MOD 510 Project 2: Can you hear the size of a reservoir?(Group 5)

Parthasarathi Jena Jing Hou Hodjdat Moradi

October 15th, 2023

Contents

1 Abstract

This project's objective is to develop a pressure solver for fluid radial flow within a reservoir and validate the model using real well data. We initiated our work with the background theory and the radial diffusivity equation as a foundation to derive an analytical solution. Building upon this and the finite difference algorithm, we implemented numerical solutions and investigated the accuracy of these approximations under different boundary conditions, including both lazy and non-lazy cases, as well as varied grid sizes. Our main finding was that the errors of lazy case scale linearly with the grid size when compared to the analytical true solution. This aligned with our theoretical expectations. The error for the non-lazy case, however, remained at zero in the same scenario.

We have successfully implemented pressure solver classes that can handle both steady-state and transient flow conditions. The performance of several solvers, including Dense, Sparse, and the Thomas Algorithm, was rigorously evaluated. In general, when dealing with large matrices, sparse matrices prove to be a more efficient choice in terms of time and memory utilization. However, for smaller matrices, the use of dense matrices via NumPy libraries is a good choice. Remarkably, the Thomas Algorithm emerged as the top-performing solver that fits for handling both small and large matrices.

At last, we applied our model to fit well test data, as well as utilizing techniques such as the line-source solution and estimated the volume of water.

2 Introduction

Our research focuses on the extended version of the continuity equation, specifically the radial diffusivity equation, both in theoretical exploration and practical application within our numerical pressure solver models. Our initial phase involves gaining a deep understanding of the foundational principles and deriving an analytical solution, which serves as our benchmark for model comparisons. We explore by implementing our models with a simpler, steady-state scenario at first.

Thereafter, we implement more complex, time-variant models, seeking to analyze and refine our approach. we explore a variety of solutions, including NumPy and SciPy libraries, as well as the Thomas algorithm. Our aim is to assess and identify the optimal choice in terms of computational efficiency, ensuring that our numerical models are not only accurate but also efficient in their computation.

2.1 Theory

The mass conservation equation for a reservoir (or any system which produces, transports and accumulates mass) can be written as

$$\frac{\partial q(x,t)}{\partial t}A(x) = -\frac{\partial (J(x,t)A(x)}{\partial x} + \dot{\sigma}(x,t)A(x) \tag{1}$$

where,

A = the cross-sectional area at position x,

q = the mass concentration of fluid

J = the amount of mass flowing into the volume per unit time per unit area (the mass flux)

$\dot{\sigma}$ = a source term that describes the amount of mass of fluid *generated* per unit volume per unit time.

Writing equation 1 in terms of physical quantities

$$\begin{aligned}&\text{Rate of mass accumulation per unit area}\\ &\qquad = \text{Rate of mass entering or exiting per unit area}\\ &\qquad\qquad + \text{Rate of mass being generated per unit area} \qquad (2)\end{aligned}$$

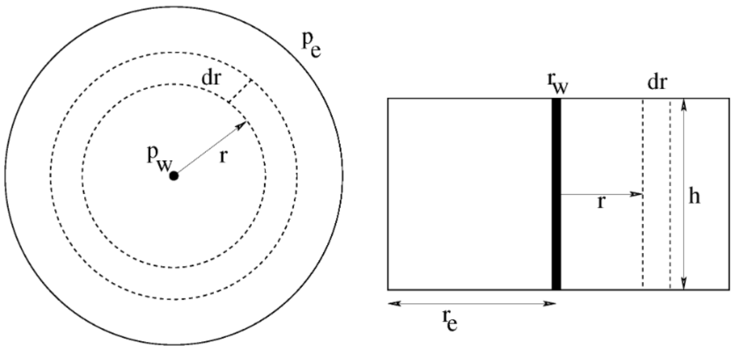

Figure 1: A simplified reservoir with a well in the center. left: top view, right: side view

In figure 1 the reservoir is assumed to be a cylinder with radius r_e and a height of h. A well for extraction of fluids from the reservoir is placed in the middle of the cylinder and has a radius of r_w. The reservoir is constituted of porous material which contains the fluids we are interested in extracting in the substrate gap. The well is a source or sink(mass is being *produced* in case of extraction or being *lost* during injection)

Since our reservoir is cylindrical in shape , we shall go about manipulating the each variable in equation 1 so we can use radial coordinate system instead of Cartesian coordinates to ease computation.

By assuming that the flow is radially symmetric, we can make the following transformations:

$$x \to r, \tag{3}$$

$$A(x) \to A(r) = 2\pi r h \tag{4}$$

Since no mass is being generated in the reservoir we can set the mass generation term $\dot{\sigma} = 0$ everywhere other than at the interface of well and reservoir, $r = r_w$. This is in contrast to systems where mass is being consumed or produced, like in nuclear processes.

Mass flux can be written as

$$J(x,t) = \text{mass flux} = \frac{\text{mass}}{\text{area} \cdot \text{time}} \tag{5}$$

Rewriting equation 5

$$\begin{aligned} J(r,t) &= \text{mass flux} = \frac{\text{mass}}{\text{volume}} \cdot \frac{\text{length}}{\text{time}} \\ J(r,t) &= \text{density} \cdot \text{velocity} \\ J(r,t) &= \rho \cdot u(r,t) \end{aligned} \tag{6}$$

Mass concentration of fluid, q can written as $\phi\rho$, where ϕ is the porosity of the medium. It should be noted that both the density, ρ, and the porosity, ϕ, can vary with pressure (and thus time, as pressure changes with time).

$$q = \phi\rho \tag{7}$$

Replacing A, q and J in equation 1 we get

$$\begin{aligned} 2\pi rh\frac{\partial(\phi\rho)}{\partial t} &= -\frac{\partial}{\partial r}(2\pi rh\rho u_r) \\ \frac{\partial(\phi\rho)}{\partial t} &= -\frac{1}{r}\frac{\partial}{\partial r}(r\rho u_r) \end{aligned} \tag{8}$$

For radically symmetric flow the fluid speed parameter can be written in terms of pressure difference in the following way, ref. [1]

$$u_r = -\frac{k}{\mu}\frac{\partial p}{\partial r} \tag{9}$$

where, k is the absolute permeability of the rock medium, μ is the fluid viscosity, and p is the fluid pressure. Assuming the fluid has a low compressibility and applying some domain knowledge *magic* equation 8 can be written as

$$\frac{\partial p}{\partial t} = \eta\frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial p}{\partial r}) \tag{10}$$

where, η is the hydraulic diffusivity, given by

$$\eta = \frac{k}{\mu\phi c_t} \tag{11}$$

$c_t = c_f + c_r$ is the total compressibility factor of the rock medium and the fluid trapped in it. c_f is the fluid compressibility factor and c_r the rock compressibility factor.

