MOD 510 Project 2: Can you hear the size of a reservoir?

Parthasarathi Jena Jing Hou Hodjat

October 9th, 2023

1 Introduction

2 Problem defination and fomulation

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

$$\begin{aligned}\text{Rate of mass accumulation} = &\text{ Rate of mass entering or exiting} \\ &+ \text{Rate of mass being generated}\end{aligned} \tag{2}$$

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:

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

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

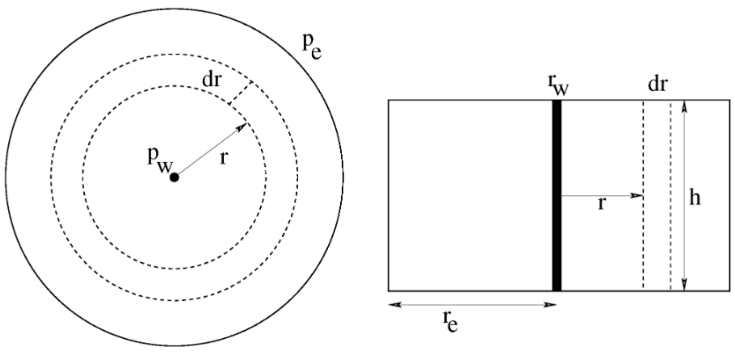

Figure 1: A simplified reservoir with a well in the center. left: top view, right: side view

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.

i am not really convinced this is true.

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]

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

where, k is the absolute permeability of the rock medium, μ is the fluid viscosity,

Remember to insert ref to Darcey's paper 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

This transformation is probably done as the pressure changes exponentially with distance from well head

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

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

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

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

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

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. the boundary at the reservoir exterior wall $r = r_e$. At this boundary the pressure is constant.

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

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

2. the boundary at the well exterior wall $r = r_w$. At this boundary production fluid is being taken out. Assuming the well is producing a constant volume of fluid per unit time, from Darcys insert ref for Darcey equation

9

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

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

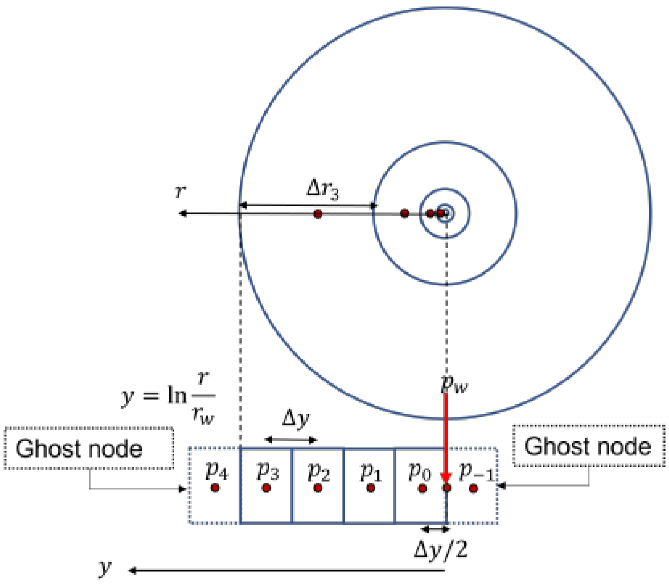

Figure 2: Sketch of coordinate transformation and node placement. top view of the well

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

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

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

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

3.1 Finding the analytical solution

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

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

Applying the boundary condition at the reservoir boundary, 18, to equation 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 79) at node i is

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

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

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 we set $p_4 = p_{init}$

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

$$\begin{aligned} \frac{p_1 - p_{-1}}{2\Delta y} &= \alpha \\ p_{-1} &= p_1 - 2\alpha\Delta y \end{aligned} \tag{32}$$

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

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

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

BLAH BLAH BLAH

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

3.4.1 Theoretical error in numerical approximation

We have used eqation 79 for formulation of my numerical grid and used equation 74 for enforcing our boundary condition. So theoretical numerical order should be of the order $\mathcal{O}(h)$ at each node.

