Numerically Simulating the Buckley-Leverett Equation with TCAT Capillary Pressure

Benjamin T. Shepard

October 14, 2021

Motivation

Imagine we have a well of length L that is composed of a porous medium (usually soil) that is fully saturated with liquid (usually a mixture of water and oil).

Motivation

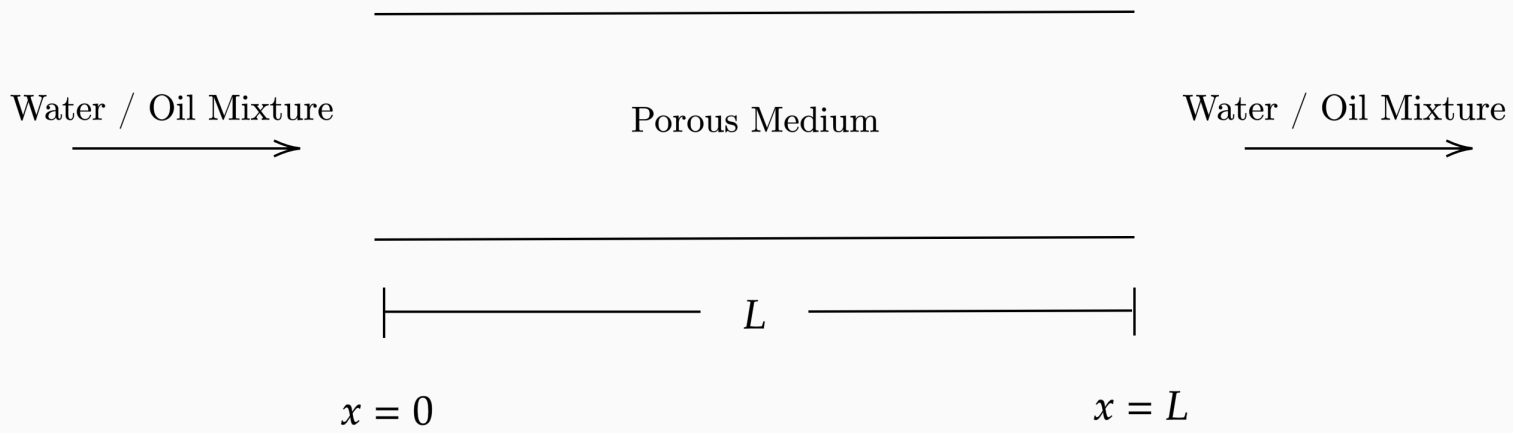
Imagine we have a well of length L that is composed of a porous medium (usually soil) that is fully saturated with liquid (usually a mixture of water and oil).

If we inject another mixture of liquid into the left end of the well, the two mixtures combine.

Motivation

Imagine we have a well of length L that is composed of a porous medium (usually soil) that is fully saturated with liquid (usually a mixture of water and oil).

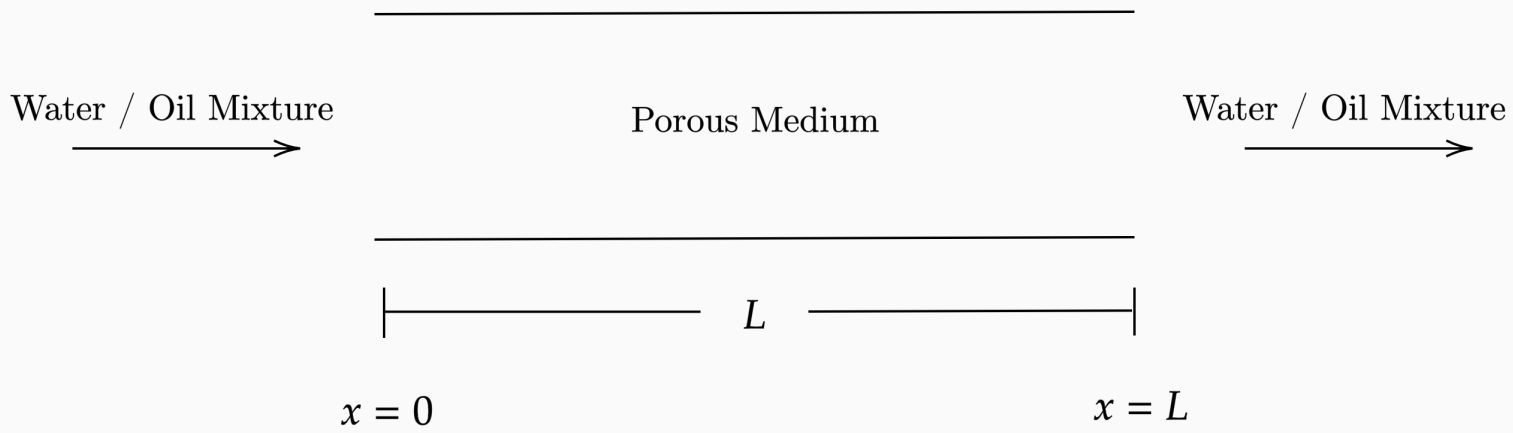
If we inject another mixture of liquid into the left end of the well, the two mixtures combine.



Motivation

Imagine we have a well of length L that is composed of a porous medium (usually soil) that is fully saturated with liquid (usually a mixture of water and oil).

If we inject another mixture of liquid into the left end of the well, the two mixtures combine.



We wish to fully understand how this interaction behaves.

The Buckley-Leverett Equation

The Buckley-Leverett Equation is a partial differential equation that models two-phase flow in porous media. The porous medium we are looking at is taken to be fully saturated, i.e. there is no air or other fluids involved in the system.

The Buckley-Leverett Equation

The Buckley-Leverett Equation is a partial differential equation that models two-phase flow in porous media. The porous medium we are looking at is taken to be fully saturated, i.e. there is no air or other fluids involved in the system.

We are working with two phases, namely water and oil. The phase with more water is called the *wetting* phase, and the phase with more oil is called the *non-wetting* phase. The wetting phase makes better contact with particles in the porous medium, and spreads more easily.

The Buckley-Leverett Equation

The Buckley-Leverett Equation is a partial differential equation that models two-phase flow in porous media. The porous medium we are looking at is taken to be fully saturated, i.e. there is no air or other fluids involved in the system.

We are working with two phases, namely water and oil. The phase with more water is called the *wetting* phase, and the phase with more oil is called the *non-wetting* phase. The wetting phase makes better contact with particles in the porous medium, and spreads more easily.

Let $u(x, t)$ denote the saturation of the wetting phase at $x \in [0, L]$ and time $t \in \mathbb{R}^+$. That is, u is the volume percentage of the wetting phase in the medium, and $1 - u$ would be the percentage of the non-wetting phase.

The Buckley-Leverett Equation

The Buckley-Leverett Equation is a partial differential equation that models two-phase flow in porous media. The porous medium we are looking at is taken to be fully saturated, i.e. there is no air or other fluids involved in the system.