We perform another transformation from radial coordinates r to linear coordinates system y by taking

$$y = ln\frac{r}{r_w} \tag{12}$$

$$\implies r = r_w e^y \tag{13}$$

$$\frac{\partial}{\partial r} = \frac{\partial y}{\partial r}\frac{\partial}{\partial y} \tag{14}$$

$$= \frac{1}{r}\frac{\partial}{\partial y} \tag{15}$$

$$= \frac{1}{r_w e^y}\frac{\partial}{\partial y} \tag{16}$$

Thus, in the new coordinate system equation 10 transforms to

$$\frac{\partial p}{\partial t} = \eta \frac{e^{-2y}}{r_w^2} \frac{\partial^2 p}{\partial y^2} \tag{17}$$

This will be the governing equation for this assignment.

2.2 Boundary conditions

There are two boundaries of the system

1. At the boundary, the pressure is constant, and the boundary at the reservoir exterior wall $r = r_e$.

$$p|_{r=r_e} = p_{init} \tag{18}$$

   where p_{init} is the initial pressure of the well

2. The boundary at the well exterior wall where radius $r = r_w$. At this boundary production fluid is being taken out. Assuming the well is producing a constant volume of fluid per unit of time, from Darcy's equation 9 [2]

$$\begin{aligned} Q|_{r_w} = -A \cdot u|_{r_w} &= -2\pi r h \cdot -\frac{k}{\mu}\frac{\partial p}{\partial r}\bigg|_{r_w} \\ \implies \frac{\partial p}{\partial r} &= \frac{Q\mu}{2\pi r h k}\bigg|_{r_w} \\ \implies \frac{1}{r}\frac{\partial p}{\partial y} &= \frac{Q\mu}{2\pi r h k}\bigg|_{r_w} \\ \implies \frac{\partial p}{\partial y} &= \frac{Q\mu}{2\pi h k}\bigg|_{r_w} \\ \implies \frac{\partial p}{\partial y} &= \alpha|_{r_w}, \quad \alpha = \frac{Q\mu}{2\pi h k} \end{aligned} \tag{19}$$

2.3 Node diagram

In order to be consistent with node numbering we will use the diagram which was graciously provided in the assignment statement

3 Exercise 1: Steady-state solution

In this exercise we will model the steady state solution of the equation 17, our governing fluid flow equation. Steady state can be defined as the state of a system where the paramater we are interested in does not change with time. In this assignment we are interested in the pressure of the fluid in the reservoir. Thus to model the steady state solution of equation 17, we will set its LHS to 0

$$\begin{aligned} \eta \frac{e^{-2y}}{r_w^2}\frac{\partial^2 p}{\partial y^2} &= 0 \\ \implies \frac{\partial^2 p}{\partial y^2} &= 0 \end{aligned} \tag{20}$$

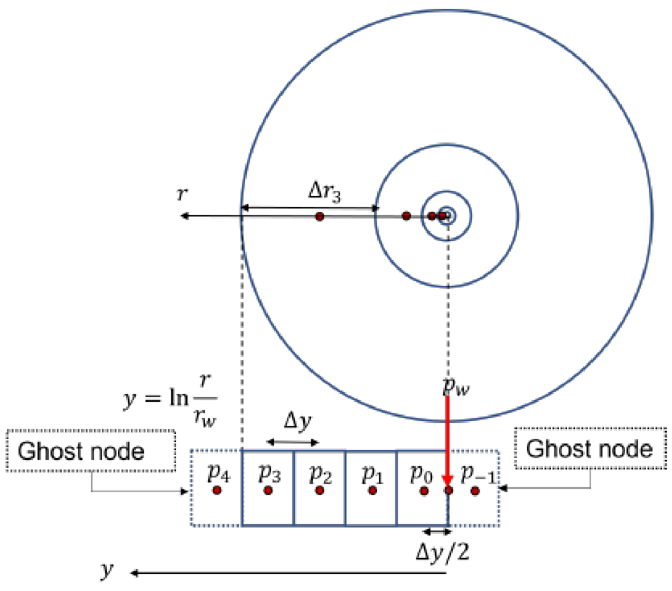

Figure 2: Sketch of coordinate transformation and node placement. top view of the well

$$\implies \frac{\partial p}{\partial y} = C_1 \tag{21}$$

$$\implies p = C_1 y + C_2 \tag{22}$$

3.1 Finding the analytical solution

Applying the boundary condition at the well, equation 19 to 21

$$C1 = \alpha = \frac{Q\mu}{2\pi hk} \tag{23}$$

Applying the boundary condition at the reservoir boundary, equation 18 to 22 we get

$$\begin{aligned} p_{init} &= C_1 y_e + C_2 \\ \implies C_2 &= p_{init} - \alpha y_e \end{aligned} \tag{24}$$

Rewriting the analytical solution 22 with values of C_1 and C_2 we get

$$\begin{aligned} p &= \alpha y + (p_{init} - \alpha y_e) \\ p &= \alpha(y - y_e) + p_{init} \end{aligned} \tag{25}$$

Thus the analytical solution to the flow problem can be written as equation 25

3.2 Lazy numerical model

For the steady state, equation 20 can be written as a numerical differential (using 80) at node i is

$$\frac{d^2p}{dy^2} = \frac{p_{i+1} - 2p_i + p_{i-1}}{\Delta y^2} = 0 \tag{26}$$

$$p_{i+1} - 2p_i + p_{i-1} = 0 \tag{27}$$

Applying equation 27 at each node $[p_0, p_1, p_2, p_3]$ we obtain the following set of equations

$$p_{-1} - 2p_0 + p_1 = 0 \tag{28}$$

$$p_0 - 2p_1 + p_2 = 0 \tag{29}$$

$$p_1 - 2p_2 + p_3 = 0 \tag{30}$$

$$p_2 - 2p_3 + p_4 = 0 \tag{31}$$

Applying the boundary condition 19 at node 3 (equation 31) and being lazy condition where $p_4 = p_{init}$