3.4.2 Not so lazy solution

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

$$\left.\frac{\partial^2 p}{\partial y^2}\right|_{y=1} = 0 \tag{35}$$

$$\left.\frac{f_{i+1} - 2f_i + f_{i-i}}{h^2}\right|_{y=1} - \mathcal{O}(h^2) = 0 \tag{36}$$

$$\frac{p_N - 2p_e + p_{N-1}}{(h/2)^2} - \mathcal{O}(h^2) = 0 \tag{37}$$

$$p_N - 2p_e + p_{N-1} = \mathcal{O}(h^4) \tag{38}$$

$$p_N = 2p_e - p_{N-1} + \mathcal{O}(h^4) \tag{39}$$

$$\tag{40}$$

We can arrive at the same formulation

$$p_N = p(x+h) = p((x+h/2)+h/2), \tag{41}$$

$$= p_{N+1/2} + \left.\frac{dp}{dy}\right|_{y+h/2} + \mathcal{O}(h^2), \tag{42}$$

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

$$p_N = 2p_e - p_{N-1} + \mathcal{O}(h^2) \tag{44}$$

3.4.3 Matrix formulation for $N = 4$

Writing 33 to include 44 in matrix form we get

$$\begin{pmatrix} -1 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -3 \end{pmatrix} \cdot \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix} = \begin{pmatrix} \alpha\Delta y \\ 0 \\ 0 \\ -2p_{init} \end{pmatrix} \tag{45}$$

3.5 Simulation of steady state solution

3.5.1 Steady state simulator

The python code used for evaluation of the steady state solution is presented in Appendix B. In this code implementation, both *the lazy solution* (matrix 34) and the *not so lazy* (matrix 45) are coded. 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 45 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 irrespective of the node sizes. The lazy solution, because of faulty implementation of the boundary conditions produces results with errors, and the error reduces with increase in grid elements modelled.

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

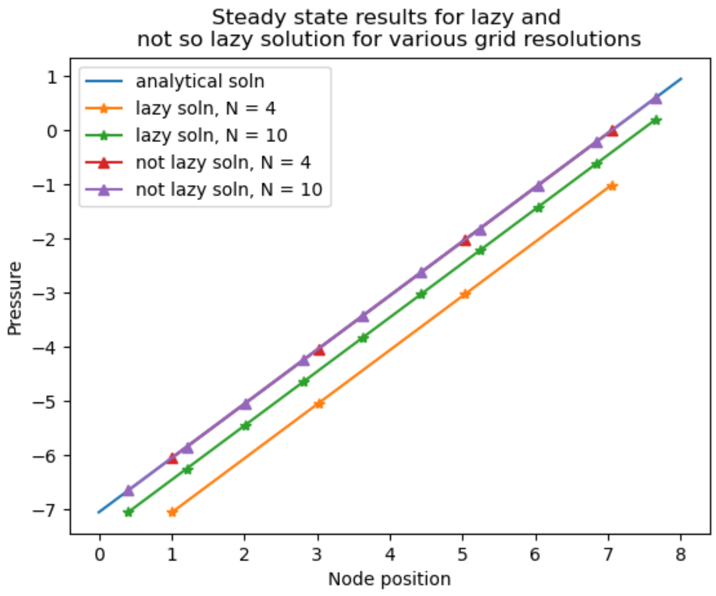

Figure 3: Steady state results for lazy and not so lazy solution for various grid resolutions

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.

