledge in my major, and would help me in future projects such as programming for wastewater process optimization and machine learning in Python. For these reasons, I consider the project an overall success.

REFERENCES

[1] Munshoor Ahmed, Qurat Ul Ain Zainab, and Shamsul Qamar. 2017. Analysis of One-Dimensional Advection–Diffusion Model with Variable Coefficients Describing Solute Transport in a Porous medium. *Transp Porous Med 118*, 3 (July 2017), 327–344. DOI:https://doi.org/10.1007/s11242-017-0833-0

[2] Stefan Banzhaf and Klaus Hebig. 2016. Use of column experiments to investigate the fate of organic micropollutants - A review. *Hydrology and Earth System Sciences 20*, (September 2016), 3719–3737. DOI:https://doi.org/10.5194/hess-20-3719-2016

[3] Richard P. Brent. 2013. *Algorithms for Minimization Without Derivatives*. Courier Corporation.

[4] BYL4746. 2021. BYL4746/columntracer. Retrieved July 7, 2021 from https://github.com/BYL4746/columntracer

[5] R. Courant and D. Hilbert. 1989. *Methods of Mathematical Physics* (1st ed.). Wiley, New York, New York. DOI:https://doi.org/10.1002/9783527617210

[6] Roger Fletcher. 1987. *Practical Methods of Optimization*. Wiley, New York, New York. Retrieved June 29, 2021 from http://archive.org/details/ practicalmethods0000flet

[7] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature 585*, 7825 (September 2020), 357–362. DOI:https://doi.org/10.1038/s41586-020-2649-2

[8] Åse Høisæter, Anja Pfaff, and Gijs D. Breedveld. 2019. Leaching and transport of PFAS from aqueous film-forming foam (AFFF) in the unsaturated soil at a firefighting training facility under cold climatic conditions. *Journal of Contaminant Hydrology 222*, (April 2019), 112–122. DOI: https://doi.org/10.1016/j.jconhyd.2019.02.010

[9] John D. Hunter. 2007. Matplotlib: A 2D Graphics Environment. *Computing in Science Engineering 9*, 3 (May 2007), 90–95. DOI: https://doi.org/10.1109/MCSE.2007.55

[10] Project Jupyter, Matthias Bussonnier, Jessica Forde, Jeremy Freeman, Brian Granger, Tim Head, Chris Holdgraf, Kyle Kelley, Gladys Nalvarte, Andrew Osheroff, M. Pacer, Yuvi Panda, Fernando Perez, Benjamin Ragan-Kelley, and Carol Willing. 2018. Binder 2.0 - Reproducible, interactive, sharable environments for science at scale. In *Proceedings of the 17th Python in Science Conference (2018)*, July 9-15, 2018, Austin, Texas, 113–120. DOI:https://doi.org/10.25080/Majora-4af1f417-011

[11] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Prez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damian Avila, Safia Abdalla, Carol Willing, and Jupyter Development Team. 2016. Jupyter Notebooks – a publishing format for reproducible computational workflows. *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (2016), 87–90. DOI:https://doi.org/10.3233/978-1-61499-649-1-87

[12] David Lampert. 2020. An introduction to Python programming for environmental professionals. *EM: Air and Waste Management Association's Magazine for Environmental Managers 2020*, (February 2020), 11–14.

[13] Erica R. McKenzie, Robert L. Siegrist, John E. McCray, and Christopher P. Higgins. 2015. Effects of chemical oxidants on perfluoroalkyl acid transport in one-dimensional porous media columns. *Environ. Sci. Technol. 49*, 3 (February 2015), 1681–1689. DOI:https://doi.org/10.1021/es503676p

[14] J. A. Nelder and R. Mead. 1965. A simplex method for function minimization. *The Computer Journal 7*, 4 (January 1965), 308–313. DOI:https://doi.org/10.1093/comjnl/7.4.308

[15] Jorge Nocedal and Stephen J. Wright. 2006. *Numerical Optimization* (2nd ed ed.). Springer, New York.