Applying the boundary condition 18 at node 0 (equation 28) and using central difference we get

$$\begin{aligned} \frac{p_0 - p_{-1}}{\Delta y} &= \alpha \\ p_{-1} &= p_0 - \alpha \Delta y \end{aligned} \tag{32}$$

Substituting values of p_{-1} and p_4 into equation 28 and 31 we get

$$\begin{aligned} -p_0 + p_1 &= \alpha \Delta y \\ p_0 - 2p_1 + p_2 &= 0 \\ p_1 - 2p_2 + p_3 &= 0 \\ p_2 - 2p_3 &= -p_{init} \end{aligned} \tag{33}$$

Writing 33 in matrix form we get

$$\begin{pmatrix} -1 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{pmatrix} \cdot \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix} = \begin{pmatrix} \alpha \Delta y \\ 0 \\ 0 \\ -p_{init} \end{pmatrix} \tag{34}$$

As we were rather casual with enforcement of the boundary conditions this is the lazy solution.

3.3 Truncation error for the finite difference approximation

The truncation error for the finite difference approximation at interior grid points is mentioned in the book [3]

Based on Taylor's formula, it can be re-written as

$$\frac{d^2p}{dy^2} = \frac{p_{i+1} + p_{i-1} - 2p_i}{\Delta y^2} = \frac{2}{4!}p^{(4)}(\eta_i)\Delta y^2 \tag{35}$$

As we calculate the derivative of the analytical solution,

$$p(y) = p_{init} + \alpha(y - y_e) \tag{36}$$

$$\frac{dp}{dy} = \alpha \tag{37}$$

$$\frac{d^2p}{dy^2} = 0 \tag{38}$$

$$p^3(y) = 0, p^4(y) = 0 \ldots \tag{39}$$

It clearly shows that the analytical solution is a polynomial order of 1. Therefore, the truncation error will be zero at interior grid points. The only source of error is our lazy implementation of the boundary condition at the last node. Its an error in linear interpolation of the pressure at the **exact** well boundary. So the error for lazy solution should be of the $\mathcal{O}(h)$.

3.4 Theoretical investigation of the lazy solution and not so lazy solution

3.4.1 Theoretical error in numerical approximation

We have used equation 80 for the formulation of my numerical grid and used equation 75 for enforcing our boundary condition. So theoretical numerical order should be of the order $\mathcal{O}(h)$ at each node. when using the 'lazy' approximation, the boundary condition is $p_N = p_e = p_{approximation}$, therefore, the error is basically $p_{true} - p_{approximation}$

$$p(y_e + \Delta y/2) - p(y_e) = p'(\eta)\frac{\Delta y}{2}$$

Therefore the order of numerical error for the accumulated global error is one and it will scale as Δy

3.4.2 Not so lazy solution

Equation 20 holds true for all y including for $y = 1$ which is at the reservoir boundary.

We can derive the formulation

$$p_N = p(x + h) = p((x + h/2) + h/2), \tag{40}$$

$$= p_{N+1/2} + \left.\frac{dp}{dy}\right|_{y+h/2} + \mathcal{O}(h^2), \tag{41}$$

$$= p_e + \frac{p_N - p_{N-1}}{h}\frac{h}{2} + \mathcal{O}(h^2), \tag{42}$$

$$p_N = 2p_e - p_{N-1} + \mathcal{O}(h^2) \tag{43}$$

Note: with this enforcement of boundary condition, error is now $\mathcal{O}(h^2)$. So this finite difference solution should match the analytical solution irrespective of number of nodes.

3.4.3 Matrix formulation for $N = 4$

For this not-so-lazy version of boundary condition is:

$$p_N = 2p_e - p_{N-1}$$

As $N = 4$, we replace

$$p_4 = 2p_e - p_3$$

Therefore, the matrix equation can be modified and re-written as

$$\begin{aligned} -p_0 + p_1 &= \alpha\Delta y \\ p_0 - 2p_1 + p_2 &= 0 \\ p_1 - 2p_2 + p_3 &= 0 \\ p_2 - 3p_3 &= -2p_e \end{aligned}$$

Then it can be written as

$$\begin{pmatrix} -1 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -3 \end{pmatrix} \cdot \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix} = \begin{pmatrix} \alpha\Delta y \\ 0 \\ 0 \\ -2p_e \end{pmatrix} \tag{44}$$

3.5 Simulation of steady-state solution

3.5.1 Steady state simulator

The Python code used for the evaluation of the steady-state solution is presented in the notebook. In this implementation, both *the lazy solution* (matrix 34) and the *not so lazy* (matrix 44) are included. Since we are mostly interested in the implementation of the code and not the exact physical parameters we set α and p_{init} in equations 34 and 44 to be 1.

Figure 3 shows the steady-state solution results for both the lazy and non-lazy formulations. The *not so lazy* solution of problem produces exact results with no error irrespective of the node sizes, whereas the lazy solution, because of faulty implementation of the boundary conditions produces results with errors. As we increase the number of nodes, the lazy approximation is compelled to approach the real model.

3.5.2 Fixing the node

For computing the error at a fixed y we apply a cleaver trick. We divide the whole reservoir into odd number of grids and we always compute the error at the middle grid point. For example, we take, $N = 5$ and then compute error at the 2^{nd} grid point. For $N = 7$ we take compute the error at the 3^{rd} grid point and so on. Note the grid numbering starts with 0.

3.5.3 Error scaling

From out theoretical investigation we expected that the error will be of the order $\mathcal{O}(h)$. The figure 4 shows that the error at any given node increases linearly with the size of each node which is global error in the first order. This confirms our theoretical estimation of the error where local error is in the second order and global error is accumulated and summed up in the first order.