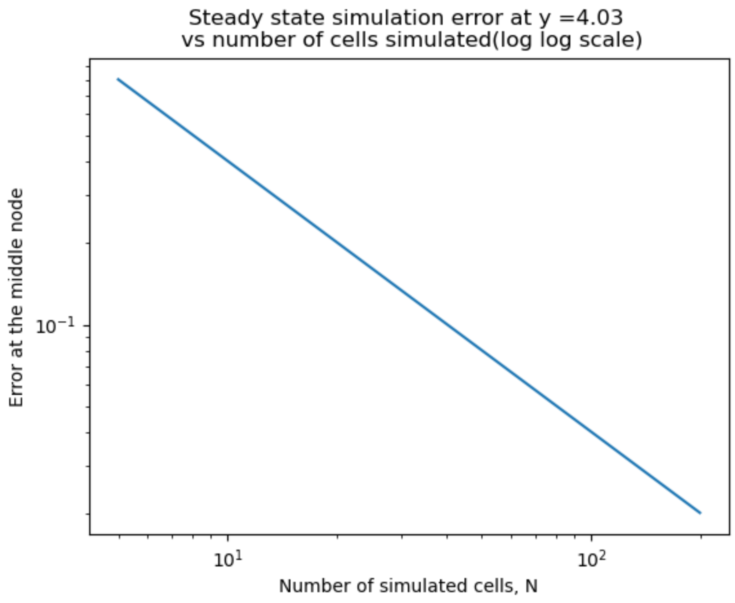

Figure 4: Error in stead state solution (lazy method)

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. This confirms our theoretical estimation of the error.

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 with time we use the implicit scheme for time discretization. The implicit scheme for an interior point $0 \leq i < N$ is given by

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

4.1 Special case $N = 4$

Rearranging equation 46 :

$$p_i^{n+1} - p_i^n = \frac{\eta e^{-2y_i}\Delta t}{r_w^2 \Delta y^2} \cdot (p_{i+1}^{n+1} - 2p_i^{n+1} + p_{i-1}^{n+1}) \tag{47}$$

$$p_i^{n+1} - p_i^n = \xi_i \cdot (p_{i+1}^{n+1} - 2p_i^{n+1} + p_{i-1}^{n+1}) \tag{48}$$

$$\text{where,} \quad \xi_i = \frac{\eta e^{-2y_i}\Delta t}{r_w^2 \Delta y^2} \tag{49}$$

Equation 48 can be further reformulated by separating pressure terms at time step n and $n + 1$

$$-\xi_i \cdot p_{i+1}^{n+1} + (1 + 2\xi_i) \cdot p_i^{n+1} - \xi_i \cdot p_{i-1}^{n+1} = p_i^n \tag{50}$$

Equation 50 is valid for all spacial grid indices i and time grid indices n. For the special case of $N = 4$ equation 50 can be written as the following series of equations for $0 \leq i < N$.

$$-\xi_0 \cdot p_{-1}^{n+1} + (1 + 2\xi_0) \cdot p_0^{n+1} - \xi_0 \cdot p_1^{n+1} = p_0^n \tag{51}$$

$$-\xi_1 \cdot p_0^{n+1} + (1 + 2\xi_1) \cdot p_1^{n+1} - \xi_1 \cdot p_2^{n+1} = p_1^n \tag{52}$$

$$-\xi_2 \cdot p_1^{n+1} + (1 + 2\xi_2) \cdot p_2^{n+1} - \xi_2 \cdot p_3^{n+1} = p_2^n \tag{53}$$

$$-\xi_3 \cdot p_2^{n+1} + (1 + 2\xi_3) \cdot p_3^{n+1} - \xi_3 \cdot p_4^{n+1} = p_3^n \tag{54}$$

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

$$p_{-1}^{n+1} = p_0^{n+1} - \alpha\Delta y \tag{55}$$

$$\equiv p_{-1}^{n+1} = p_0^{n+1} - \beta \tag{56}$$

$$\text{where,} \quad \beta = \alpha\Delta y \tag{57}$$

Similarly for the boundary condition at $y = r_e$, modifying equation 44

$$p_4^{n+1} = 2p_e - p_3^{n+1} \tag{58}$$

Substituting for p_{-1}^{n+1} and p_4^{n+1} from equaration 56 and 58 into equation 51 and 54 respectively we get

$$(1+\xi_0)\cdot p_0^{n+1} - \xi_0 \cdot p_1^{n+1} = p_0^n - \beta\xi_0 \tag{59}$$

$$-\xi_1 \cdot p_0^{n+1} + (1+2\xi_1)\cdot p_1^{n+1} - \xi_1 \cdot p_2^{n+1} = p_1^n \tag{60}$$

$$-\xi_2 \cdot p_1^{n+1} + (1+2\xi_2)\cdot p_2^{n+1} - \xi_2 \cdot p_3^{n+1} = p_2^n \tag{61}$$

$$-\xi_3 \cdot p_2^{n+1} + (1+3\xi_3)\cdot p_3^{n+1} = p_3^n + 2\xi_3 p_e \tag{62}$$

