MXB326 Group Project: Reservoir Simulation

Semester 1, 2025

Name	Student Number
Sheren Zein	n10818120
Sophia Sekulic	n10755861
Zach Eyre	n10818189

Table of Contents

1. Introduction

There is an underground reservoir full of valuable oil that we're trying to extract. On one side of this reservoir, a well bore is installed to collect the oil. However, if we're only reducing the volume of liquids in the reservoir, the lack of internal pressure pushing the oil towards the bore will drive down the collection rate. To remedy this, water is pumped into the opposite side of the reservoir to help displace oil towards the bore. This water injection maintains internal pressure and allows for more consistent oil collection, but introduces a new problem, as we now must avoid accidentally pumping up this water. This can be difficult due to the complexities of a continually changing volume of liquids that are hidden underground.

Avoiding this issue motivates the development of accurately tracking the saturation of oil within the space to identify the expected time for the water to reach the well bore, referred to as the 'breakthrough time', which puts a time limit on the oil collection. Inaccurate timings that are significantly off can ruin the collection process or needlessly leave oil behind.

Since oil and water have different viscosities, the computational models to predict the simultaneous flow of the fluids in a complex irregular three-dimensional space become computationally intensive. However, by restricting it to a horizontal, one-dimensional reservoir, the model can be simplified into a single partial

differential equation that can be more easily implemented into a computation model. This report aims to develop both a numerical and semi-analytical model for these simplified oil reservoirs, to better identify and understand the fluid dynamics.

2. Mathematical Model

The foundation of this equation comes from the conservation of mass for the immiscible and incompressible phases of water (w) and oil (o) as given by their transport equations,

$$\phi \frac{\partial S_w}{\partial t} + \frac{\partial q_w}{\partial x} = 0 \tag{1}$$

$$\phi \frac{\partial S_o}{\partial t} + \frac{\partial q_o}{\partial x} = 0 \tag{2}$$

Where:

- x and t are the variables for space and time
- ϕ is the porosity of the reservoir
- S is the saturation of the reservoir by the fluids, a further volume constraint requires that $S_w + S_o = 1$ which allows for S_o to be the only variable we care about instead of modeling both oil and water saturation. Additionally, we define S_{wr} and S_{or} as the minimum saturation values for each phase as some small amount will remain trapped inside the reservoir in small pockets.
- q_o and q_w are the flow rates per unit cross-sectional area of the reservoir

The underpinning concept of the model is the relation between the rate of change for the saturations being linked to the flow rates for each liquid. To further simplify, we assume a 'water-wet' porous medium where the capillary pressure is positive, meaning water prefers to occupy the small pores of the reservoir and will push out the oil. This guarantees that eventually the continued injection of water should force out all the available oil.

Next, we define the initial and boundary conditions for the reservoir where initially the reservoir is full to the brim. Oil saturation will be at the maximum value as it nearly occupies the entire space except for the irreducible amount of trapped water S_{wr}.

$$S(x, 0) = 1 - S_{wr} \tag{3}$$

Water injection at a fixed rate q from the side of the reservoir at $x = 0$ is given by the following, which states only water will be flowing into the system

$$q_w(0, t) = q \quad \text{and} \quad q_o(0, t) = 0 \tag{4}$$

Finally, assuming the reservoir is semi-infinite, the saturation of oil far away from the water inlet at $x = 0$ will be unchanged from the initial state since it'll take a similarly infinite amount of time for water to affect it

$$\lim_{x \to \infty} S\,(x, t) = 1 - S_{wr} \tag{5}$$

With these as a framework, an exact solution for the saturation changes over time with a constant rate of water injection can be developed. Following methods from Fokas and Yortsos, this model can be developed with the restriction that the water-to-oil viscosity ratio F behaves as the following

$$F = \frac{1 - S_{wr} + \frac{\gamma}{\beta}}{S_{or} + \frac{\gamma}{\beta}} \tag{6}$$

The full model can now be reduced to the following one-dimensional initial boundary value problem for the oil saturation $S(x,t)$.

$$\frac{\partial S}{\partial t} = \frac{\partial}{\partial x}\left[g(S)\frac{\partial S}{\partial x} + f(S)\right] \tag{7}$$

$$S(x,0) = 1 - S_{wr} \tag{8}$$

$$\frac{\partial S}{\partial x}(0,t) = \frac{\alpha}{\beta}(\beta S(0,t) + \gamma) + \frac{\omega}{\beta}(S(0,t) + \gamma)^2 \tag{9}$$

$$\lim_{x\to\infty} S(x,t) = 1 - S_{wr} \tag{10}$$

The variables S_{wr}, S_{or}, F and β are prespecified in the task outline allowing for γ, ω and α to be determined using the following relations.

$$\gamma = \frac{\beta(1 - S_{wr} - FS_{or})}{F - 1} \tag{11}$$

$$\alpha = -\frac{\beta^2\left(S_{or} + \frac{\gamma}{\beta}\right)\left(1 - S_{wr} + \frac{\gamma}{\beta}\right)}{1 - S_{wr} - S_{or}} \tag{12}$$

$$\omega = \beta - \frac{\alpha}{\beta - \beta S_{wr} + \gamma} \tag{13}$$

The capillary-hydraulic properties of the fluid-porous system functions $f(S)$ and $g(S)$ are written as the following to allow them to be exactly solvable and can be seen below.

$$f(S) = \frac{\alpha}{\beta^2}\left[\frac{1}{1 - S_{wr} + \frac{\gamma}{\beta}} - \frac{1}{S + \frac{\gamma}{\beta}}\right], \qquad g(S) = \frac{1}{(\beta S + \gamma)^2}$$