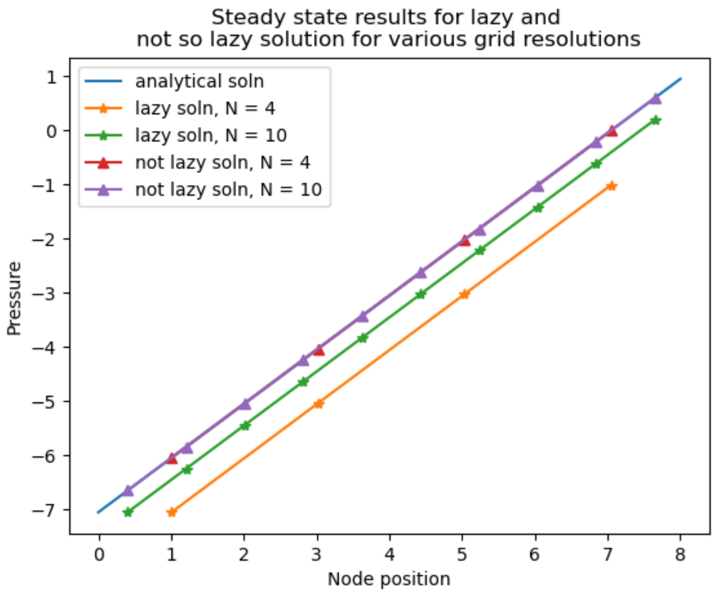

Figure 3: Steady state results for lazy and not so lazy solution for various resolutions

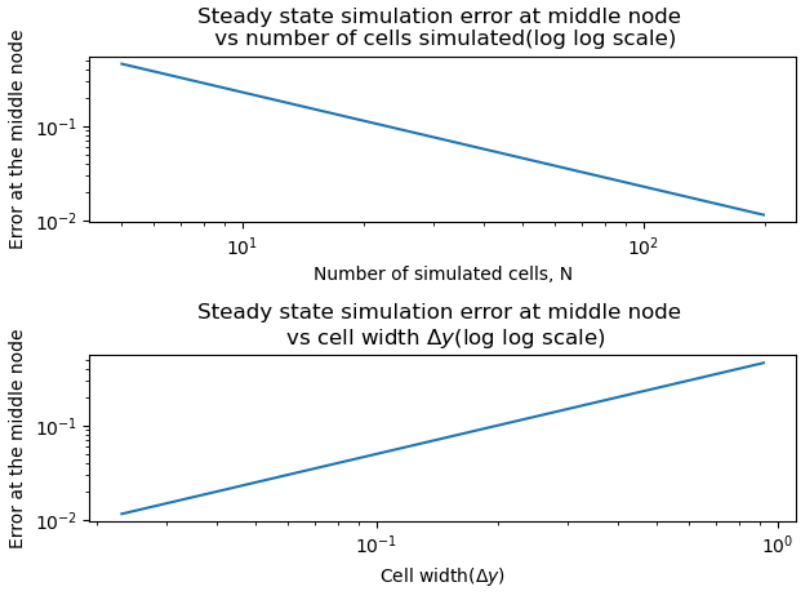

Figure 4: Error in stead state solution (lazy method)

4 Exercise 2: Time-Dependent solution

In this section we will implement the time dependent solution to the reservoir flow problem.

To model how the pressure across the reservoir changes with time we use the implicit scheme for time discretization. The implicit scheme for an interior point $0 \leq i < N$ is given by

$$\frac{p_i^{n+1} - p_i^n}{\Delta t} = \eta \cdot \frac{e^{-2y_i}}{r_w^2} \cdot \frac{p_{i+1}^{n+1} - 2p_i^{n+1} + p_{i-1}^{n+1}}{\Delta y^2} \tag{45}$$

4.1 Special case $N = 4$

Rearranging equation 45 :

$$p_i^{n+1} - p_i^n = \frac{\eta e^{-2y_i} \Delta t}{r_w^2 \Delta y^2} \cdot (p_{i+1}^{n+1} - 2p_i^{n+1} + p_{i-1}^{n+1}) \tag{46}$$

$$p_i^{n+1} - p_i^n = \xi_i \cdot (p_{i+1}^{n+1} - 2p_i^{n+1} + p_{i-1}^{n+1}) \tag{47}$$

$$\text{where,} \quad \xi_i = \frac{\eta e^{-2y_i} \Delta t}{r_w^2 \Delta y^2} \tag{48}$$

Equation 47 can be further reformulated by separating pressure terms at time step n and $n+1$

$$-\xi_i \cdot p_{i+1}^{n+1} + (1 + 2\xi_i) \cdot p_i^{n+1} - \xi_i \cdot p_{i-1}^{n+1} = p_i^n \tag{49}$$

Equation 49 is valid for all spacial grid indices i and time grid indices n. For the special case of $N = 4$ equation 49 can be written as the following series of equations for $0 \leq i < N$.

$$-\xi_0 \cdot p_{-1}^{n+1} + (1 + 2\xi_0) \cdot p_0^{n+1} - \xi_0 \cdot p_1^{n+1} = p_0^n \tag{50}$$

$$-\xi_1 \cdot p_0^{n+1} + (1 + 2\xi_1) \cdot p_1^{n+1} - \xi_1 \cdot p_2^{n+1} = p_1^n \tag{51}$$

$$-\xi_2 \cdot p_1^{n+1} + (1 + 2\xi_2) \cdot p_2^{n+1} - \xi_2 \cdot p_3^{n+1} = p_2^n \tag{52}$$

$$-\xi_3 \cdot p_2^{n+1} + (1 + 2\xi_3) \cdot p_3^{n+1} - \xi_3 \cdot p_4^{n+1} = p_3^n \tag{53}$$

We can then apply boundary conditions 18 and 19. For the boundary at the well $y = r_w$,

$$p_{-1}^{n+1} = p_0^{n+1} - \alpha \Delta y \tag{54}$$

$$\equiv p_{-1}^{n+1} = p_0^{n+1} - \beta \tag{55}$$

$$\text{where,} \quad \beta = \alpha \Delta y \tag{56}$$

Similarly for the boundary condition at $y = r_e$, modifying equation 43

$$p_4^{n+1} = 2p_e - p_3^{n+1} \tag{57}$$

Substituting for p_{-1}^{n+1} and p_4^{n+1} from equaration 55 and 57 into equation 50 and 53 respectively we get

$$(1 + \xi_0) \cdot p_0^{n+1} - \xi_0 \cdot p_1^{n+1} = p_0^n - \beta\xi_0 \tag{58}$$

$$-\xi_1 \cdot p_0^{n+1} + (1 + 2\xi_1) \cdot p_1^{n+1} - \xi_1 \cdot p_2^{n+1} = p_1^n \tag{59}$$

$$-\xi_2 \cdot p_1^{n+1} + (1 + 2\xi_2) \cdot p_2^{n+1} - \xi_2 \cdot p_3^{n+1} = p_2^n \tag{60}$$

$$-\xi_3 \cdot p_2^{n+1} + (1 + 3\xi_3) \cdot p_3^{n+1} = p_3^n + 2\xi_3 p_e \tag{61}$$

