1. High level code overview

- *`material_balance.py`* holds 2 functions that check for incremental material balance. One checks for total material balance, the other one checks for water material balance using saturation. These two functions will be called in the main loop after each step.
- *`solver_functions.py`* holds all calculations for every timestep of the simulation. It includes transmissibility calculations, mobility calculations, LHS matrix and RHS vector assembly and adding well model calculations.
- *`RelativePermeability.py`* fits the Corey-correlation coefficients and allow quick retrieval of relative permeability at any input saturation.
- *`PVT_Table.py`* has a class that stores and linear interpolate between values of pressure, viscosity, volume factor and density
- *`yaml_parser.py`* parses the input from yaml input file and feeds it into the main loop.
- *`main.py`* holds the main loop and any logging for plotting afterwards.

2. Input file

Refer to *"input_config/config.yml"* for more details. The config file for the validating run can be found in *"validation/sample_config.yml"*. To run the simulator, navigate to the parent working folder and run `main.py`.

3. Software requirements

Any recent version of python alongside with numpy, scipy and pandas.

4. Validation

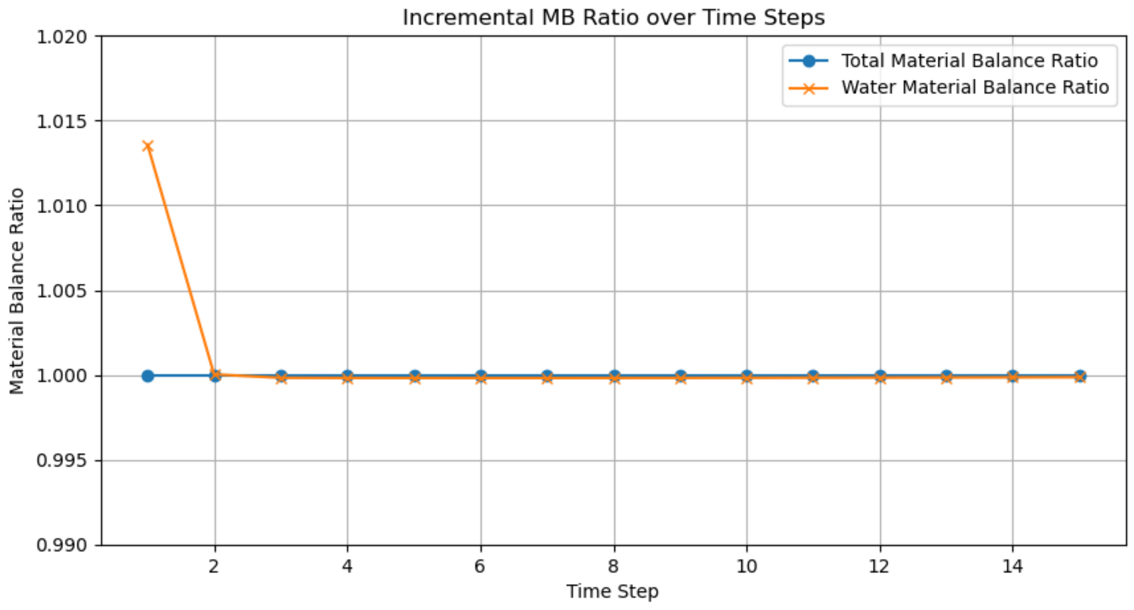
- I used the 1d example that we discussed in class to test my simulator. To create the input file "*validation/sample_config.yml",* I created the grid of dimension 5 x 1 x 1, and set the permeability to 1e-13 and porosity to 0.15. There is one injector well at cell [1, 0, 0] and a producer at [3, 0, 0]. The injector operates at 25 m3/day, and the producer has bottom hole pressure of 0.8 MPa.
- To obtain the PVT table, since we assumed constant compressibility and volume factor in the example, I created file *"validation/sample_pvt.csv"* that has the same value of viscosity, volume factor, compressibility and density across every value of pressure.
- To simulate the fit of the Corey rock fluid model, I sampled noiseless points from the given model in the example, then perform the fit to obtain the same n1 = n2 = 2
- After one step with dt = 43200s = 0.5 day,

  the simulated pressure is
  [1788691.6,   1786642.03,   1277340.26,   761029.045,   772931.01]
  compared to the in-class solution
  [1788300.1,   1786247.61,   1277145.56,   761035.032,   772932.141]

My saturation
[0.25001013, 0.2603195, 0.25003464, 0.253074, 0.25005882]
In-class solution
[0.25001, 0.26032, 0.25003, 0.25312, 0.25006]

And my calculated oil production rate for the producer well is -33.6938 m3/day versus in-class value of -33.8046 m3/day.