[16] N. Perujo, X. Sanchez-Vila, L. Proia, and A.M. Romaní. 2017. Interaction between physical heterogeneity and microbial processes in subsurface sediments: A laboratory-scale column experiment. *Environ. Sci. Technol. 51*, 11 (June 2017), 6110–6119. DOI:https://doi.org/10.1021/acs.est.6b06506

[17] M. J. D. Powell. 1964. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal 7,* 2 (January 1964), 155–162. DOI:https://doi.org/10.1093/comjnl/7.2.155

[18] Bernd S. W. Schröder. 2007. *Mathematical Analysis: A Concise Introduction.* John Wiley & Sons, Inc., Hoboken, NJ. DOI:https://doi.org/10.1002/9780470226773

[19] Guido Van Rossum. 2007. Python programming language. The Guru is in session, at *USENIX Annual Technical Conference*, Santa Clara, CA.

[20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, and Paul van Mulbregt. 2020. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nat Methods 17,* 3 (March 2020), 261–272. DOI:https://doi.org/10.1038/s41592-019-0686-2

[21] Eric W. Weisstein. Brent's Method. From MathWorld--A Wolfram Web Resource. Retrieved June 14, 2021 from https://mathworld.wolfram.com/BrentsMethod.html

[22] Paul Westerhoff, David Highfield, Mohammad Badruzzaman, and Yeomin Yoon. 2005. Rapid small-scale column tests for arsenate removal in iron oxide packed bed columns. *J. Environ. Eng. 131*, 2 (February 2005), 262–271. DOI:https://doi.org/10.1061/(ASCE)0733-9372(2005)131:2(262)

[23] Eliezer J. Wexler. 1992. *Analytical Solutions for One-, Two-, and Three-dimensional Solute Transport in Ground-water Systems with Uniform Flow*. U.S. Government Printing Office. Retrieved from https://pubs.usgs.gov/of/1989/0056/report.pdf

[24] Yunwu Xiong, Guanhua Huang, and Quanzhong Huang. 2006. Modeling solute transport in one-dimensional homogeneous and heterogeneous soil columns with continuous time random walk. *Journal of Contaminant Hydrology 86*, 3 (August 2006), 163–175. DOI:https://doi.org/10.1016/j.jconhyd.2006.03.001

[25] 1995. Continuous flow systems. Distribution of residence times. *Chemical Engineering Science 50,* 24 (December 1995), 3857–3866. DOI:https://doi.org/10.1016/0009-2509(96)81811-2

[26] PyPI · The Python Package Index. PyPI. Retrieved July 7, 2021 from https://pypi.org/

APPENDIX

Table 3. Solute Concentration as a Function of Time for Sample Problem 2

Position in Column (inch)	Time (hr)				
	2.5	5.0	10.0	15.0	20.0
	Solute Concentration (mg/L)				
0.0	0.79858	0.90992	0.97530	0.99197	0.99716
0.5	0.68921	0.85904	0.96098	0.98727	0.99549
1.0	0.56799	0.79673	0.94230	0.98097	0.99322
1.5	0.44466	0.72419	0.91871	0.97276	0.99021
2.0	0.32919	0.64364	0.88977	0.96231	0.98629
2.5	0.22958	0.55821	0.85524	0.94926	0.98128
3.0	0.15033	0.47151	0.81509	0.93331	0.97499
3.5	0.09217	0.38726	0.76955	0.91415	0.96720
4.0	0.05280	0.30880	0.71911	0.89156	0.95771
4.5	0.02820	0.23875	0.66455	0.86537	0.94630
5.0	0.01402	0.17878	0.60686	0.83551	0.93276
5.5	0.00648	0.12953	0.54722	0.80201	0.91692
6.0	0.00278	0.09072	0.48691	0.76503	0.89862
6.5	0.00111	0.06137	0.42724	0.72482	0.87775
7.0	0.00041	0.04008	0.36949	0.68179	0.85425
7.5	0.00014	0.02525	0.31477	0.63644	0.82814
8.0	0.00004	0.01534	0.26404	0.58940	0.79952
8.5	0.00001	0.00898	0.21800	0.54138	0.76864
9.0	0.00000	0.00507	0.17710	0.49322	0.73590
9.5	0.00000	0.00275	0.14160	0.44591	0.70194
10.0	0.00000	0.00144	0.11154	0.40065	0.66775
10.5	0.00000	0.00072	0.08691	0.35904	0.63487
11.0	0.00000	0.00035	0.06782	0.32340	0.60563
11.5	0.00000	0.00018	0.05487	0.29733	0.58368
12.0	0.00000	0.00012	0.04982	0.28674	0.57463