3. Semi-Analytical Solution

The following transformation linearises (7) and allows for the derivation of a semi-analytical solution of the initial-boundary value problem (IVP):

$$\tilde{x} = \int_0^x (\beta S(\xi,t) + \gamma)\, d\xi + \omega t \tag{14}$$

This specifically identifies the oil saturation as follows:

$$S(x,t) = \frac{1}{\beta}\left[\frac{\alpha e^{\alpha x}}{\phi_{\tilde{x}}(\tilde{x},t)} - \gamma\right] \tag{15}$$

where, for values of x and t, $\tilde{x}$ is the solution of the nonlinear equation

$$\phi(\tilde{x},t) = e^{\alpha x} \tag{16}$$

The function $\phi(\tilde{x},t)$ in (15) and (16) is defined as

$$\phi(\tilde{x},t) = \frac{1}{2}erfc\left(\frac{\tilde{x}}{2\sqrt{t}}\right) + \frac{1}{2}exp(-\omega\tilde{x} + \omega^2 t)erfc\left(\frac{\tilde{x}}{2\sqrt{t}} - \omega\sqrt{t}\right) - \frac{1}{2}exp(-\beta\tilde{x} + \beta^2 t)erfc\left(\frac{\tilde{x}}{2\sqrt{t}} - \beta\sqrt{t}\right)$$
$$- \frac{1}{2}exp(-(\omega - \beta)\tilde{x} + (\omega - \beta)^2 t)erfc\left(\frac{\tilde{x}}{2\sqrt{t}} - (\omega - \beta)\sqrt{t}\right) + \exp\left((\beta - \omega)\tilde{x} + (\beta - \omega)^2 t\right)$$

The semi-analytical function computes the above solution of the IVP by taking the minimum saturations of water (S_{wr}) and oil (S_{or}), relative magnitude of viscous to capillary terms (β), water-to-oil viscosity ratio (F), domain length (L), spatial resolution (N), time (T), and time steps (M) as inputs. These inputs then return an $N \times (M+1)$ solution matrix as the output. To determine $\phi_{\tilde{x}}(\tilde{x},t) = \frac{\partial \phi}{\partial \tilde{x}}(\tilde{x},t)$, a symbolic function is created and then differentiated with respect to $\tilde{x}$. The obtained symbolic derivative expression is first converted to string, then an anonymous function, to then be evaluated as part of the solution.

The value of $\tilde{x}$ corresponding to $x = x_i$ and $t = t_{n-1}$ to solve the nonlinear equation (16) is found using MATLAB's 'fzero' function with an initial guess, $\tilde{x}^{(0)}$ obtained by first evaluating (14) at $x = x_i$ and $t = t_{n-1}$.

$$\tilde{x}^{(0)} = \int_0^{x_i} (\beta S(\xi, t_{n-1}) + \gamma)\, d\xi + \omega t_{n-1} \tag{17}$$

The above equation is then approximated using MATLAB's 'trapz' function which utilizes the trapezoidal rule (with $i-1$ sub-intervals) and the previously computed values of $S(x_j, t_{n-1})$ where $j = 1, \dots, i$.

4. Numerical Solution

The numerical solution function approximates the solution of $(7)-(10)$ with the following initial-boundary value problem on a finite domain given a suitably large value of L.

$$\frac{\partial S}{\partial t} = \frac{\partial}{\partial x}\left[g(S)\frac{\partial S}{\partial x} + f(S)\right] \tag{18}$$

$$S(x,0) = 1 - S_{wr} \tag{19}$$

$$g\big(S(0,t)\big)\frac{\partial S}{\partial x}(0,t) + f\big(S(0,t)\big) = 1 \tag{20}$$

$$g\big(S(L,t)\big)\frac{\partial S}{\partial x}(L,t) + f\big(S(L,t)\big) = 1, \text{or } \frac{\partial S}{\partial x}(L,t) = 0 \tag{21}$$

The equivalence of the boundary conditions (9) and (20) was shown in MATLAB by rearranging (20) as $\frac{\partial S}{\partial x}(0,t) = \frac{1-f(S(0,t))}{g(S(0,t))}$, creating symbolic expressions in MATLAB, and testing their equivalence through MATLAB's 'isequal' function. As this returned a logical output of '1', the expressions were equivalent.

Within the numerical solution function, the PDE (18) was discretised in space using the vertex centered finite volume method. This divided the domain into nodes and control volumes, constructed control volumes around each node, and integrated the PDE (18) over each volume. This resulted in a system of non-linear equations in the form of equation (22), describing the time derivative at each node. The fluxes at the boundaries of each control volume were approximated using finite difference approximations and weighted average approximations of $f(S)$ and $g(S)$. To allow non-uniform node spacing, the variable x describing node locations was included as an input of the function. Additionally, the inputs 'BC_type' and 'sigma' were implemented to allow for different boundary conditions at $x = L$ and different weighted approximations for $f(S)$ and $g(S)$. For $\sigma = \frac{1}{2}$, the weighted approximation is known as '*averaging*' and is second-order accurate in space. Alternatively, when $\sigma = 0$ or $\sigma = 1$, the weighted approximation is known as '*upwinding*' and is first-order accurate in space. After testing both $\sigma = \frac{1}{2}$ and $\sigma = 0$, it was decided that averaging $\sigma = \frac{1}{2}$ produced a more accurate numerical solution as $\sigma = 0$ resulted in a solution with a smeared front. However, to ensure there were no oscillations in the solution, the node spacing had to be tested and adjusted. The resulting system of equations were assembled into a vector (23) in MATLAB through the subfunction 'Gfunc', where the i^{th} entry corresponds to $G_i(S)$.

$$\frac{dS_i}{dt} = \frac{1}{V_i}\left[q_{w_i} - q_{e_i}\right] + R_i = G_i(S) \tag{22}$$

$$\frac{dS}{dt} = G(S), \quad where\ S = [S_1, \dots, S_N]^T\ and\ G(S) = [G_1(S), \dots, G_N(S)]^T \tag{23}$$

The function then discretises the PDE in time through the theta method. 'Theta' was kept as an input to allow for different variants of the theta method, such as the Backward Euler (BE) ($\theta = 1$) and the Crank-Nicolson (CN) $\left(\theta = \frac{1}{2}\right)$ methods. By integrating equation (23) from $t = t_n$ to $t = t_{n+1} = t_n + \delta t$ and approximating the resulting integrals of the flux and reaction/source terms using a weighted θ approximation, the following equation is obtained:

$$S^{i+1} - S^i = \delta t\left[\theta G\left(S^{i+1}\right) + (1-\theta)G\left(S^i\right)\right] \tag{24}$$