- I also run the simulator with the above settings for another 20 steps and plot the incremental material balance ratio.

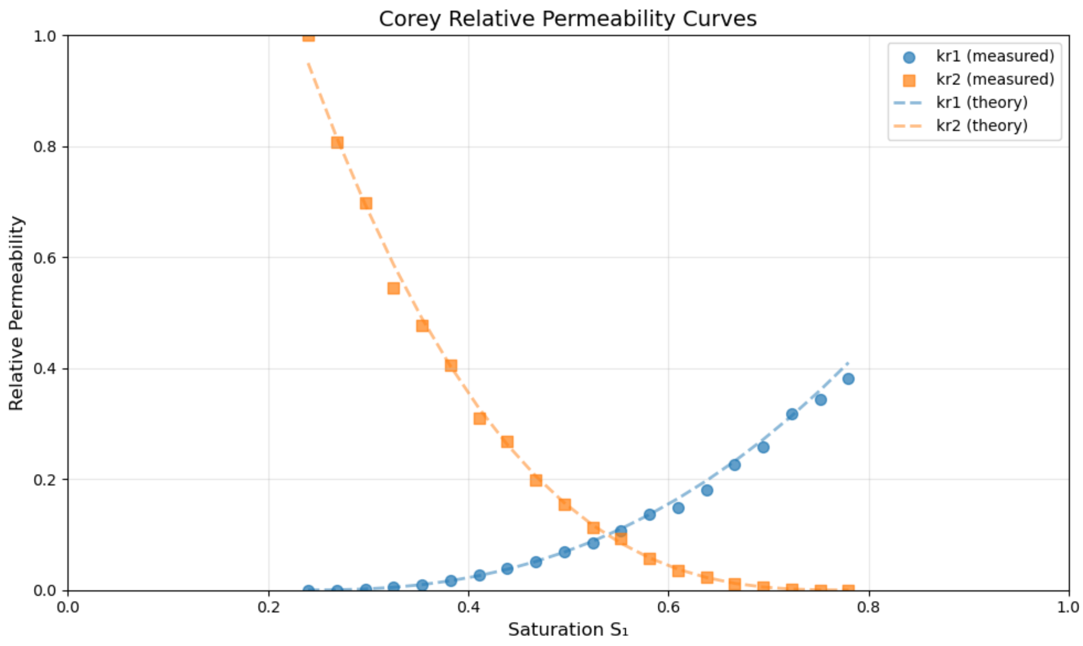


5. My specific configurations

- I want to make a water flooding case with a symmetrical 5-spot pattern. There is one producer in the center, and 4 injectors around the producer. Assume no dipping, cells on the same xy-plane have the same depth.
- *Rock-fluid model:* I initially wanted to take the table in Example 14-5 from *Reservoir Engineering Handbook* by T. Ahmed (Chapter 14, Principles of Waterflooding). The fitted coefficient from this data is n_w = 1.687 and n_o = 1.306. However, these coefficients are a little bit low, indicating little interference between phase flows.

S_w	k_{ro}	k_{rw}
0.24	0.95	0.00
0.30	0.89	0.01
0.40	0.74	0.04
0.50	0.45	0.09
0.60	0.19	0.17
0.65	0.12	0.28
0.70	0.06	0.22
0.75	0.03	0.36
0.78	0.00	0.41

Therefore, I took inspiration from the above table and set S1r = 0.24, S2r = 0.22, Kr1_max = 0.41 and Kr2_max = 0.95. I then manually set n_1 = 2.4 and n_2 = 2.8 as standard water-wet sandstone, and added some noise to simulate error from lab data. The noise is sampled from a normal distribution of mean 0 and std 0.05. The data can be found in *"input_config/rock_fluid.csv"*. Here is the rel. perm. Vs saturation plot.