We are working with two phases, namely water and oil. The phase with more water is called the *wetting* phase, and the phase with more oil is called the *non-wetting* phase. The wetting phase makes better contact with particles in the porous medium, and spreads more easily.

Let $u(x,t)$ denote the saturation of the wetting phase at $x \in [0, L]$ and time $t \in \mathbb{R}^+$. That is, u is the volume percentage of the wetting phase in the medium, and $1-u$ would be the percentage of the non-wetting phase.

Let $p_c(u) = p_n - p_w$ denote the capillary pressure of the system, where p_n and p_w are the capillary pressures of the non-wetting and wetting phases, respectively. This does not change as the fluids interact.

The Buckley-Leverett Equation

In this work, we only consider one dimension.

The Buckley-Leverett Equation

In this work, we only consider one dimension.

The one-dimensional Buckley-Leverett equation with normal capillary pressure is

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = \frac{\partial}{\partial x}\left[H(u) \frac{\partial u}{\partial x} \right]$$

The Buckley-Leverett Equation

In this work, we only consider one dimension.

The one-dimensional Buckley-Leverett equation with normal capillary pressure is

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = \frac{\partial}{\partial x}\left[H(u) \frac{\partial u}{\partial x} \right]$$

where $f(u)$ is the *fractional flow rate* and $H(u)$ is the *dissipation function*:

$$f(u) = \frac{u^2}{u^2 + 2(1-u)^2} \quad \text{and} \quad H(u) = \frac{u^2(1-u)^2}{u^2 + 2(1-u)^2}.$$

TCAT Capillary Pressure

Thermodynamically Constrained Averaging Theory (TCAT).

TCAT Capillary Pressure

Thermodynamically Constrained Averaging Theory (TCAT).

By introducing TCAT capillary pressure, we gain a dependence on the time rate of change of the saturation as well as some fluid properties, which makes the model more accurate.

TCAT Capillary Pressure

Thermodynamically Constrained Averaging Theory (TCAT).

By introducing TCAT capillary pressure, we gain a dependence on the time rate of change of the saturation as well as some fluid properties, which makes the model more accurate.

TCAT capillary pressure is given by

$$p_c\left(u, \frac{\partial u}{\partial t}\right) = p_c^e(u) - \tau_A \frac{\partial u}{\partial t} - \tau_B \frac{(\varepsilon^{wn} - \varepsilon_{eq}^{wn})}{p_c^e(u)}$$

TCAT Capillary Pressure

Thermodynamically Constrained Averaging Theory (TCAT).

By introducing TCAT capillary pressure, we gain a dependence on the time rate of change of the saturation as well as some fluid properties, which makes the model more accurate.

TCAT capillary pressure is given by

$$p_c\left(u, \frac{\partial u}{\partial t}\right) = p_c^e(u) - \tau_A \frac{\partial u}{\partial t} - \tau_B \frac{(\varepsilon^{wn} - \varepsilon_{eq}^{wn})}{p_c^e(u)}$$

where p_c^e is the equilibrium capillary pressure, τ_A and τ_B are dimensionless constants, and ε^{wn} and ε_{eq}^{wn} are the interfacial area and equilibrium interfacial area, respectively.

TCAT Capillary Pressure

Modifying the Buckley-Leverett equation to include TCAT capillary pressure yields

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = \frac{\partial}{\partial x}\left[H(u)\,\frac{\partial u}{\partial x}\right] + \tau_A\,\frac{\partial}{\partial x}\left[H(u)\,\frac{\partial u}{\partial t \partial x}\right] + \tau_B\,\frac{\partial}{\partial x}\left[\frac{H(u)(\varepsilon^{wn} - \varepsilon^{wn}_{eq})}{u^2}\,\frac{\partial u}{\partial x}\right].$$

TCAT Capillary Pressure

Modifying the Buckley-Leverett equation to include TCAT capillary pressure yields

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = \frac{\partial}{\partial x}\left[H(u)\,\frac{\partial u}{\partial x}\right] + \tau_A\,\frac{\partial}{\partial x}\left[H(u)\,\frac{\partial u}{\partial t \partial x}\right] + \tau_B\,\frac{\partial}{\partial x}\left[\frac{H(u)(\varepsilon^{wn} - \varepsilon_{eq}^{wn})}{u^2}\,\frac{\partial u}{\partial x}\right].$$

We cannot solve this nonlinear PDE analytically.

TCAT Capillary Pressure

Modifying the Buckley-Leverett equation to include TCAT capillary pressure yields

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = \frac{\partial}{\partial x}\left[H(u)\,\frac{\partial u}{\partial x}\right] + \tau_A\,\frac{\partial}{\partial x}\left[H(u)\,\frac{\partial u}{\partial t \partial x}\right] + \tau_B\,\frac{\partial}{\partial x}\left[\frac{H(u)(\varepsilon^{wn} - \varepsilon^{wn}_{eq})}{u^2}\,\frac{\partial u}{\partial x}\right].$$

We cannot solve this nonlinear PDE analytically.

Most authors that are relevant in this area of research make the PDE slightly less nonlinear by disregarding τ_B. To do this, set $\varepsilon^{wn} = \varepsilon^{wn}_{eq}$.

TCAT Capillary Pressure

Modifying the Buckley-Leverett equation to include TCAT capillary pressure yields

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = \frac{\partial}{\partial x}\left[H(u)\,\frac{\partial u}{\partial x}\right] + \tau_A\,\frac{\partial}{\partial x}\left[H(u)\,\frac{\partial u}{\partial t \partial x}\right] + \tau_B\,\frac{\partial}{\partial x}\left[\frac{H(u)(\varepsilon^{wn} - \varepsilon^{wn}_{eq})}{u^2}\,\frac{\partial u}{\partial x}\right].$$

We cannot solve this nonlinear PDE analytically.

Most authors that are relevant in this area of research make the PDE slightly less nonlinear by disregarding τ_B. To do this, set $\varepsilon^{wn} = \varepsilon^{wn}_{eq}$. Then

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = \frac{\partial}{\partial x}\left[H(u)\,\frac{\partial u}{\partial x}\right] + \tau_A\,\frac{\partial}{\partial x}\left[H(u)\,\frac{\partial u}{\partial t \partial x}\right] \qquad (*)$$

and the only parameter we need to account for is τ_A.