These equations provide a relationship between the current and previous time step, $[{S_1}^{n+1}, \dots, {S_N}^{n+1}]^T$ and $[{S_1}^n, \dots, {S_N}^n]^T$, in the form of a system of linear equations. As $[{S_1}^0, \dots, {S_N}^0]^T$ is given from the initial condition, completing M time steps obtains the numerical solutions from $[{S_1}^0, \dots, {S_N}^0]^T$ to $[{S_1}^M, \dots, {S_N}^M]^T$. In MATLAB, the solution ends at $M + 1$ as the indexing begins at 1. By moving all terms to one side, another system of nonlinear equations of the form $F(S^{i+1})$ is stored in the function 'Ffunc'.

$$F(S^{i+1}) = 0 = S^{i+1} - S^i - \delta t\left[\theta G\left(S^{i+1}\right) + (1-\theta)G\left(S^i\right)\right] \tag{25}$$

To solve this system, Newton's method is employed through the subfunction 'newton_solver', which computes Newton iterations until the norm of the solution is less than a specified tolerance, or the maximum number of iterations is reached. This subfunction includes various Jacobian generation techniques (analytical, finite difference with column-wise or tridiagonal structure), different line-searching techniques (simple, two-point parabolic, three-point parabolic), different methods for solving the linear system at each iteration (backslash with full matrix, sparse format, or tridiagonal algorithms) and different methods of storing the Jacobian which will be discussed. Through a for loop, the 'Ffunc' and 'newton_solver' functions are used in conjunction to compute the solution at the next time step.

Given an initial guess, $S^{(0)}$, the Newton's iteration takes the form:

$$S^{(k+1)} = S^{(k)} - J\left(S^{(k)}\right)^{-1} F\left(S^{(k)}\right) \tag{26}$$

where $S = (S_1, \dots, S_N)^T$, F is the vector-valued function (25) and J is the Jacobian matrix of F. The numerical solution allows for different techniques to compute the Jacobian. This includes computing the Jacobian analytically, or through finite differences, either column-wise or by exploiting its tridiagonal structure. The analytical Jacobian method was employed by computing the Jacobian of $F(S)$ manually and creating the subfunction 'Jfunc' which evaluates the Jacobian at S.

As the analytical Jacobian can be difficult to compute, an alternative method approximates the derivatives using finite differences. Typically, a first-order forward difference approximation is used, whereby the *j*th column of the Jacobian matrix is approximated by (27) for a suitably small value $\varepsilon > 0$, where e_j is the *j*th column of the $N \times N$ identity matrix (Carr, 2025)

$$J_{*,j}(x) = J(x)e_j \approx \tilde{J}_{*,j}(x) = \frac{F\left(x + \varepsilon e_j\right) - F(x)}{\varepsilon}, \qquad \varepsilon = \begin{cases} \sqrt{\in_M}\ \|x\|_2\ , if\ x \neq 0 \\ \sqrt{\in_M}\ , if\ x = 0 \end{cases} \tag{27}$$

$\in_M$ is the unit roundoff which is denoted by 'eps' in MATLAB and equal to $2.2204e-16$. When the Jacobian matrix is tridiagonal, it is possible to shift multiple variables at once provided their corresponding

columns in J do not overlap, instead of shifting one variable at a time (x_j). For a given vector $x \in \mathbb{R}^N$, each of the three shift vectors can be approximated as:

$$J(x)s_j \approx \frac{F(x + \varepsilon_j s_j) - F(x)}{\varepsilon_j}, \qquad \varepsilon_j = \begin{cases} \dfrac{\sqrt{\epsilon_M}\ \|x\|_2}{\|s_j\|_2}, if\ x \neq 0 \\ \dfrac{\sqrt{\epsilon_M}}{\|s_j\|_2}, if\ x = 0 \end{cases} \tag{28}$$

and ϵ_M is the unit roundoff also denoted by 'eps' in MATLAB. The tridiagonal Jacobian matrix can then be formed by appropriately indexing into and extracting entries from the shift vectors Js_1, Js_2 and Js_3.

In the numerical solution, there are three different methods for storing the Jacobian. Full storage is when every individual entry of the Jacobian is stored explicitly. This is the default method used in the numerical solution and is chosen when 'linear_method' is 'none'. Sparse storage refers to when only the nonzero elements of the Jacobian matrix are stored and can be done via the 'sparse' function in MATLAB. Lastly, when only the sub, main and super diagonal of a tridiagonal Jacobian are stored as vectors, this is known as tridiagonal storage ('sparsediag' in numerical solution). The subfunction 'newton_solver' holds all the algorithms for computing these storage methods.

Line searching Newton methods are a class of globally convergent methods that attempt to limit the size of the Newton step by considering the modified Newton iteration:

$$x^{(k+1)} = x^{(k)} + \lambda^{(k)}\delta x^{(k)}, \delta x^{(k)} = -J\left(\delta x^{(k)}\right)^{-1} F\left(x^{(k)}\right) \tag{29}$$

where $0 < \lambda^{(k)} \leq 1$. The value of $\lambda^{(k)}$ is selected to ensure the decrease of the nonlinear residual norm during each Newton iteration. According to Carr (2025), simple line searching chooses $\lambda^{(k)}$ as half of its previous value until the resulting step produces a reduction in the nonlinear residual norm. In practice, the Armijo rule of simple line searching is applied as follows:

$$\left\|F\left(x^{(k)}\right) + \lambda^{(k)}\delta x^{(k)}\right\| < \left(1 - \alpha\lambda^{(k)}\right)\left\|F\left(x^{(k)}\right)\right\| \tag{30}$$

where α is a small, positive constant. To allow for greater sophisticated line searching strategies, the algorithm step $\lambda^{(k)} = \lambda^{(k)}/2$ is replaced with $\lambda^{(k)} = \sigma\lambda^{(k)}$ where $\sigma \in [\sigma_0, \sigma_1]$ and $0 < \sigma_0 < \sigma_1 < 1$. The rejected values of $\lambda^{(k)}$ can be used to model the scalar-valued function

$$g(\lambda) = \left\|F\left(x^{(k)}\right) + \lambda^{(k)}\delta x^{(k)}\right\|_2^2 \tag{31}$$

with a polynomial $p(\lambda)$, where chosen values of $\lambda^{(k)}$ minimizes $p(\lambda)$. Two polynomials of degree two (parabolic models) are implemented within the numerical solution: the two-point and three-point parabolic models. In the two-point parabolic model, the known values of $g(0)$, $g'(0)$ and $g\left({\lambda_1}^{(k)}\right)$ are used to build a quadratic approximation to $g(\lambda)$, where ${\lambda_1}^{(k)}$ is the most recently rejected value of $\lambda^{(k)}$. The quadratic that satisfies all conditions is

$$p(\lambda) = g(0) + g'(0)\lambda + \left(\frac{g\left({\lambda_1}^{(k)}\right) - g(0) - g'(0){\lambda_1}^{(k)}}{{\lambda_1}^{(k)^2}}\right)\lambda^2 \tag{32}$$