Writing the above equations in matrix form we get

$$\begin{pmatrix} 1+\xi_0 & -\xi_0 & 0 & 0 \\ -\xi_1 & 1+2\xi_1 & -\xi_1 & 0 \\ 0 & -\xi_2 & 1+2\xi_2 & -\xi_2 \\ 0 & 0 & -\xi_3 & 1+3\xi_3 \end{pmatrix} \cdot \begin{pmatrix} p_0^{n+1} \\ p_1^{n+1} \\ p_2^{n+1} \\ p_3^{n+1} \end{pmatrix} = \begin{pmatrix} p_0^n \\ p_1^n \\ p_2^n \\ p_3^n \end{pmatrix} + \begin{pmatrix} -\beta\xi_0 \\ 0 \\ 0 \\ 2\xi_3\ p_e \end{pmatrix} \tag{63}$$

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

The code for the implementation of time dependent solution is presented in the appendix C.

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.

```
array([  [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,   -2.99757363e-02,    1.08992721e+00]   ]
```

5 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 out model functions when compared with real life data from a well

5.1 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

$$p_0 = p_w + \frac{\Delta y}{2}\frac{\partial p}{\partial y}|_{r_w} + \mathcal{O}(\Delta y^2) \tag{64}$$

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. Substituting it in the above equation we get

$$p_w = p_0 - \frac{\Delta y}{2}\alpha - \mathcal{O}(\Delta y^2) \tag{65}$$

5.2 Well test data

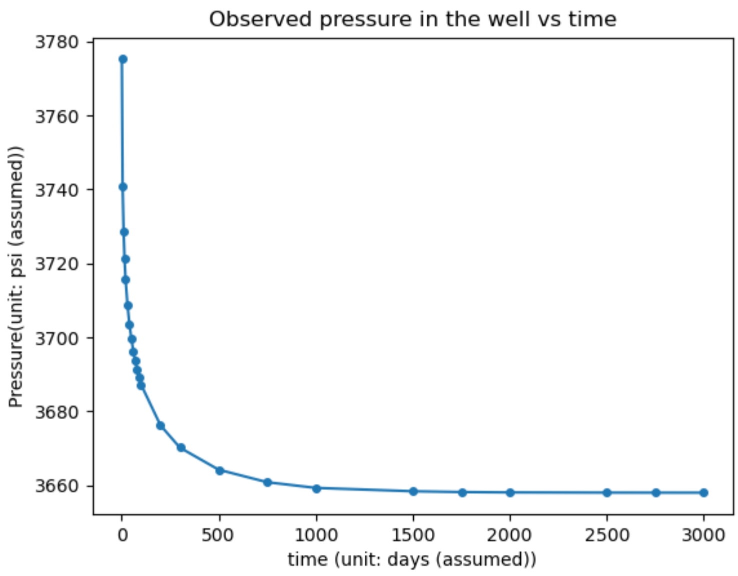

Figure 5: Observed pressure in the well vs time

Figure 5 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).

5.3 Fitting numeric 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 days
- the units of the pressure axis is psi
- the first observation is taken on day 1 i.e. the production starts on day 0 but the data starts being recorded on day 1.
- the pressure throughout the reservoir is uniform (p_i) on day 0.

A Tylors expression

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

$$f(x+h) = f(x) + \frac{1}{1!}f'(x)h + \frac{1}{2!}f''(x)h^2 + \frac{1}{3!}f'''(x)h^3 + \cdots \qquad (66)$$

Generalising the above equation 66

$$f(x+h) = f(x) + \sum_{n=1}^{\infty} \frac{1}{n!} f^n(x)h^n \qquad (67)$$

In numerical modelling we usually truncate the tylor expression after a certain number of terms. Thus, expanding equation 67 upto n terms can also be written as

$$f(x+h) = f(x) + \frac{1}{1!}f'(x)h + \cdots + \frac{1}{(n-1)!}f^{n-1}(x)h^{n-1} + \mathcal{O}(h^n) \qquad (68)$$