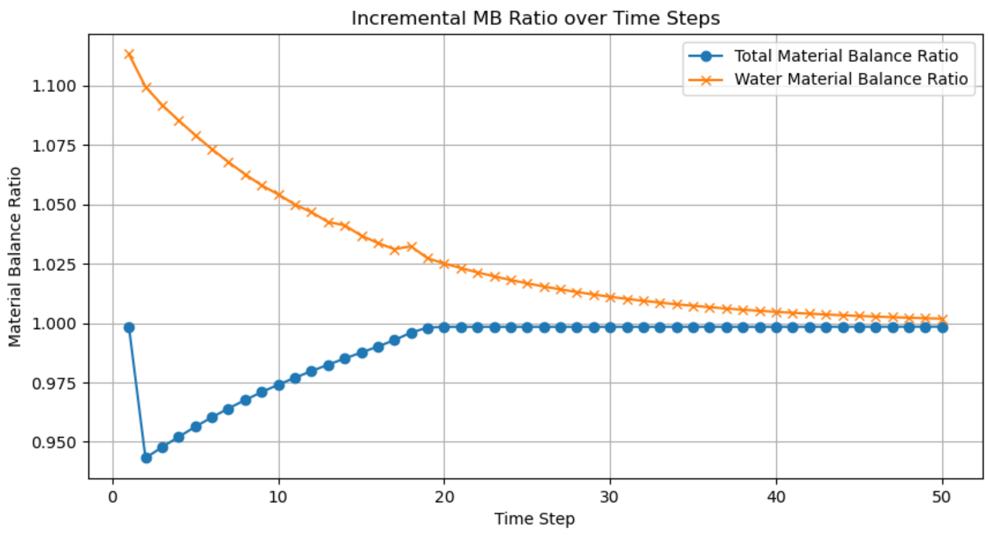


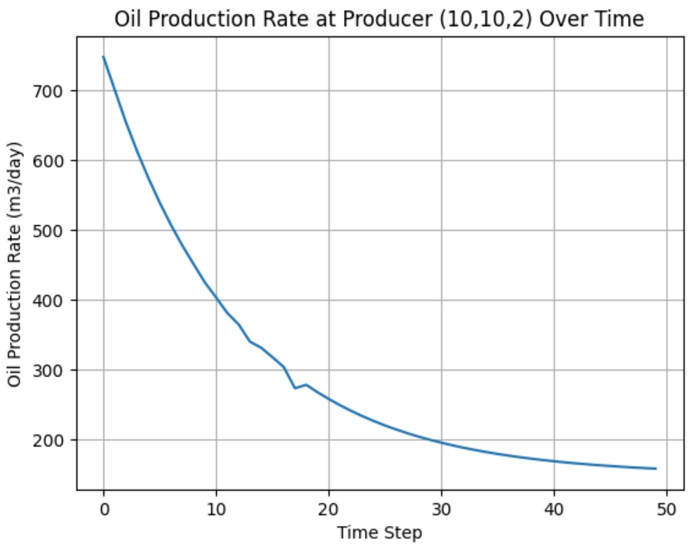
The fitted n1 = 2.3941 and n2 = 2.844. Our rock fluid model will use these two fitted coefficients to obtain relative permeability. Code to sample this data can be found in *"sampling_corey.py"*.