The next candidate value of $\lambda^{(k)}$ is then given by the solution of $p'(\lambda) = 0$. In the three-point parabolic model, the known values of $g(0)$, $g\left({\lambda_1}^{(k)}\right)$ and $g\left({\lambda_2}^{(k)}\right)$ are used to construct a quadratic approximation of $g(\lambda)$, where ${\lambda_1}^{(k)}$ and ${\lambda_2}^{(k)}$ are the two most recently rejected value of $\lambda^{(k)}$. The quadratic that satisfies the necessary conditions is

$$p(\lambda) = g(0) + \frac{\lambda}{{\lambda_1}^{(k)} - {\lambda_2}^{(k)}}\left(\frac{\left(\lambda^{(k)} - {\lambda_2}^{(k)}\right)\left(g\left({\lambda_1}^{(k)}\right) - g(0)\right)}{{\lambda_1}^{(k)}} + \frac{\left({\lambda_1}^{(k)} - \lambda\right)\left(g\left({\lambda_2}^{(k)}\right) - g(0)\right)}{{\lambda_2}^{(k)}}\right) \quad (33)$$

Then, like the two-point model, the next candidate value of $\lambda^{(k)}$ is given by the solution of $p'(\lambda) = 0$. All three applied algorithm for simple line searching, two-point and three-point line searching can be found in the subfunction 'newton_solver' of the numerical solution.

5. Analysis and Findings

5.1.1 Plotting the Semi-analytical and Numerical Solutions

The size of the time step dt was selected such that each time was a multiple of it and would therefore correspond to a column in the solution matrix. This was chosen to be 0.0025 so that the times 0.01,0.05,0.1,0.3,0.8125,1 corresponded to columns 5,21,41,121,326,401 of the solution matrix respectively.

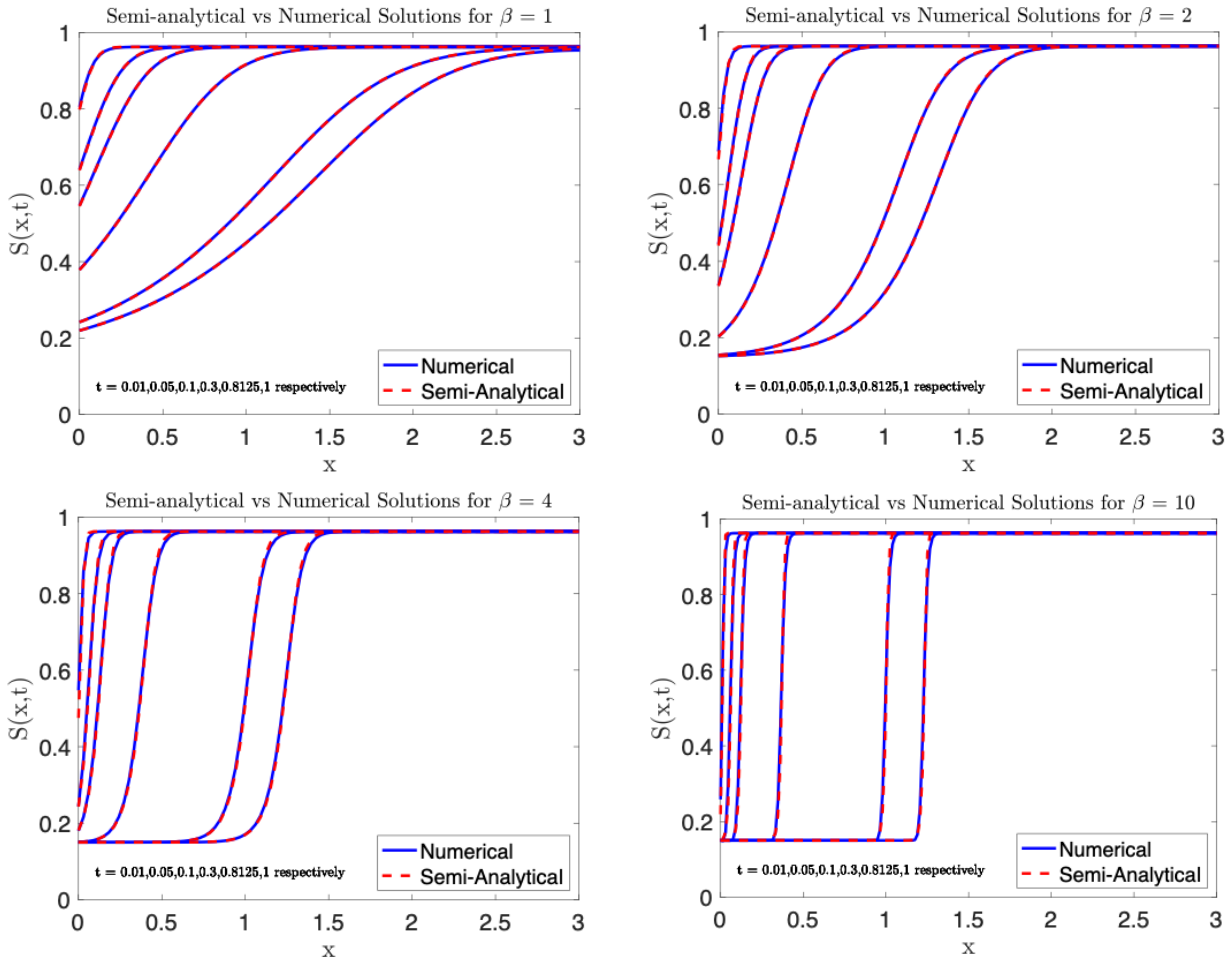

Figure 1: Semi analytical and numerical solutions at t = 0.01,0.05,0.1,0.3,0.8125,1 for different injection rates

Figure 1 displays comparisons of the semi-analytical and numerical solutions at the provided times for different injection rates. It appears that the numerical solution provides a good approximation to the

semi-analytical solution in all cases. Although only slight, the numerical solution looks be a better approximation for the smaller values of beta.