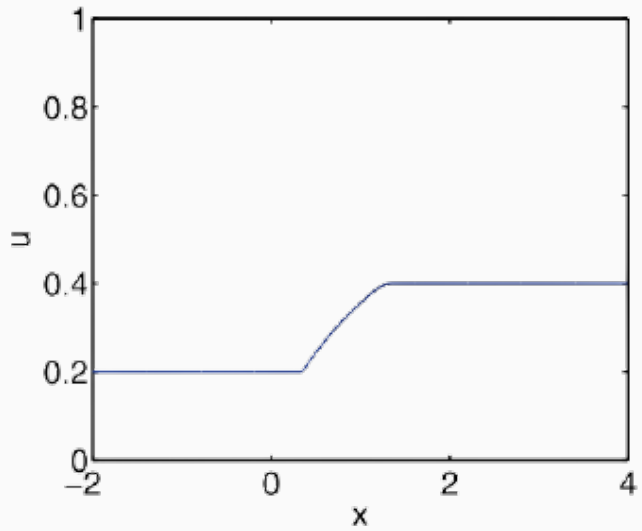
Solution Structures

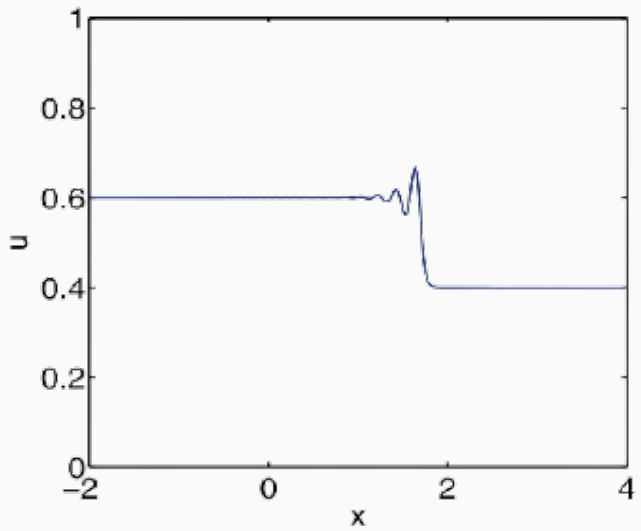
It turns out that there are only four types of solution structures, namely rarefaction, Lax shock, rarefaction shock, and double shock.

Solution Structures

It turns out that there are only four types of solution structures, namely rarefaction, Lax shock, rarefaction shock, and double shock.

Rarefaction and Lax shock are the basic building blocks for nonlinear PDE solutions.

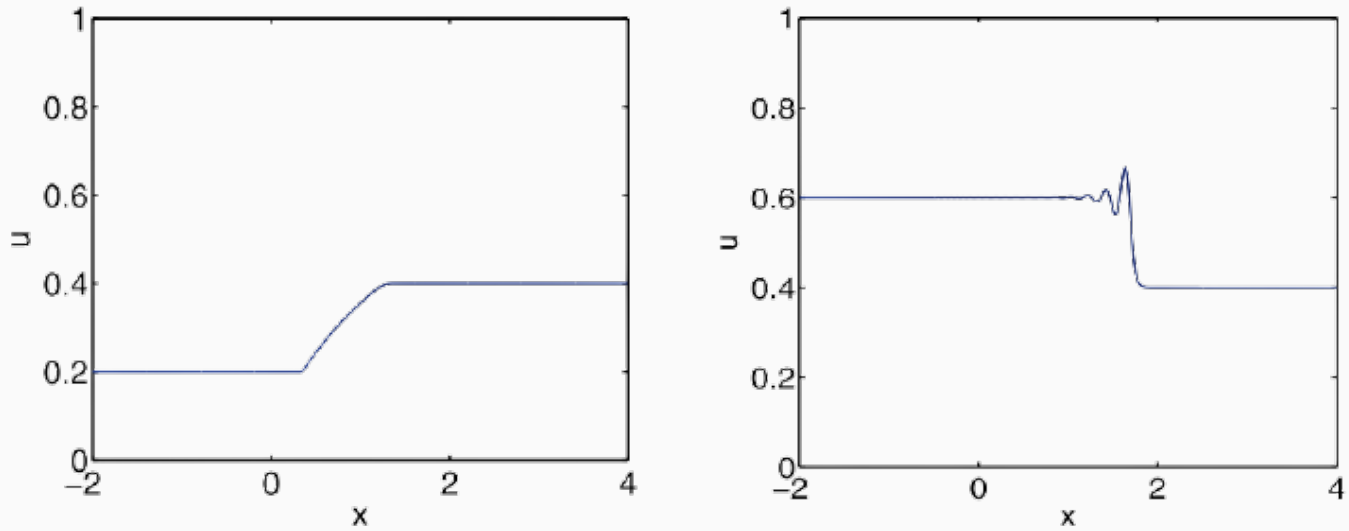




Solution Structures

It turns out that there are only four types of solution structures, namely rarefaction, Lax shock, rarefaction shock, and double shock.

Rarefaction and Lax shock are the basic building blocks for nonlinear PDE solutions.



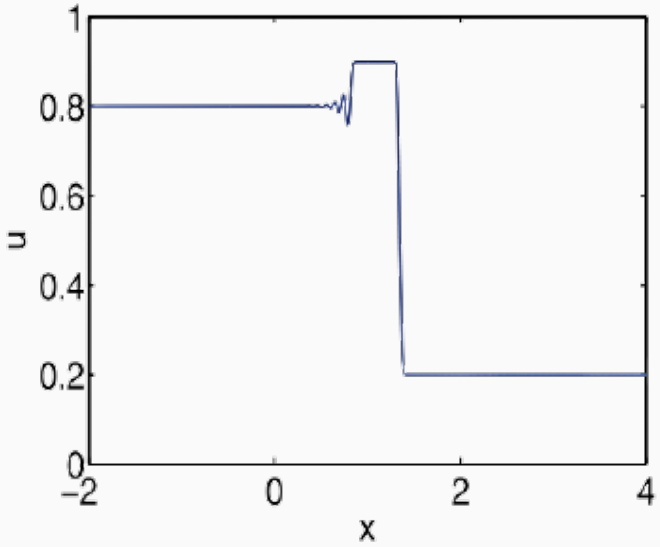
We can see some oscillation in the solution structure; this could be an artifact of the numerical analysis used in the simulation, or a separate wave structure.

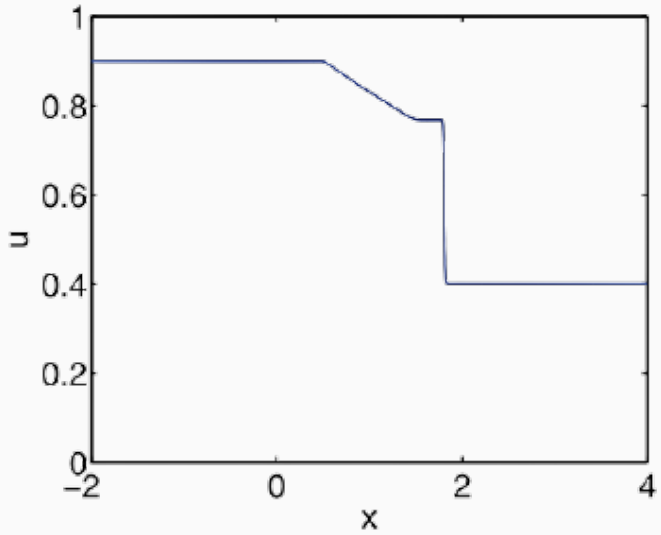
Solution Structures

The rarefaction shock and double shock are combinations of the previous two.