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

Ignoring the error term for now and expanding equation 68 to first detivative

$$\begin{aligned} f(x+h) &= f(x) + \frac{1}{1!}f'(x)h + \mathcal{O}(h^2) \\ \implies f'(x) &= \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h) \end{aligned} \qquad (69)$$

$$f_i' = \frac{f_{i+1} - f_i}{h} + \mathcal{O}(h) \qquad (70)$$

Equation 69 is the first derivative of $f(x)$ by forward difference. Similarly reverse difference can be calculated as

$$f(x-h) = f(x) - \frac{1}{1!}f'(x)h + \mathcal{O}(h^2) \qquad (71)$$

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

$$f_i' = \frac{f_i - f_{i-1}}{h} + \mathcal{O}(h^2) \qquad (73)$$

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

$$f_i' = \frac{f_{i+1} - f_{i-1}}{2h} + \mathcal{O}(h) \qquad (74)$$

From equation 68 taking $dy \equiv \pm h$ and expanding to the second derivative term we get

$$f(x+h) = f(x) + \frac{1}{1!}f'(x)h + \frac{1}{2!}f''(x)h^2 + \frac{1}{3!}f'''(x)h^3 + \mathcal{O}(h^4) \qquad (75)$$

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

adding equations 75 and 76 we get

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

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

$$\equiv f_i'' = \frac{f_{i+1} - 2f_i + f_{i-i}}{h^2} - \mathcal{O}(h^2) \tag{79}$$

Equations 70, 73, 74, and 79 form the mathematical basis for calculation of first and second derivatives and gradients numerically. These are arrived at using truncation of the tylor series and thus have associated errors of $\mathcal{O}(h)$ and $\mathcal{O}(h^2)$.