Writing the above equations in matrix form we get

$$\begin{pmatrix} 1+\xi_0 & -\xi_0 & 0 & 0 \\ -\xi_1 & 1+2\xi_1 & -\xi_1 & 0 \\ 0 & -\xi_2 & 1+2\xi_2 & -\xi_2 \\ 0 & 0 & -\xi_3 & 1+3\xi_3 \end{pmatrix} \cdot \begin{pmatrix} p_0^{n+1} \\ p_1^{n+1} \\ p_2^{n+1} \\ p_3^{n+1} \end{pmatrix} = \begin{pmatrix} p_0^n \\ p_1^n \\ p_2^n \\ p_3^n \end{pmatrix} + \begin{pmatrix} -\beta\xi_0 \\ 0 \\ 0 \\ 2p_e\xi_3 \end{pmatrix} \tag{62}$$

4.2 Implementation of the time dependant problem and A matrix for $N = 4$

The code for the implementation of the time-dependent solution is presented in the notebook.

The code gives the following output for the A matrix for 4 nodes and a time delta of 0.01 days, which matches the solution provided by the solution provided.

$$\begin{pmatrix} 5.28702460e+03 & 5.28602460e+03 & 0.00000000e+00 & 0.00000000e+00 \\ 9.42633218e+01 & 1.89526644e+02 & 9.42633218e+01 & 0.00000000e+00 \\ 0.00000000e+00 & 1.68095582e+00 & 4.36191165e+00 & 1.68095582e+00 \\ 0.00000000e+00 & 0.00000000e+00 & 2.99757363e02 & 1.08992721e+00 \end{pmatrix}$$

5 Exercise 3: Accuracy and performance of time-dependent solution (Bonus)

5.1 Line-source implementation

Figure 5 shows the comparison of line source solution with time domain simulation for default values of physical parameters. Figure 6 shows the comparison of line source solution with time domain simulation using real values of a reservoir(this is calculated in the next section when we fit well data).

The figures show that the time domain simulation has a good match the Line Source solution[4]. The pressure wave can be seen moving further away from the well block with the passage of time. It is interesting to note that the modelled solution matches with the line source solution transient period(i.e. before the pressure wave hits the reservoir boundary). Once the steady state solution is reached the line source solution and the numeric solutions deviate.

5.2 Computational expense

The code was updated to include option to either of the following solvers for $Ax = b$

1. Dense, using numpy.linalg.solve.
2. Sparse using, scipy.sparse.linalg.spsolve.
3. Thomas algorithm with *numba.jit accelerator*

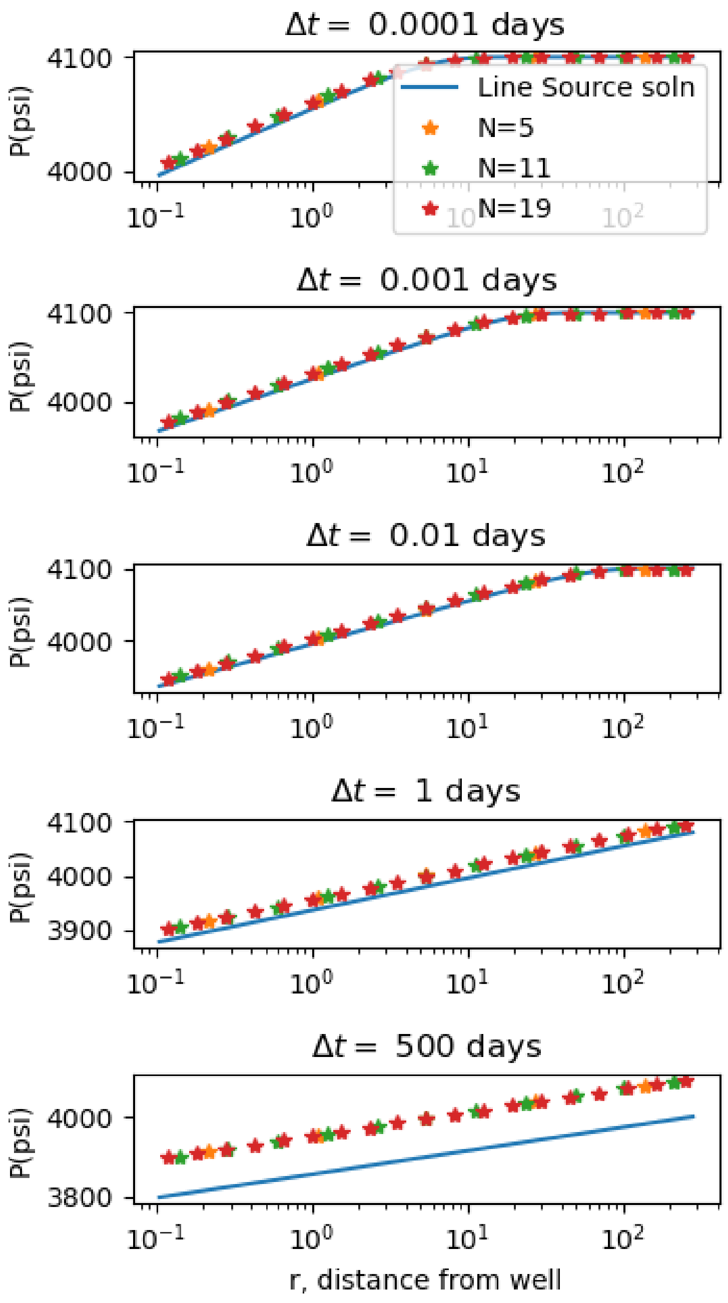

Figure 5: Comparision of line source solution with time domain sim for default values presented