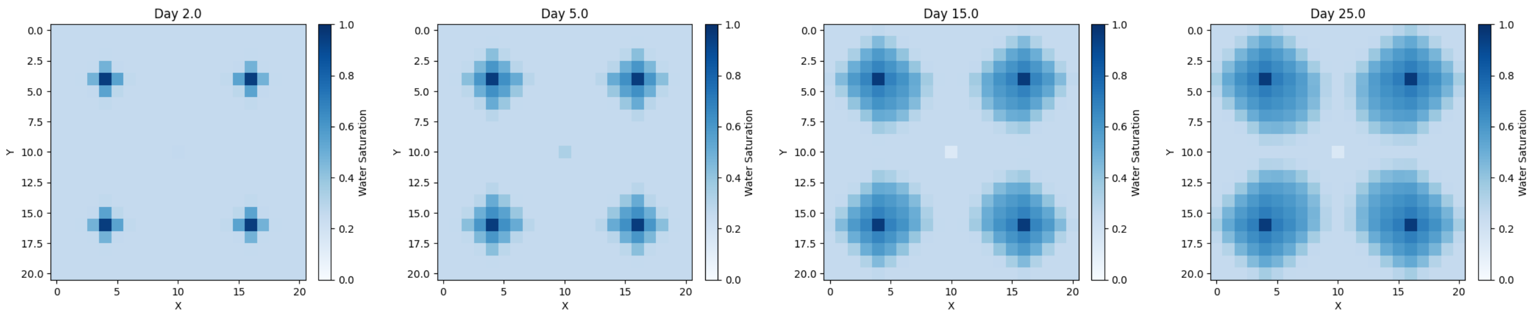
- *PVT Table:* I tried to find an actual table in literature but couldn't find a suitable one. There is a work that provides a PVT table for water (Z. Zhang et al. *Prediction of the PVT properties of water over wide range of temperatures and pressures from molecular dynamics simulation)* but it only has pressure range of 100 MPa to 30 GPa, so I decided to sample my own PVT table. I tried to search for the most general parameters on the internet, and include them in *"pvt_sampling.py"*. They look fine to me, but I understand an actual PVT table from a reservoir is much preferred. The interpolation functions for compressibility, volume factor, and viscosity are taken from CMG documentation.
- *Grid:* I chose a 21 by 21 by 5 grid, with dx = dy = 5m, dz = 10m. Depth top = 2000m. I set k_x = k_y = k_z = 1e-13m^2 = 0.1 Darcy, and porosity = 0.15. The assumption is that across 5 * 21 = 105m of the area of interest, the permeability and porosity did not change drastically. However, my implementation does handle anisotropy and non uniform grid size if necessary. The initial pressure is 12 MPa and initial water saturation is 0.25 for every grid block.
- *Well controls:* I did not implement timestep specific control. Rather, every well will operate with a fixed rate/bhp. Each injector is injecting 50 m3/day, and the producer is operating at bhp of 3.6 MPa.

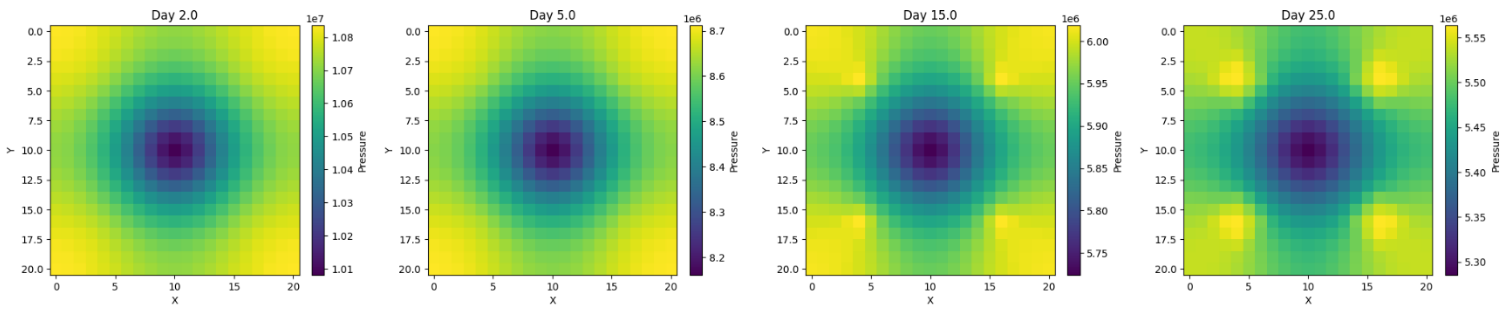
6. Results

- Using dt = 43200s = 0.5 day, and running for 50 steps = 25 days
- Below is the plot of incremental material balance ratio, oil production rate, pressure map and saturation map over time.









We can see the injectors and producers on the saturation and pressure map. Especially, we see on the saturation map, water is moving from the injectors towards the producer. Material balance wise, we started off being quite far away from 1, but gradually stabilized over time, reaching values very close to 1 around step 20. This behaviour is quite consistent across multiple set of dxdydz, initial conditions and dt in my experiments. I however cannot come up with an explanation for this behaviour.