B Code: Steady state equation

```
class PressureSolver:
    """
    A finite difference solver to solve pressure distribution in
    a reservoir, logarithmic grid has been used, y = ln(r/rw)
    The solver uses SI units internally, while "practical field units"
    are required as input.

    Input arguments:
        name                                symbol   unit
        -------------------------------------------------------------
        Number of grid points                N       dimensionless
        Constant time step                   dt      days
        Well radius                          rw      ft
        Outer reservoir boundary             re      ft
        Height of reservoir                  h       ft
        Absolute permeability                k       mD
        Porosity                             phi     dimensionless
        Fluid viscosity                      mu      mPas (cP)
        Total (rock+fluid) compressibility   ct      1 / psi
        Constant flow rate at well           Q       bbl / day
        Initial reservoir pressure           pi      psi
        Laziness                             lazines index takes value between (0,1,2
        ------------------------------------------------------------

    """
    def __init__(self,N, dt,
                 rw=0.318,
                 re=1000.0,
                 h=11.0,
                 phi=0.25,
                 mu=1.0,
                 ct=7.8e-6,
                 Q=1000.0,
                 k=500,
                 pinit=4100.0,
                 laziness=1):
        ######################################################################
        #####            Physical parameter set up             ################
        ######################################################################
        # Unit conversion factors (input units --> SI)
        self.ft_to_m_ = 0.304
        self.psi_to_pa_ = 6894.75729
        self.day_to_sec_ = 24.*60.*60.
        self.bbl_to_m3_ = 0.1589873

        # Grid
        self.N_ = N
        self.rw_ = rw*self.ft_to_m_
        self.re_ = re*self.ft_to_m_
        self.h_ = h*self.ft_to_m_

        # Rock and fluid properties
        self.k_ = k*1e-15 / 1.01325 # from Darcy to m^2
        self.phi_ = phi
        self.mu_ = mu*1e-3 # from mPas to Pas
        self.ct_ = ct / self.psi_to_pa_

        # Initial and boundary conditions
        self.Q_ = Q*self.bbl_to_m3_ / self.day_to_sec_
        self.pinit_ = 1 # pinit*self.psi_to_pa_

        # Time control for simulation
        self.dt_ = dt*self.day_to_sec_

        # TO DO: Add more stuff below here....
        # (grid coordinates, dy, eta, etc.
        ######################################################################


        ######################################################################
        #####                 Model parameter set up         ################
        ######################################################################
        #Grid discretization
        min_ = 0
        max_ = np.log(self.re_/self.rw_)
        self.maxy = max_
        self.h=(max_-min_)/self.N_    # delta y: grid spatial spacing
        self.y=np.arange(0,max_-self.h/2,self.h)+self.h/2  # grif
        self.r = self.rw_*np.exp(self.y)
        self.lazy = laziness


        #alpha
        self.alpha = 1#self.Q_*self.mu_/(2*np.pi*self.h_*self.k_)

        #print(self.y)#, self.r, self.re_, self.rw_)
        #print(np.log(self.re_/self.rw_))

        ######################################################################

    def tri_diag(self,a, b, c, k1=-1, k2=0, k3=1):
        '''
        a,b,c diagonal terms
            default k-values for 4x4 matrix:
            | b0 c0 0  0 |
            | a0 b1 c1 0 |
            | 0  a1 b2 c2|
            | 0  0  a2 b3|
        '''
        a1=np.diag(a, k1)
        a2 =np.diag(b, k2)
        a3=np.diag(c, k3)
        return a1 + a2 + a3

    def analytical_steadystate(self,y):
        '''
        returns the analytical solution of
        '''
        return -self.alpha*(np.log(self.re_/self.rw_)-y)+self.pinit_

    def make_equation_steadystate(self):
        '''
        laziness = 2 very lazy
                 = 1 not so lazy
                 = 0 accurate
        '''
        a=np.ones(self.N_-1)
        b=-2*np.ones(self.N_)
        c=np.ones(self.N_-1)
        self.rhs=np.zeros(self.N_)
        #print('make eq',a,b,c)
        if self.lazy==2: # very lazy soluti
            b[0]=-1
            self.rhs[0]=self.alpha*self.h
            self.rhs[-1]=-self.pinit_
            self. A=self.tri_diag(a,b,c)
        elif self.lazy==1: # not so lazy solution
            b[0]=-1
            b[-1]=-3
            self.rhs[0]=self.alpha*self.h
            self.rhs[-1]=-2*self.pinit_
            self.A=self.tri_diag(a,b,c)

    def solve_steady_state(self):
        self.make_equation_steadystate()
        self.soln = np.linalg.solve(self.A,self.rhs)
        return(self.soln)

    def error(self):
        self()
        ana = self.analytical_steadystate(self.y)
        #print(len(ana), len(self.soln))
        return np.abs(ana-self.soln )

    def __call__(self):
        return self.solve_steady_state()
```