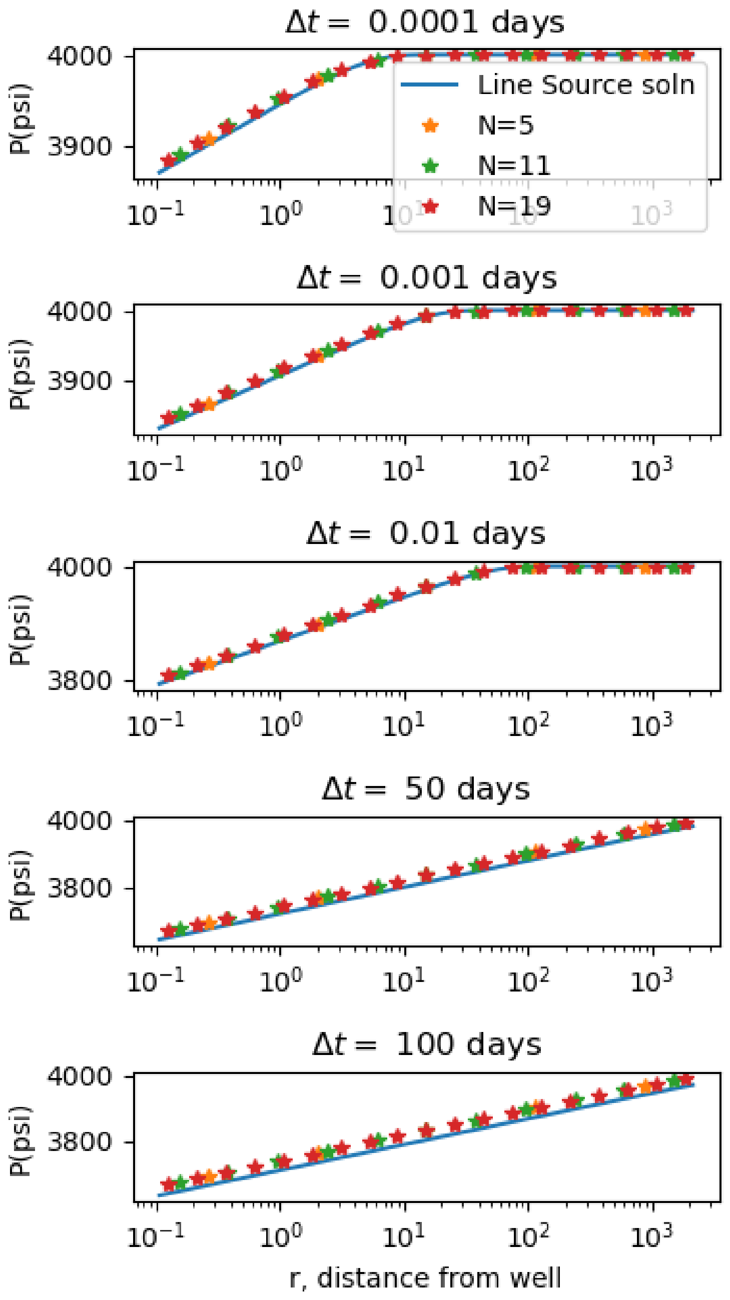

Figure 6: Comparision of line source solution with time domain sim for well BHP calculated later

Node	numpy	scipy	Thomas algorithm
10	20.4μ s	178μ s	23.3 μs
100	83μ s	310μ s	47.8μs
1000	23.6 ms	10.1 ms	3.61 ms
5000	1.86 s	0.97 s	1.03s
10,000	8 s	3.4 s	4.1s
20,000	-	14.9s	15.5s
100,000	-	-	-

Table 1: Execution time of various solvers

Each of the solvers was used in the same model of N nodes and the execution time we recorded and presented in table 1

The **numpy solver** is quickest for smaller size matrices and gets increasingly slower compared to the other two. The **scipy solver** shines when the matrices being solved are large. When solving smaller matrices its slow because of computational overheads.

The **Thomas algorithm**(especially when used with **numba**) works surprisingly fast over both when handling small and large matrices.

6 Exercise4: Match model to well test data

So far we have calculated how the fluid pressure varies inside the reservior, both at steady-state and time-dependent solution for incremental change in time. In this exercise we will check how well our model functions when compared with real-life data from a well

6.1 Pressure inside the well

From the exercises above we are able to calculate the pressure in the reservoir at N number of nodes, node 0 being closest to the well and node $N-1$ being closest to the outer reservior boundary.

Taking the pressure at the well to be p_w, the pressure p_0 at node 0 can be written in the form

$$p_0 = p_w + \frac{\Delta y}{2}\frac{\partial p}{\partial y}|_{r_w} + \mathcal{O}(\Delta y^2) \tag{63}$$

Note: the distance between node 0 and the well is $\Delta y/2$. From the equation 19 we know that p'_w at the well location as given that

$$\frac{\partial p}{\partial y}(y = y_w) = \frac{Q\mu}{2\pi hk} \tag{64}$$

Therefore, Substituting it in the above equation we get

$$p_0 = p_w + \frac{Q\mu}{2\pi hk}\Delta\frac{y}{2} + \mathcal{O}(\Delta y^2) \tag{65}$$

$$p_w = p_0 - \frac{Q\mu\Delta y}{4\pi hk} - \mathcal{O}(\Delta y^2) \tag{66}$$

The implementation of the well pressure is provided in the class implementation.