Solution Structures

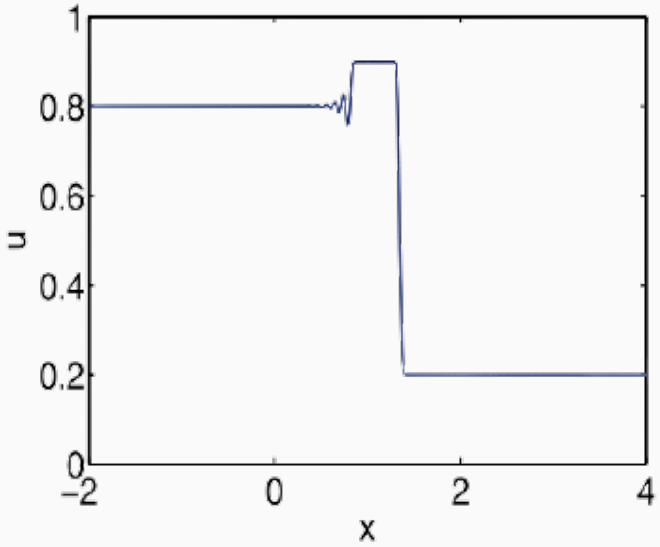
The rarefaction shock and double shock are combinations of the previous two.

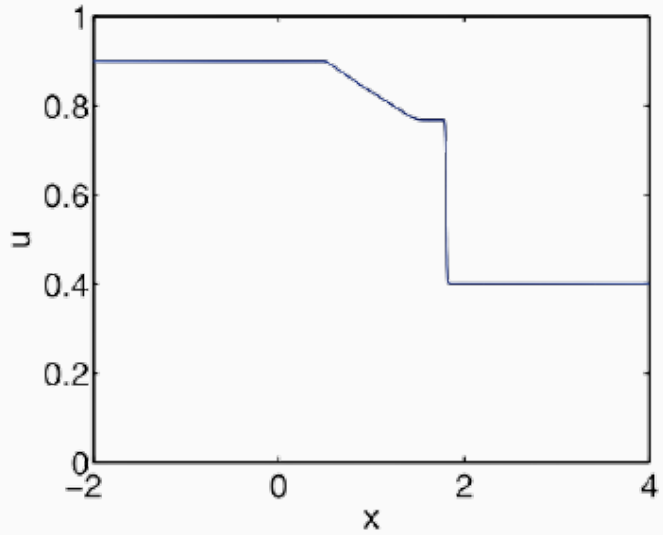




Solution Structures

The rarefaction shock and double shock are combinations of the previous two.

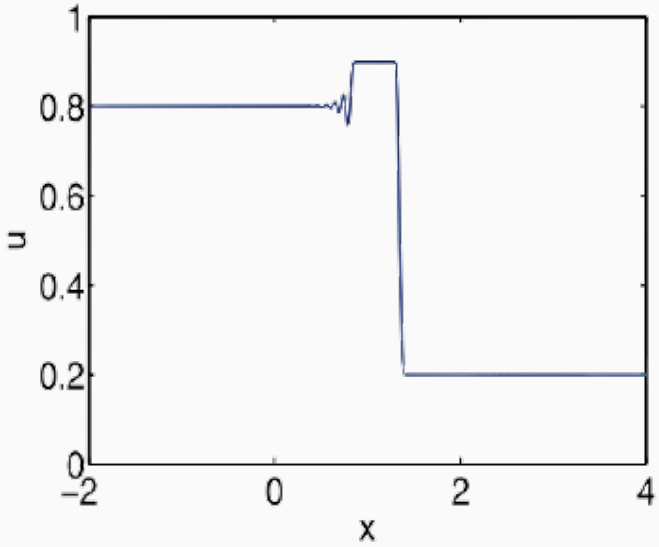


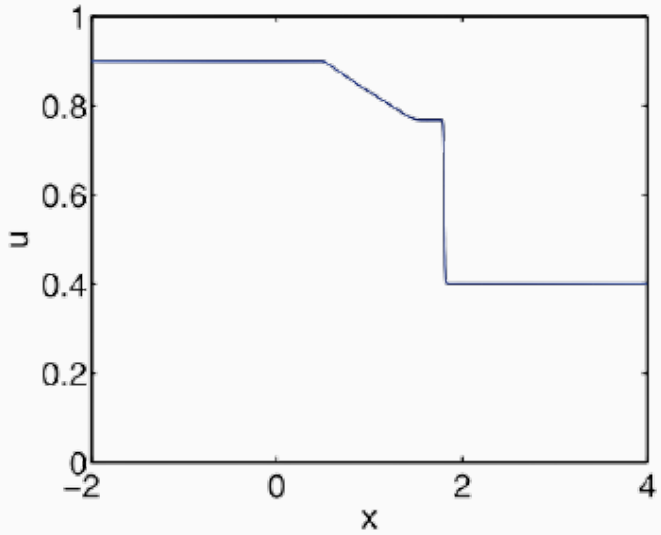


The double shock is the only solution that is non-monotonic. This is of particular interest.

Solution Structures

The rarefaction shock and double shock are combinations of the previous two.