C Code: time dependent equation

```python
class PressureSolver:
    """
    A finite difference solver to solve pressure distribution in
    a reservoir, logarithmic grid has been used, y = ln(r/rw)
    The solver uses SI units internally, while "practical field units"
    are required as input.

    Input arguments:
        name                                    symbol   unit
        --------------------------------------------------------------
        Number of grid points                    N       dimensionless
        Constant time step                       dt      days
        Well radius                              rw      ft
        Outer reservoir boundary                 re      ft
        Height of reservoir                      h       ft
        Absolute permeability                    k       mD
        Porosity                                 phi     dimensionless
        Fluid viscosity                          mu      mPas (cP)
        Total (rock+fluid) compressibility       ct      1 / psi
        Constant flow rate at well               Q       bbl / day
        Initial reservoir pressure               pi      psi
        Laziness                                 lazines index takes value between (0,1,2
        -------------------------------------------------------------

    """

    def __init__(self, N, dt,
                 rw=0.318,
                 re=1000.0,
                 h=11.0,
                 phi=0.25,
                 mu=1.0,
                 ct=7.8e-6,
                 Q=1000.0,
                 k=500,
                 pinit=4100.0,
                 laziness=1):
        #####################################################################
        #####            Physical parameter set up           ################
        #####################################################################
        # Unit conversion factors (input units --> SI)
        self.ft_to_m_ = 0.3048
        self.psi_to_pa_ = 6894.75729
        self.day_to_sec_ = 24.*60.*60.
        self.bbl_to_m3_ = 0.1589873

        # Grid
        self.N_ = N
        self.rw_ = rw*self.ft_to_m_
        self.re_ = re*self.ft_to_m_
        self.h_ = h*self.ft_to_m_

        # Rock and fluid properties
        self.k_ = k*1e-15 / 1.01325  # from Darcy to m^2
        self.phi_ = phi
        self.mu_ = mu*1e-3  # from mPas to Pas
        self.ct_ = ct / self.psi_to_pa_

        # Initial and boundary conditions
        self.Q_ = Q*self.bbl_to_m3_ / self.day_to_sec_
        self.pinit_ = pinit*self.psi_to_pa_

        # Time control for simulation
        self.dt_ = dt*self.day_to_sec_

        #####################################################################
        #####                 Model parameter set up         ################
        #####################################################################
        # Grid discretization
        min_ = np.log(self.rw_/self.rw_)
        max_ = np.log(self.re_/self.rw_)
        self.maxy = max_
        self.dy = (max_-min_)/self.N_    # delta y: grid spatial spacing
        #self.y = np.linspace(0, 1, self.N_, endpoint=False) * self.maxy+self.dy/2  # grid spat
        self.y=np.arange(0,max_-self.dy/2,self.dy)+self.dy/2#+0.00262812
        self.r = self.rw_*np.exp(self.y)
        #self.lazy = laziness

        # model combined parameters
        self.alpha = self.Q_*self.mu_/(2*np.pi*self.h_*self.k_)
        self.beta = self.alpha*self.dy
        # hydraulic diffusivity
        self.eta = self.k_/(self.mu_*self.phi_*self.ct_)
        # this paramter xi0 needs to be multiplied by e^(-2y)*delta(t)
        # to get xi on the spatial grid
        self.xi = self.eta/self.rw_**2/self.dy**2*self.dt_*np.exp(-2*self.y)

        #####################################################################

    def tri_diag(self, a, b, c, k1=-1, k2=0, k3=1):
        '''
        a,b,c diagonal terms
            default k-values for 4x4 matrix:
            | b0 c0 0  0 |
            | a0 b1 c1 0 |
            | 0  a1 b2 c2|
            | 0  0  a2 b3|
        '''
        a1 = np.diag(a, k1)
        a2 = np.diag(b, k2)
        a3 = np.diag(c, k3)
        return a1 + a2 + a3

    def analytical_steadystate(self, y):
        '''
        returns the analytical solution of well steady state well pressure
        '''
        return -self.alpha*(np.log(self.re_/self.rw_)-y)+self.pinit_

    def make_equation_steadystate(self):
        '''
        creates the A matrix and the RHS for the steady state solution
        '''
        # initiate the diagonals and the RHS values
        a = np.ones(self.N_-1)
        b = -2*np.ones(self.N_)
        c = np.ones(self.N_-1)
        self.rhs = np.zeros(self.N_)

        # modify the elements of 0th diagonal
        # and the RHS for applying boundary conditions
        b[0] = -1
        b[-1] = -3
        self.rhs[0] = self.alpha*self.dy
        self.rhs[-1] = -2*self.pinit_
        self.A = self.tri_diag(a, b, c)  # create the sparce square matrix

    def solve_steady_state(self):
        '''
        Makes the matrix for steady state solution and then solves for pressure
        '''
        self.make_equation_steadystate()
        self.soln_steady_state = np.linalg.solve(self.A, self.rhs)

    def make_equations(self, dt = None):
        '''
        creates the A matrix and the 'd' array for RHS for the time dependant solutio
        '''
        # initiate diagonals of the A matrix and d array
        if not dt: dt = self.dt_
        xi = self.xi
        a = -xi[1:]  # term below the diagonal
        b = 1+2*xi   # terms on the diagonal
        c = -xi[:-1] #terms above the diagonal
        d = np.zeros(self.N_)
        self.xi = xi
```

```
# modify the elements of 0th diagonal
# and the RHS for applying boundary conditions
b[0] -= xi[0]
b[-1] += xi[-1]
d[0] = -self.beta*xi[0]
d[-1] = 2*self.pinit_*xi[-1]

#create sparce matrix A and store d matrix
self.A = self.tri_diag(a, b, c)
self.rhs_d = d
```