6.2 Well test data

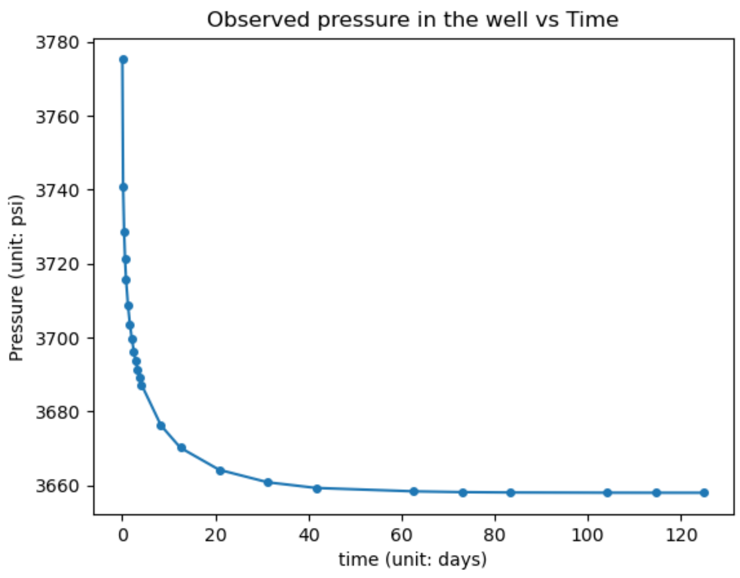

Figure 7: Observed pressure in the well vs time

Figure 7 shows the data provided in the data file along with this assignment. Please note that the units in the data file are assumed to be *real world units* (psi and days).

6.3 Fitting numerical model to real-world well data

In this section we will fit our numeric model to real-life well data by varying some of the input parameters. There are several assumptions we make in this process.

Assumptions:

- the units of time axis in the data file is in hours
- the units of the pressure axis is psi

Our model is then fitted to the well bhp data by changing parameters absolute permeability k, initial reservoir pressure p_i, and outer reservoir boundary radius r_e.

Fitted parameters for our numeric model with $N = 100$ cells:
$r_e = 7805.20ft$, $k = 378.498mD$ and $pinit = 4000.88psi$.

Making a model with the above mentioned reservoir parameters and plotting it against the well BHP data and line source solution we get the following graph (figure 8)

The theory for the line solution is not presented in this submission, but has been implemented in the notebook submitted. The line source solution is calculated using the simulation time provided in the well BHP data and setting $r = r_w$

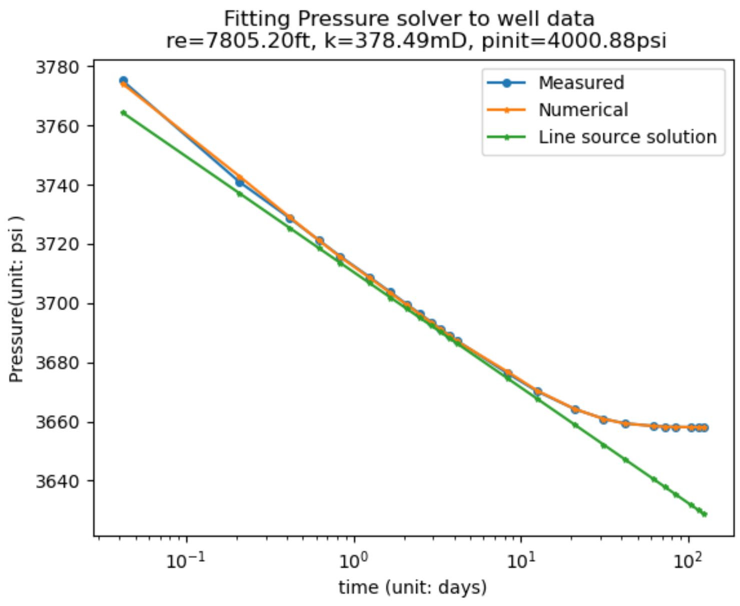

Figure 8: Fitting well data

6.4 Volume of water

We have now estimated the radius of the external boundary of the reservoir r_e
Radius of the well, r_e = 2379.02m

Height of the well, h = 3.35m

Porosity of the well, ϕ = 0.25

Volume of reservoir, $V_r = \pi r_e^2 h = 59614925.0m^3$

Volume of water in the reservoir $V_w = \phi V_r$ = 14903731.3m^3

7 Conclusion

In this project, we have translated our theoretical understanding of discretization and finite difference methods into practical applications for solving real-world problems. The foundation of our approach lies in the continuity equation, which we learned in class. This equation serves as the basis for describing mass and energy conservation (radial diffusivity equation utilized in this study)

Our initial step was to theoretically derive discretized equations, and subsequently, we implemented and solve these linear equations in Python code with both lazy and not-so-lazy boundary conditions under steady-state. During our investigation, we observed that the lazy boundary condition introduces the error that scales linearly with the size of the node, as shown in the generated plot. This error is a global error, consistent with our theoretical understanding, where local errors scale at a second-order, while global errors accumulate and are summed up at a first-order scale. However, the not-so-lazy solution can be compelled to approach to true solution by increasing the grid node parameter N at a higher computational cost.

In our investigation of time efficiency for solving with sparse versus dense matrices, we discovered that as the number of nodes increases in the computation, the scipy library's sparse matrices outperform both NumPy's and the Thomas algorithm in terms of speed. On the other hand, NumPy is highly efficient and well-suited for smaller matrix sizes. Surprisingly, the Thomas algorithm, particularly when used with Numba, demonstrates remarkable speed and efficiency when handling both small and large matrices.

At last, our numerical model is fit for real-life well data. The auto curve-fitting technique also allowed us to estimate the total volume of water.

8 Reflection

Partha:

This problem was a good mixture of domain I had some knowledge on (fluids under pressure) and domain I am not familiar with (flow through porous medium). It was super fun (through countless frustrations) to implement this and i learned quiet a bit.

Theory: Clarified why we can get away with setting $p = p_init$ everywhere. The source of errors in numeric models became a lot clearer, solving this project. Dont feel like i fully understand line source solution. My current theory is that it *simulates* how the 'pressure wave' travels through a infinite well that never reaches a steady state. That is why it matches our time sim to the reservoir boundary and then deviates (travelling further). Which would imply the reservoir is being squeezed at a constant pressure(p_init) and is reducing in volume at steady state, as fluid is being extracted.

Coding: Learned a lot about classes and I have a better feel of what the structure of a class should be like. Could be fun to implement well pressure that varies with well height. I think our model doesn't match the well bhp data exactly because its not a cylinder.

Overall this project was a rich learning experience. I think the problem statement provided for this project does a very good job in guiding me through a complicated problem to its solution. Being able to visualize the entirety of the problem and then breaking it down helped immensely. I think the notes pointing our specific common errors we can do while implementing the solutions are very cleaver.