5.1.2 True Boundary Condition Approximation

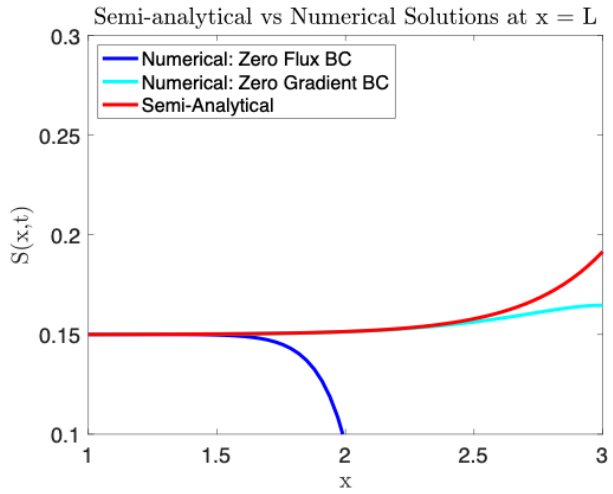

Figure 2: Semi-analytical vs numerical solutions using different boundary conditions at $x = L$

Figure 2 suggests that the zero-gradient Neumann boundary condition $\frac{\partial S}{\partial x}(L,t) = 0$ in the numerical solution best approximates the true boundary condition as $x \to \infty$.

This is supported by the working below which shows how the boundary condition is approximated once the PDE is limited to a finite domain.

$$\lim_{x\to\infty} S(x,t) = 1 - S_{wr} \implies \lim_{x\to L} S(x,t) \approx 1 - S_{wr}$$

$$\lim_{x\to L} \frac{\partial}{\partial x} S(x,t) \approx \frac{\partial}{\partial x}(1 - S_{wr})$$

$$\lim_{x\to L} \frac{\partial S}{\partial x}(x,t) \approx 0$$

$$\frac{\partial S}{\partial x}(L,t) \approx 0$$

5.2 Average Reservoir Oil Saturation

A differential equation for the average reservoir oil saturation can be found by integrating the governing partial differential equation (18) over the spatial interval $[0, L]$.

$$\frac{\partial S}{\partial t} = \frac{\partial}{\partial x}\left[g\big(S(x,t)\big)\frac{\partial S}{\partial x}(x,t) + f(S(x,t))\right]$$

$$\int_0^L \frac{\partial S}{\partial t}\,\partial x = \int_0^L \left[\frac{\partial}{\partial x}\left(g\big(S(x,t)\big)\frac{\partial S}{\partial x}(x,t) + f\big(S(x,t)\big)\right)\right]\partial x$$

$$\frac{\partial}{\partial t}\int_0^L S(x,t)\,\partial x = \left(g\big(S(L,t)\big)\frac{\partial S}{\partial x}(L,t) + f\big(S(L,t)\big)\right) - \left(g\big(S(0,t)\big)\frac{\partial S}{\partial x}(0,t) + f\big(S(0,t)\big)\right)$$

Introduce spatial average of S: $\bar{S}(t) = \frac{1}{L}\int_0^L S(x,t)\,\partial x$

$$\frac{\partial}{\partial t}(L\bar{S}(t)) = \left(g\big(S(L,t)\big)\frac{\partial S}{\partial x}(L,t) + f\big(S(L,t)\big)\right) - \left(g\big(S(0,t)\big)\frac{\partial S}{\partial x}(0,t) + f\big(S(0,t)\big)\right)$$

$$\frac{\partial \bar{S}(t)}{\partial t} = \frac{1}{L}\left[\left(g\big(S(L,t)\big)\frac{\partial S}{\partial x}(L,t) + f\big(S(L,t)\big)\right) - \left(g\big(S(0,t)\big)\frac{\partial S}{\partial x}(0,t) + f\big(S(0,t)\big)\right)\right]$$

BC at $x = 0$:

$$g\big(S(0,t)\big)\frac{\partial S}{\partial x}(0,t) + f\big(S(0,t)\big) = 1$$

Zero flux BC at $x = L$:

$$g\big(S(L,t)\big)\frac{\partial S}{\partial x}(L,t) + f\big(S(L,t)\big) = 0$$

$$\frac{\partial \bar{S}(t)}{\partial t} = \frac{1}{L}[(0) - (1)]$$

$$\frac{\partial \bar{S}(t)}{\partial t} = \frac{-1}{L}$$

Zero gradient BC $x = L$:

$\frac{\partial S}{\partial x}(L,t) = 0$, assume $S(L,t) = 1 - S_{wr}$

$$\frac{\partial \bar{S}(t)}{\partial t} = \frac{1}{L}\left[\big(g(1 - S_{wr}) \times 0 + f(1 - S_{wr})\big) - (1)\right]$$

$$\frac{\partial \bar{S}(t)}{\partial t} = \frac{1}{L}\left[\frac{\alpha}{\beta^2}\left[\frac{1}{1 - S_{wr} + \frac{\gamma}{\beta}} - \frac{1}{1 - S_{wr} + \frac{\gamma}{\beta}}\right] - 1\right]$$

$$\frac{\partial \bar{S}(t)}{\partial t} = \frac{1}{L}\left[\frac{\alpha}{\beta^2}[0] - 1\right]$$

$$\frac{\partial \bar{S}(t)}{\partial t} = \frac{-1}{L}$$

Initial condition:

$$\bar{S}(0) = \frac{1}{L}\int_0^L S(x,0)\,\partial x$$

$$\bar{S}(0) = \frac{1}{L}\int_0^L (1 - S_{wr})\,\partial x$$

$$\bar{S}(0) = \frac{1 - S_{wr}}{L}[(L) - (0)]$$

$$\bar{S}(0) = 1 - S_{wr}$$

Resulting IVP:

$$\int \frac{\partial \bar{S}(t)}{\partial t}\partial t = \int \frac{-1}{L}\partial t$$

$$\bar{S}(t) = \frac{-t}{L} + C$$

$$\bar{S}(0) = \frac{-0}{L} + C$$

$$\bar{S}(0) = C = 1 - S_{wr}$$

$$\bar{S}(t) = \frac{-t}{L} + 1 - S_{wr}$$

Figure 3 plots the numerical and analytical solutions for the average reservoir oil saturation for different injection rates. The analytical solution was plotted using the solution computed above $\bar{S}(t) = \frac{-t}{L} + 1 - S_{wr}$.