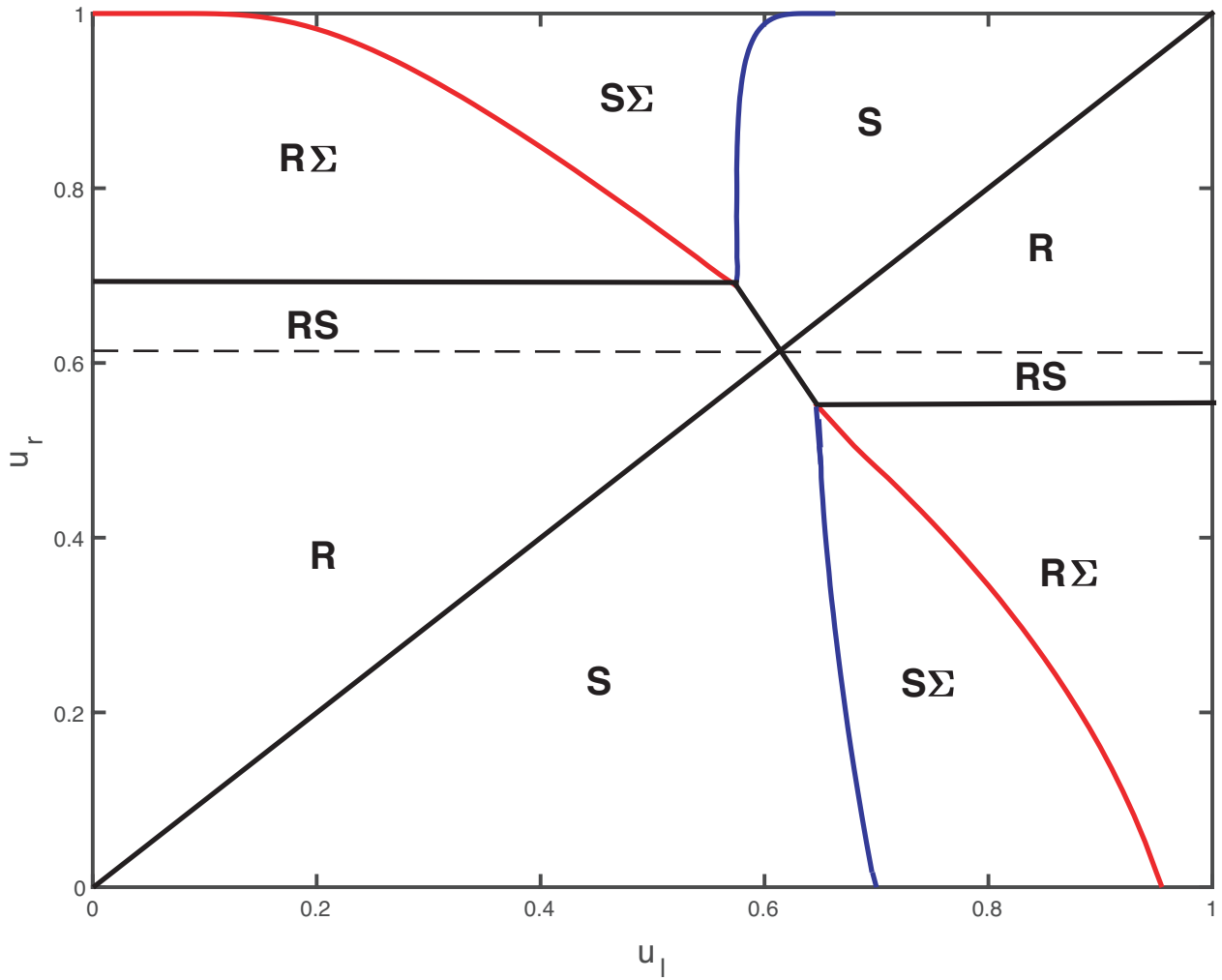


The double shock is the only solution that is non-monotonic. This is of particular interest.

The program that produced these simulations took ~ 5 hours to do so.

Solution Structures

Diagram of different solution structures:



Main Goal

The main goal is to obtain accurate values of τ_A and τ_B through model calibration.

Main Goal

The main goal is to obtain accurate values of τ_A and τ_B through model calibration.

To do this, we need to be able to simulate many, many solutions very quickly.

Main Goal

The main goal is to obtain accurate values of τ_A and τ_B through model calibration.

To do this, we need to be able to simulate many, many solutions very quickly.

Then we feed the solutions into another program that compares the simulation with real data, and determines the most accurate value for τ_A that it can.

Main Goal

The main goal is to obtain accurate values of τ_A and τ_B through model calibration.

To do this, we need to be able to simulate many, many solutions very quickly.

Then we feed the solutions into another program that compares the simulation with real data, and determines the most accurate value for τ_A that it can.

Our main focus in this research was to develop a simulation that is very fast at computing different types of solutions, yet also accurate to the previous methods.

Algorithm Implementation

The algorithm uses a finite element method, with discretization in both space and time.

Algorithm Implementation

The algorithm uses a finite element method, with discretization in both space and time.

Let

$$\alpha(u) = \frac{1}{H(u)}, \quad \beta = \frac{1}{\tau_A}, \quad g(u) = \frac{p_c(u)}{\tau_A}.$$

Algorithm Implementation

The algorithm uses a finite element method, with discretization in both space and time.

Let

$$\alpha(u) = \frac{1}{H(u)}, \quad \beta = \frac{1}{\tau_A}, \quad g(u) = \frac{p_c(u)}{\tau_A}.$$

For discretization, take $\Delta t = T/N$ and $\Delta x = L/M$, where $T, N, M \in \mathbb{N}$.

Algorithm Implementation

The algorithm uses a finite element method, with discretization in both space and time.

Let

$$\alpha(u) = \frac{1}{H(u)}, \quad \beta = \frac{1}{\tau_A}, \quad g(u) = \frac{p_c(u)}{\tau_A}.$$

For discretization, take $\Delta t = T/N$ and $\Delta x = L/M$, where $T, N, M \in \mathbb{N}$.

We use $j \in [0, M]$ for the spacial step and $n \in [0, M]$ for the time step.

Denote by u_j^n and p_j^n the approximations for u and p_c at spacial step j and time step n.

Algorithm Implementation

The algorithm uses a finite element method, with discretization in both space and time.

Let

$$\alpha(u) = \frac{1}{H(u)}, \quad \beta = \frac{1}{\tau_A}, \quad g(u) = \frac{p_c(u)}{\tau_A}.$$

For discretization, take $\Delta t = T/N$ and $\Delta x = L/M$, where $T, N, M \in \mathbb{N}$.

We use $j \in [0, M]$ for the spacial step and $n \in [0, M]$ for the time step.

Denote by u_j^n and p_j^n the approximations for u and p_c at spacial step j and time step n.

We can think of the saturation as a matrix in order to better understand this discretization:

$$\begin{bmatrix} u_0^0 & \cdots & u_j^0 & \cdots & u_M^0 \\ \vdots & & \vdots & & \vdots \\ u_0^n & \cdots & u_j^n & \cdots & u_M^n \\ \vdots & & \vdots & & \vdots \\ u_0^N & \cdots & u_j^N & \cdots & u_M^N \end{bmatrix}$$

Algorithm Implementation

We use

$$u_j^{n+1} = u_j^n + \frac{\Delta t}{\phi}\left[g(u_j^{n+1}) - \beta(u_j^{n+1})p_j^{n+1}\right]$$

to advance the simulation by one timestep.

ϕ is the rock porosity (percentage of void space in porous medium).

Algorithm Implementation

Now, we iterate over k as many times as needed to approximate u_j^n for each time step. Denote $u_j^{n,k}$ as the approximation for u_j^n at iterative step k.

Algorithm Implementation

Now, we iterate over k as many times as needed to approximate u_j^n for each time step. Denote $u_j^{n,k}$ as the approximation for u_j^n at iterative step k.

The iterative procedure is as follows (simplified):

Algorithm Implementation

Now, we iterate over k as many times as needed to approximate u_j^n for each time step. Denote $u_j^{n,k}$ as the approximation for u_j^n at iterative step k.

The iterative procedure is as follows (simplified):

1. Set initial conditions $u_j^{n,0} = u_j^n$ and $p_j^{n,0} = p_j^n$.

Algorithm Implementation

Now, we iterate over k as many times as needed to approximate u_j^n for each time step. Denote $u_j^{n,k}$ as the approximation for u_j^n at iterative step k.

The iterative procedure is as follows (simplified):

1. Set initial conditions $u_j^{n,0} = u_j^n$ and $p_j^{n,0} = p_j^n$.
2. Calculate numerical flux function $f(u)$.

Algorithm Implementation

Now, we iterate over k as many times as needed to approximate u_j^n for each time step. Denote $u_j^{n,k}$ as the approximation for u_j^n at iterative step k.