Jing: I must express my sincere appreciation for the highly interactive nature of this small class. It has proven to be the most engaging class I've participated in so far. The instructor's patience in providing comprehensive explanations and deriving theoretical equations, such as ODEs, has been truly invaluable. The willingness to offer clarification repeatedly has benefited my understanding.

I particularly have the benefit of the step-by-step code implementation during the class, ranging from simple to more complex scenarios. This approach has provided me the opportunity to enhance my understanding and coding implementation skills progressively throughout the course, making me well-prepared for the project ahead. While the case study may differ somehow, I've successfully applied the knowledge and skills learned during class and made class implementation for various scenarios. However, balancing the demands of the project along with a few other assignment deadlines during the same period can be quite a challenge. Therefore, having the project assigned earlier would be highly appreciated and greatly benefit me and allow me to have more flexible time for the project, resulting in a more rewarding learning experience.

Hodjdat: The problem presented a valuable opportunity for me as a reservoir engineer, prompting me to revisit key references on well testing while addressing the issue at hand. My approach centered on a more practical methodology to align the mathematical model with the observed data, as opposed to utilizing the curve-fitting module suggested by Scipy.

The data was segmented into two distinct sections, each characterized by different flow regimes. The first segment, corresponding to infinite-acting behavior, was examined to deduce the aquifer's (kh) based on the slope of the semi-log plot. Furthermore, the pressure at the one-hour mark (P1) was leveraged to estimate the initial reservoir pressure (Pi). In the second section, corresponding to pseudo-steady-state conditions, the slope of the curve, in conjunction with the permeability determined from the infinite-acting section, was employed to ascertain the reservoir's radius.

Notably, the observed data was sourced from a reservoir simulator, eliminating complexities such as wellbore storage and skin effects. Consequently, my approach yielded results that closely aligned with methodology submitted by other team members.

In addition to addressing the immediate task, this experience significantly enhanced my coding skills, including the understanding of class inheritance, diverse curve-fitting methodologies, alternative visualization techniques, and performance analysis. It is worth noting that the problem statement was comprehensive and instructive. While the problem's intent was to provide a platform for knowledge acquisition and experience in numerical modeling, it is pertinent to acknowledge that other types of well tests, such as build-up tests, may be more suitable for determining initial reservoir pressure.

In summary, I thoroughly enjoyed the process of learning and working with this problem and collaborating with the team.

References

[1] H. Darcy, “Les fontaines publiques de la ville de Dijon: exposition et application des principes suivre et des formules employer dans les questions de distribution d'eau,” *Victor dalmont*, vol. 1, 1856.

[2] M. K. Hubbert, “Darcy's Law and the Field Equations of the Flow of Underground Fluids,” *Transactions of the AIME*, vol. 207, pp. 222–239, Dec. 1956.

[3] A. Hiorth, *Computational Engineering and Modeling.*

[4] H. J. Ramey, “Application of the Line Source Solution To Flow in Porous Media–A Review,” in *All Days*, (Dallas, Texas), pp. SPE–1361–MS, SPE, Feb. 1966.

A Taylors expression

Taylors equation: given a function $f(x)$ which is continuous and defined at a distance h from x, $f(x+h)$ can be written as [3]

$$f(x+h) = f(x) + \frac{1}{1!}f'(x)h + \frac{1}{2!}f''(x)h^2 + \frac{1}{3!}f'''(x)h^3 + \cdots \tag{67}$$

Generalising the above equation 67

$$f(x+h) = f(x) + \sum_{n=1}^{\infty} \frac{1}{n!}f^n(x)h^n \tag{68}$$

In numerical modelling we usually truncate the tylor expression after a certain number of terms. Thus, expanding equation 68 upto n terms can also be written as

$$f(x+h) = f(x) + \frac{1}{1!}f'(x)h + \cdots + \frac{1}{(n-1)!}f^{n-1}(x)h^{n-1} + \mathcal{O}(h^n) \tag{69}$$

where $\mathcal{O}(h^n)$ is the truncation error.

Ignoring the error term for now and expanding equation 69 to the first derivative

$$\begin{aligned} f(x+h) &= f(x) + \frac{1}{1!}f'(x)h + \mathcal{O}(h^2) \\ \implies f'(x) &= \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h) \end{aligned} \tag{70}$$

$$f_i' = \frac{f_{i+1} - f_i}{h} + \mathcal{O}(h) \tag{71}$$

Equation 70 is the first derivative of $f(x)$ by forward difference. Similarly reverse difference can be calculated as

$$f(x-h) = f(x) - \frac{1}{1!}f'(x)h + \mathcal{O}(h^2) \tag{72}$$

$$f'(x) = \frac{f(x) - f(x-h)}{h} + \mathcal{O}(h) \tag{73}$$

$$f_i' = \frac{f_i - f_{i-1}}{h} + \mathcal{O}(h) \tag{74}$$

Similarly the derivative of a function can also be calculated using central difference

$$f_i' = \frac{f_{i+1} - f_{i-1}}{2h} + \mathcal{O}(h) \tag{75}$$

From equation 69 taking $dy \equiv \pm h$ and expanding to the second derivative term we get

$$f(x+h) = f(x) + \frac{1}{1!}f'(x)h + \frac{1}{2!}f''(x)h^2 + \frac{1}{3!}f'''(x)h^3 + \mathcal{O}(h^4) \tag{76}$$

$$f(x-h) = f(x) - \frac{1}{1!}f'(x)h + \frac{1}{2!}f''(x)h^2 - \frac{1}{3!}f'''(x)h^3 + \mathcal{O}(h^4) \tag{77}$$

adding equations 76 and 77 we get

$$f(x+h)+f(x-h) = 2f(x)+f''(x)h^2+\mathcal{O}(h^4) \tag{78}$$

$$f''(x) = \frac{f(x+h)-2f(x)+f(x-h)}{h^2} - \mathcal{O}(h^2) \tag{79}$$

$$\equiv f_i'' = \frac{f_{i+1}-2f_i+f_{i-i}}{h^2} - \mathcal{O}(h^2) \tag{80}$$

Equations 71, 74, 75, and 80 form the mathematical basis for calculation of first and second derivatives and gradients numerically. These are arrived at using truncation of the taylor series and thus have associated errors of $\mathcal{O}(h)$ and $\mathcal{O}(h^2)$.