On the other hand, the numerical solution for the average reservoir oil saturation was calculated at each time step using the trapezoidal rule and the numerical solution of (18) at the same time step to approximate the integral $\int_0^L S(x,t)\,\partial x$. As shown below, the numerical solution appears to approximate the analytical solution for different injection rates quite well. It deviated the most for the case of $\beta = 1$.

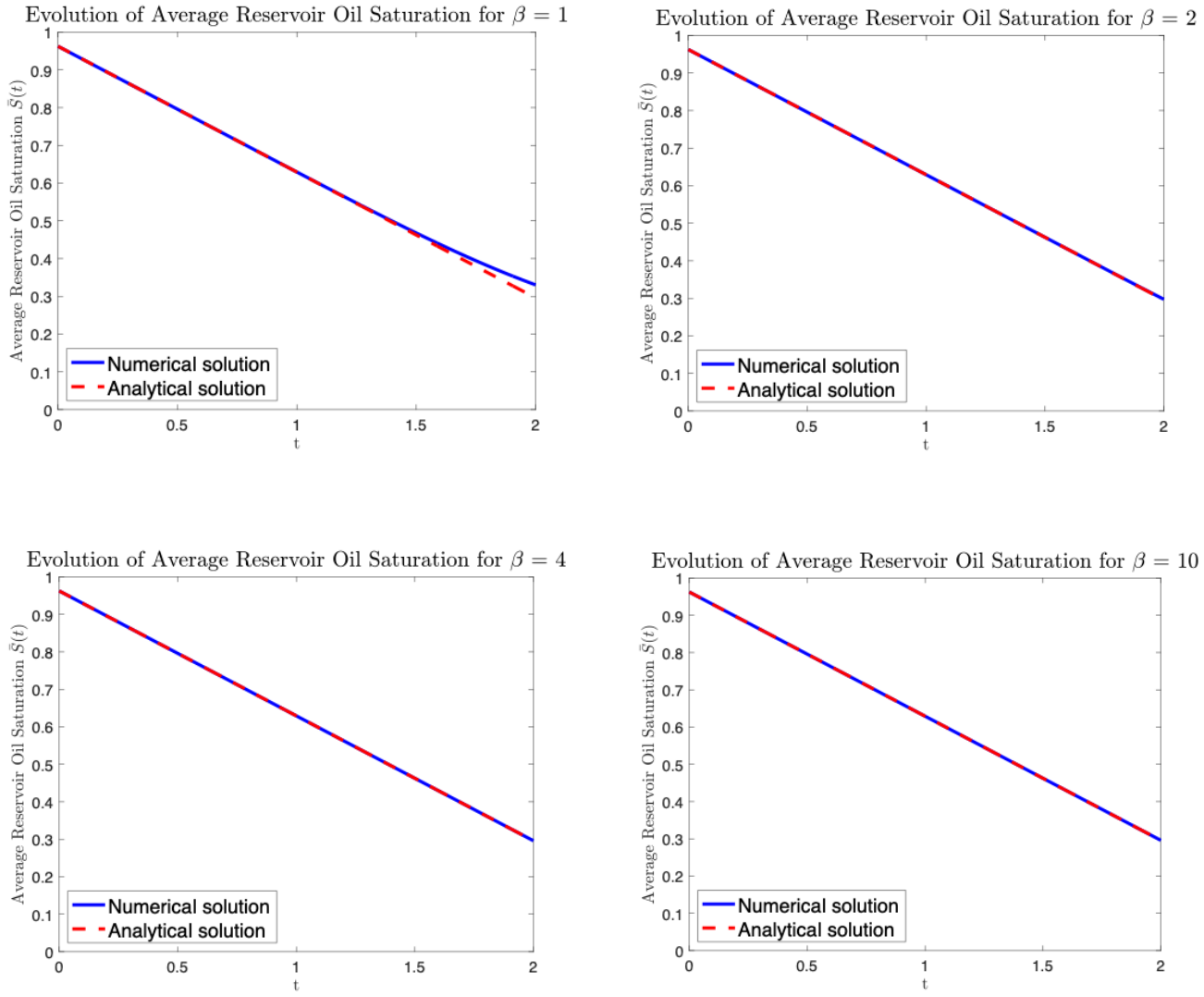

Figure 3: Semi-analytical vs numerical solutions for the evolution of the average reservoir oil saturation for different injection rates

5.3 Efficiency & Accuracy Comparison

To compare the efficiency and accuracy of each of the numerical strategies, the MATLAB function 'timeit' was used in conjunction with a subfunction 'comparison'. The semi-analytical solution was treated as the true solution and acted as the basis for comparison. The general method of testing involved setting default parameters and changing a single parameter to implement the desired strategy. Each strategy was passed in as an anonymous function to then be passed into the 'timeit' function. Additionally, the solution of the strategy was computed. The subfunction 'comparison' would take as input the following:

Sa_true (the semi-analytical solution), methods (a string of names of all the methods tested for efficiency), methods1 (another string of names of methods tested for accuracy), times (the results of the 'timeit' function), solutions (the computed solutions) and finally, tol (the accepted tolerance for relative error). The 'comparison' function outputs a table displaying the individual average time taken to compute the solutions, a list of methods and their computed times in descending order and a second table, showing whether the solutions were within acceptable tolerance.