The iterative procedure is as follows (simplified):

1. Set initial conditions $u_j^{n,0} = u_j^n$ and $p_j^{n,0} = p_j^n$.
2. Calculate numerical flux function $f(u)$.
3. Calculate $\alpha(u)$, β, and $g(u)$.

Algorithm Implementation

Now, we iterate over k as many times as needed to approximate u_j^n for each time step. Denote $u_j^{n,k}$ as the approximation for u_j^n at iterative step k.

The iterative procedure is as follows (simplified):

1. Set initial conditions $u_j^{n,0} = u_j^n$ and $p_j^{n,0} = p_j^n$.
2. Calculate numerical flux function $f(u)$.
3. Calculate $\alpha(u)$, β, and $g(u)$.
4. Update $u_j^{n+1,k}$ using the equation from the previous slide.

Algorithm Implementation

Now, we iterate over k as many times as needed to approximate u_j^n for each time step. Denote $u_j^{n,k}$ as the approximation for u_j^n at iterative step k.

The iterative procedure is as follows (simplified):

1. Set initial conditions $u_j^{n,0} = u_j^n$ and $p_j^{n,0} = p_j^n$.
2. Calculate numerical flux function $f(u)$.
3. Calculate $\alpha(u)$, β, and $g(u)$.
4. Update $u_j^{n+1,k}$ using the equation from the previous slide.
5. Check for convergence. If attained, set $u_j^{n+1} = u_j^{n+1,k}$ and go to the next timestep. Otherwise, reset iteration level k.

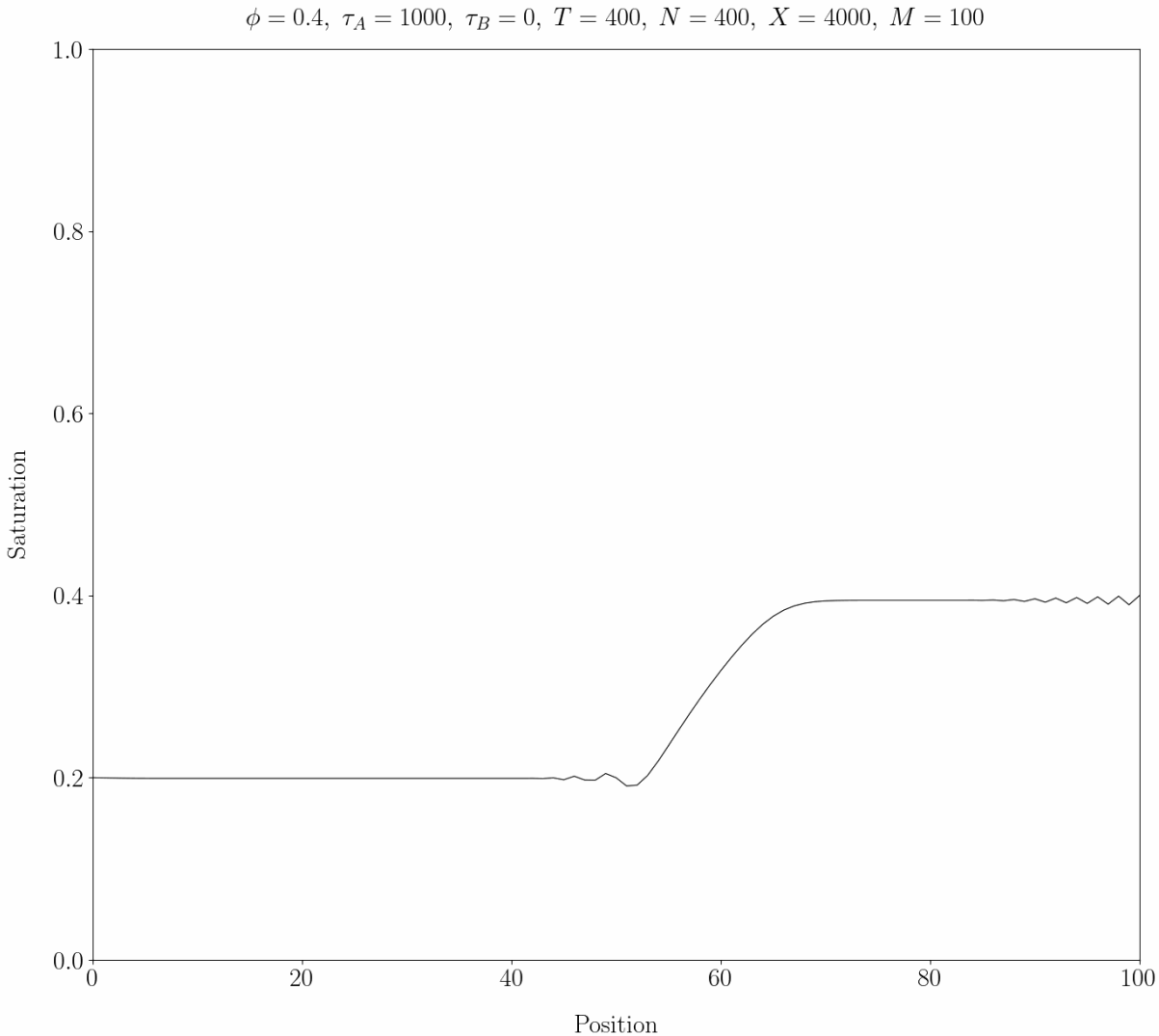
The formula for convergence uses an error tolerance that is predetermined by the user.