By inputting both 'theta = 1' and 'theta = 0.5' as parameters in the numerical solution function, the respective BN and CN methods were executed. The global error (accumulation of local errors over all M time steps) is first order in time for BE and second order in time for CN. This was observed when the CN method consistently computed its numerical solution with a lower relative error (0.0002 versus BE's 0.0004). The efficiency of each method differed between runs, possibly due to BE generally converging faster leading to an overall lower runtime or CN being less diffusive thus more stable, leading to fewer iterations. The runtimes of both methods did not differ too greatly in measure. Therefore, it is recommended to use CN due to its higher accuracy when compared to the semi-analytical solution.

As mentioned previously, the numerical solution was discretised in space using the vertex-centered finite volume method. The parameter to be tested involved passing in 'sigma = 0' to implement left upwinding as the function has a positive flow and 'sigma = 0.5' to observe an averaging discretisation. Like the theta methods, the runtimes of both spatial discretisation methods were similar and the order differed between runs deeming parallel efficiency. However, the upwinding discretisation did not compute within acceptable relative error tolerance, so it is recommended to use the averaging discretisation when solving the numerical solution.

Jacobian matrix generation techniques were implemented by altering the 'jacobian' parameter to 'analytical', 'finitecolumn' or 'finitetridiag' which computed the analytical, column-wise and tridiagonal Jacobian matrices, respectively. All methods were relatively accurate, resulting in acceptable relative error tolerances. The column-wise method was the slowest, which was expected as computing the Jacobian and solving a linear system at each iteration makes it expensive for large N value problems. The tridiagonal strategy was slightly more efficient as computations were simplified by only filling the three relevant bands of the Jacobian matrix. The analytical method was indeed the fastest since the exact derivatives of the residual function were manually derived and programmed. Therefore, the recommended Jacobian generation strategy with the best efficiency and accuracy for the numerical solution is the analytical method.

The next numerical strategies involved various line searching methods. This was implemented by changing the parameter 'ls' to 'simple', 'two-point' and 'three-point'. The results obtained were inconclusive as the order of runtimes varied during every run. All line searching methods were within the acceptable tolerance, meaning they shared similar accuracy to the semi-analytical solution. Theoretically, the three-point strategy is the most accurate and often computes within fewer iterations whereas the simple strategy is not optimal in terms of minimising the residual norm and is typically slow. The two-point method theoretically sits in between the two in terms of efficiency and accuracy. As a result, the three-point line searching method should be implemented to compute the numerical solution with the utmost accuracy.

Lastly, different Jacobian matrix storage methods such as dense, sparse and tridiagonal were tested by changing the parameter 'linear_method' to 'full', 'sparse', or 'sparsediag'. It simply alters the way in which the Jacobian elements are stored, so the accuracy of the solutions does not vary and are within acceptable tolerance. The runtimes for each method always resulted in full storage last (as expected), and sparse and tridiagonal varied at being the most efficient. Theoretically, the tridiagonal storage method should be the fastest. However, since MATLAB has a built-in 'sparse' function and the Thomas algorithm must be coded manually, the sparse method is the better choice for storage.

Based on the above information, the ideal numerical solution can be obtained by applying the CN discretisation in time and averaging discretisation in space, combined with the analytical Jacobian method stored sparsely and three-point line searching methods (see Figure 4).

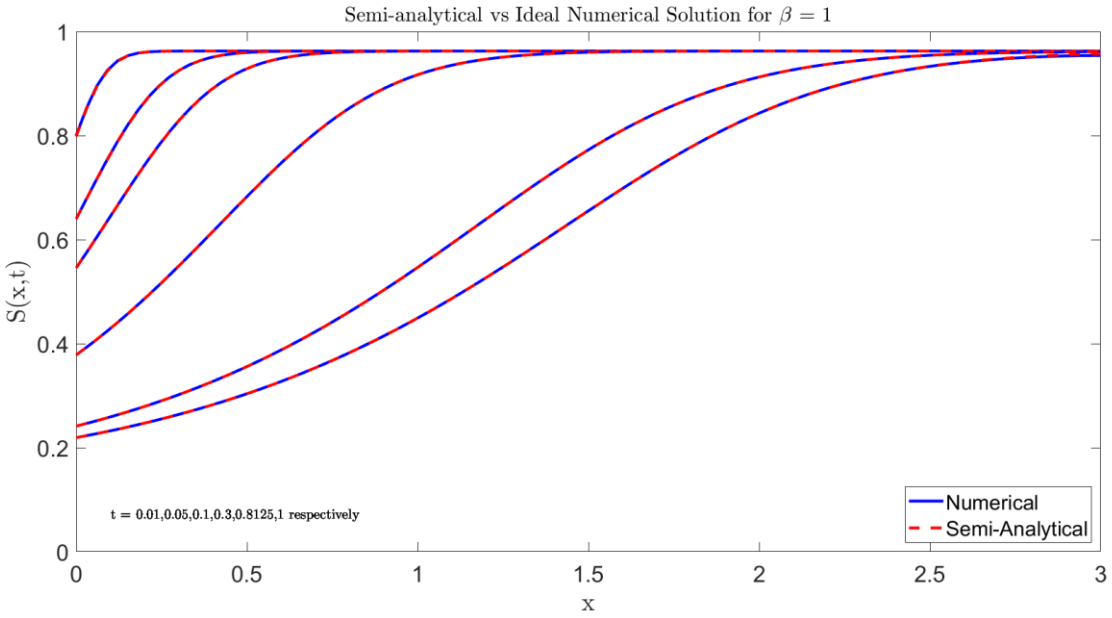

Figure 4: True solution vs ideal numerical solution t = 0.01,0.05,0.1,0.3,0.8125,1 for β=1

5.4 Buckley-Leverett Solution

With increased water injection rates ($\beta \to \infty$), the viscosities of the liquids overshadow any other factors affecting the saturation problem and significantly reduces the complexity required for modelling. It results in a shock front where the fast water jet crashes into the oil and instantly pushes the oil out of the node, leading to two distinct states of either only water or only oil at each node. This step function behaviour is given by the Buckley-Leverett solution

$$S(x,t) = \begin{cases} S_{or}, & x < vt \\ 1 - S_{wr} & x > vt \end{cases}$$

where the shock front's velocity v is calculated as 1.2308 m/s using the following equation

$$v = \frac{1}{1 - S_{wr} - S_{or}}$$

The numeric and semi-analytical models fit this solution well and a visual check and relative error are presented to support this. Seen below in figure (5):