Simulation Results: Rarefaction

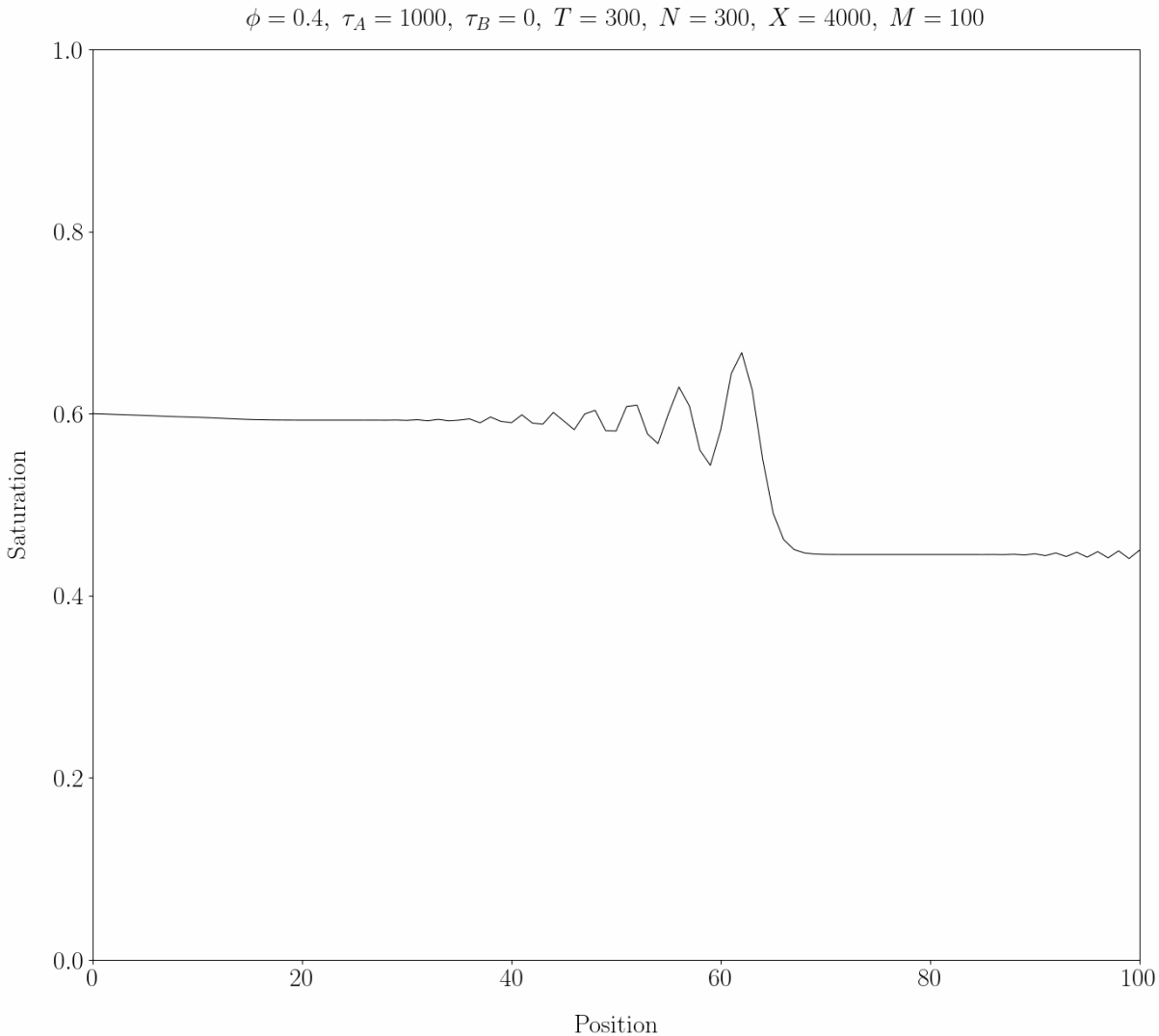
We were able to replicate all the solution structures.

Simulation Results: Rarefaction

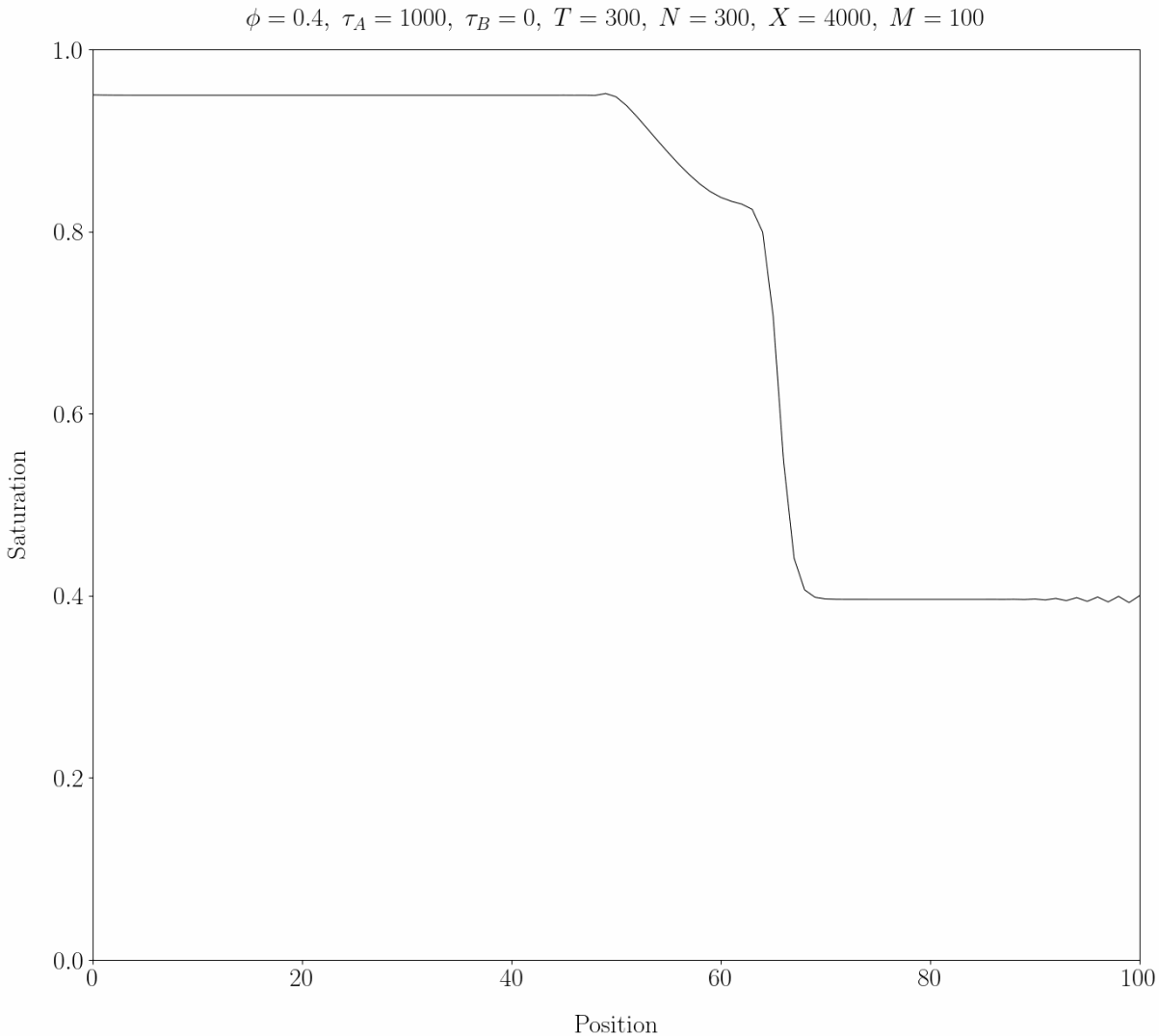
We were able to replicate all the solution structures.