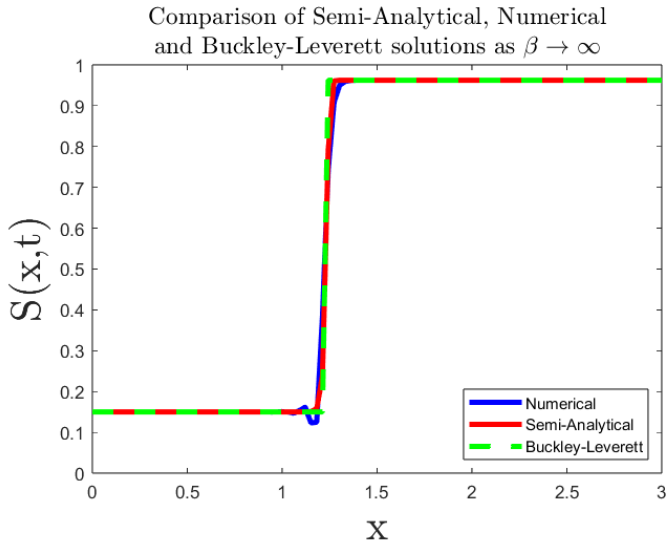

Figure 5: Comparison of modelling step function behaviour of saturations with large beta values

It's evident from this that for large β values, the numerical and semi-analytical models (blue and red) accurately resemble the sharp shock front of the Buckley-Leverett solution (green). This accuracy is reinforced by the average relative error of the numerical and semi-analytical solutions compared to the step function being 3.2×10^{-5} and 5.491×10^{-5} respectively, indicating a high degree of accuracy. From this, the model correctly implements the capillary physics associated with large water injection rates.

5.5 Injection Rates

As the water injection rates increases, reflected with $\beta \to \infty$, the breakthrough time t_B before water reaches the well bore at $x = 1$ is given simply by $t_B = 1 - S_{wr} - S_{or} = 0.8125$ seconds. We can determine the impact of variable β values on breakthrough times by clocking the first instance that the node closest to $x = 1$ decreases its saturation value indicating that water has reached that node.

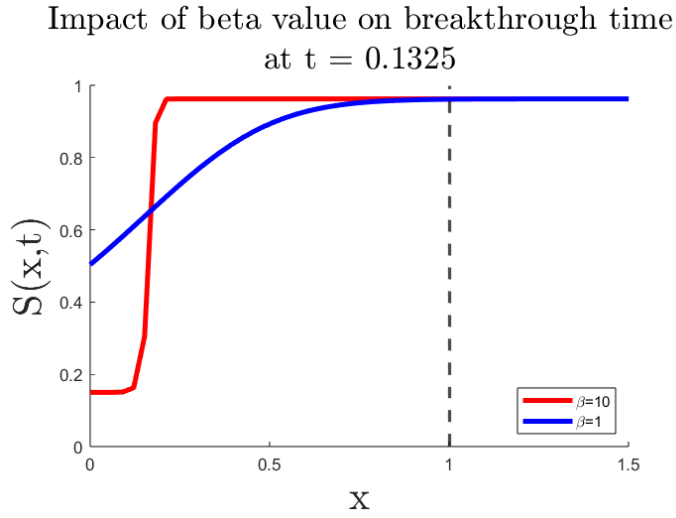

Figure 6: Visualising impact of increased beta value on the breakthrough time before water reaches the well bore (indicated with black dotted line) using the Semi-Analytical model for both

	Breakthrough times (seconds)			
	$\beta = 1$	$\beta = 2$	$\beta = 4$	$\beta = 10$
Semi-Analytical Model	*0.135*	*0.325*	*0.595*	*0.7775*
Numerical Model	*0.13*	*0.3125*	*0.5650*	*0.7150*

Comparing different β values from 1 to 10 shows that the step function behaviour associated with higher values significantly delays. It appears that having to push out a smaller proportion of the oil in these initial nodes prior to $x = 1$ results in smaller β values taking shorter times to breakthrough. This becomes evident when looking at the saturation values for the model at $\beta = 1$, where the water will reach the well bore despite the average saturation values before $x = 1$ being 0.827. Despite only collecting a sixth of the available oil, collection must stop due to the presence of water at the well bore.

Compared to the higher β values that approach a step function appearance and should be expected to extract a lot more oil before risking water getting into the bore. This is supported by the average saturation value of the higher β value at its breakthrough time of 0.7775 seconds, being a considerably lower value of 0.1988. This finding is significant in that it shows that higher water injection rates lead to a greater collection of oil as the water is unable to as easily mix and disperse throughout the entire reservoir.

6. Conclusions

By investigating the simplified water-flooding model, this report successfully developed and implemented both numerical and semi-analytical approaches to simulate reservoir dynamics and analyse the interaction between oil and water flow. The numerical solution performed particularly well for lower values of β and maintained strong alignment with the semi-analytical model across a range of injection rates, though the greatest divergence occurred at $\beta = 1$. It was discovered that applying a zero-gradient Neumann boundary condition at the production end of the reservoir closely replicates the physical behaviour at greater distances, suggesting it is a suitable approximation for modelling open-ended reservoirs. The most accurate and stable numerical results were achieved using CN time discretisation and spatial averaging, in combination with an analytically derived Jacobian stored in sparse format and refined through a three-point line searching method. These approaches effectively captured the sharp saturation front observed in the Buckley-Leverett solution, especially at higher β values. The simulations also confirmed that higher water injection rates improve oil recovery by efficiently displacing oil while limiting mixing and dispersion. Altogether, the modelling framework developed offers valuable insights into the design and optimization of enhanced oil recovery processes in simplified settings.

7. References

Carr, E. (2025). *MXB326 Computational Methods 2: Convergence* [Powerpoint slides]. Canvas.

https://canvas.qut.edu.au/

Carr, E. (2025). *MXB326 Computational Methods 2: Tridiagonal Jacobian* [Powerpoint slides]. Canvas.

https://canvas.qut.edu.au/