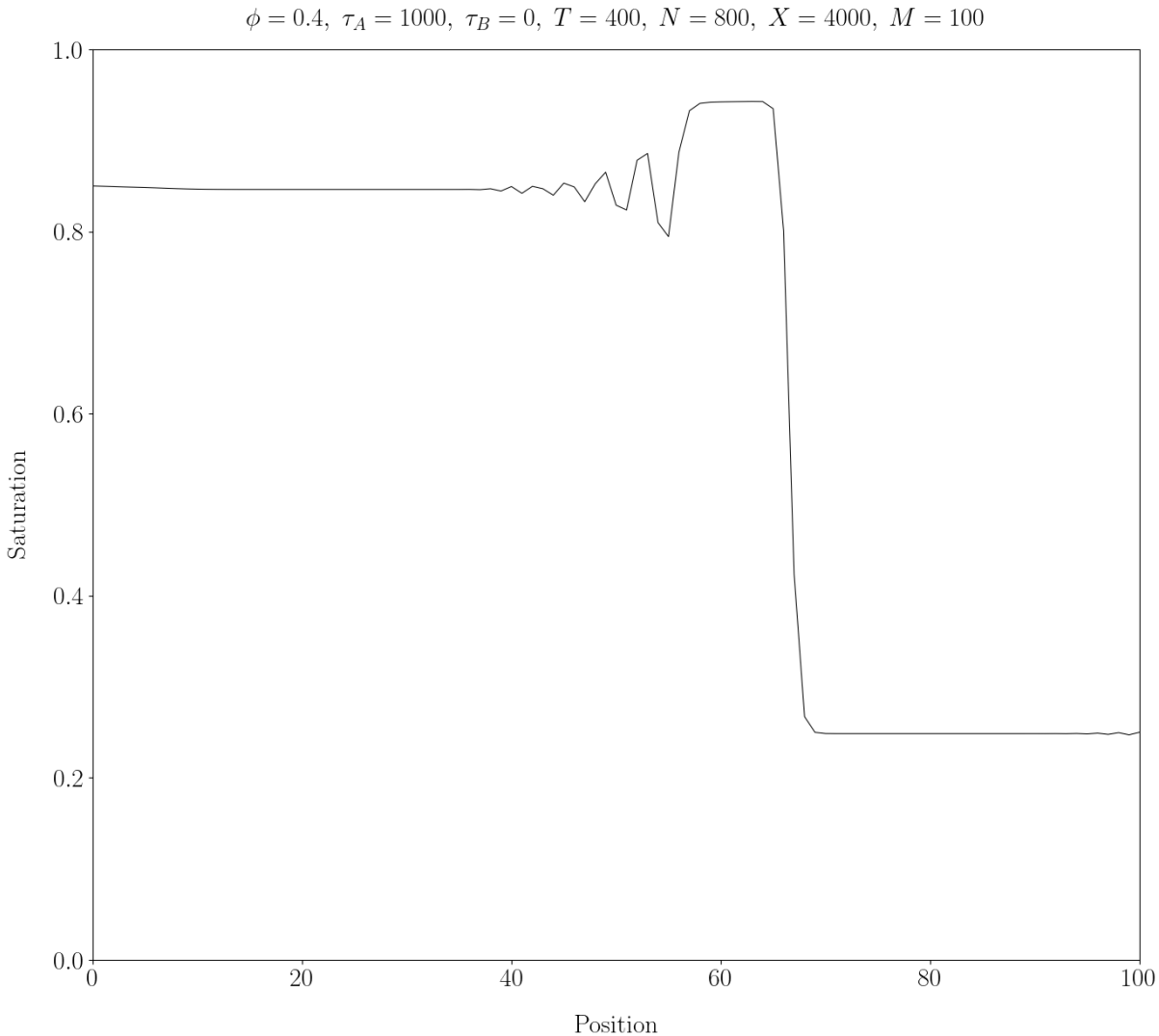
Simulation Results: Lax Shock



Simulation Results: Rarefaction Shock



Simulation Results: Double Shock



Speed

The execution time is linear in the number of time steps.

Speed

The execution time is linear in the number of time steps.

Previous simulations took ~ 5 hours to accurately simulate double shock behavior.

Speed

The execution time is linear in the number of time steps.

Previous simulations took ~ 5 hours to accurately simulate double shock behavior.

In Python:

Speed

The execution time is linear in the number of time steps.

Previous simulations took ~ 5 hours to accurately simulate double shock behavior.

In Python:

Time Steps	Build Time
100	0.4 sec
200	0.8 sec
500	2 sec
1000	4 sec

Speed

The execution time is linear in the number of time steps.

Previous simulations took ~ 5 hours to accurately simulate double shock behavior.

In Python:

Time Steps	Build Time
100	0.4 sec
200	0.8 sec
500	2 sec
1000	4 sec

This is about $20,000$ times faster than the previous simulation.

Future Work

Future work will correctly implement τ_B into the simulation.

Future Work

Future work will correctly implement τ_B into the simulation.

Currently, it is implemented, but nonlinear PDEs do NOT like to be easily simulated. The current simulation breaks down after the inequality

$$\frac{\tau_B}{\tau_A} \geq 0.0001$$

is satisfied. If $\tau_A = 10^7$, then this becomes $\tau_B \geq 1000$.

Future Work

Future work will correctly implement τ_B into the simulation.

Currently, it is implemented, but nonlinear PDEs do NOT like to be easily simulated. The current simulation breaks down after the inequality

$$\frac{\tau_B}{\tau_A} \geq 0.0001$$

is satisfied. If $\tau_A = 10^7$, then this becomes $\tau_B \geq 1000$.

We know that τ_A is somewhere around 10^7. Not much is known about τ_B.

Future Work

Future work will correctly implement τ_B into the simulation.

Currently, it is implemented, but nonlinear PDEs do NOT like to be easily simulated. The current simulation breaks down after the inequality

$$\frac{\tau_B}{\tau_A} \geq 0.0001$$

is satisfied. If $\tau_A = 10^7$, then this becomes $\tau_B \geq 1000$.

We know that τ_A is somewhere around 10^7. Not much is known about τ_B.

The program also does not like very large values of τ_A. This may have to do with other parameters not being callibrated correctly.

Future Work

Future work will correctly implement τ_B into the simulation.

Currently, it is implemented, but nonlinear PDEs do NOT like to be easily simulated. The current simulation breaks down after the inequality

$$\frac{\tau_B}{\tau_A} \geq 0.0001$$

is satisfied. If $\tau_A = 10^7$, then this becomes $\tau_B \geq 1000$.

We know that τ_A is somewhere around 10^7. Not much is known about τ_B.

The program also does not like very large values of τ_A. This may have to do with other parameters not being callibrated correctly.

Future work will also use the current program to compare real data with simulated data in order to get a more accurate value of τ_A.

References

[1] Kimberly Spayd. “Generalizing the Modified Buckley–Leverett Equation with TCAT Capillary Pressure”. In: *Euro. Jnl of Applied Mathematics* 29 (2018), pp. 338–351.

[2] Eduardo Abreu and Jardel Vieira. “Computing Numerical Solutions of the Pseudo-parabolic Buckley–Leverett Equation with Dynamic Capillary Pressure”. In: *Mathematics and Computers in Simulation* 137 (2017), pp. 29–48.