Single-phase Fluid Finite-difference Simulator using Python

Benyamin Manullang, Institut Teknologi Bandung

Abstract

The author presents the development of a basic finite-difference reservoir simulator using Python as the programming language. The simulator is similar to that of an old traditional simulator, that is to say that it is used to solve a single-phase fluid flow, in a homogeneous and isotropic medium, and discretized in a cartesian coordinate system using a finite-difference approach. The author then gives a few examples with different flow direction (1D, 2D, or 3D).

Keywords: finite-difference, reservoir simulation, Python (programming language)

Introduction

Often, many undergraduate students majoring in petroleum engineering fail to understand how a reservoir simulator works behind the monitor. Most of these undergraduate students, if not all, must have taken introductory classes on programming, numerical method, and partial differential equations, but still fail to apply them to the field of reservoir simulation. As a result, many of them accept the results at face value without knowing what has happened inside the box. This paper can hopefully give clear explanations on how to utilize those basic knowledges and put them in the form of actual computer program.

Since computers are not able to evaluate the continuous form of a differential equation, we need to approximate the solution to a differential mathematical expression into its discrete form. One method that was widely used is known as finite-difference method. Although most commercial reservoir simulators available nowadays no longer use the traditional finite-difference approach, it is still an eye-opening experience to understand how the governing equation is translated into its finite-difference form. In fact, finite-difference is arguably more intuitive than other discretizing approaches (i.e. corner-point, control-volume).

After we derive the finite-difference form of the differential equation of interest, we then proceed with the presentation of how to put it in the form of computer program. Python is chosen as the programming language because it is easy to understand with its clear syntax.

Diffusivity Equation for Fluid Flow in Porous Media

In the field of reservoir engineering, the diffusivity equation governs the fluid flow and is derived using Darcy's law on the basis of conservation of mass. Firstly, one needs to choose the coordinate system that will represent the space. The choice of coordinate system is mainly influenced by the predicted nature of the flow. Due to the radial nature of the fluid flow, the diffusivity equation is usually derived using the cylindrical coordinate system with flow in θ and z direction neglected (see **Eq. 1**, with no sink/source term). However, in this paper we will only consider the cartesian coordinate system (see **Eq. 2**).

$$\phi \rho c_T \frac{\partial P}{\partial t} = -\frac{1}{r}\frac{\partial}{\partial r}\left(r\rho \frac{k_r}{\mu}\frac{\partial P}{\partial r}\right) \tag{1}$$

$$\phi \rho c_T \frac{\partial P}{\partial t} = -\frac{\partial}{\partial x}\left(\rho \frac{k_x}{\mu}\frac{\partial P}{\partial x}\right) - \frac{\partial}{\partial y}\left(\rho \frac{k_y}{\mu}\frac{\partial P}{\partial y}\right) - \frac{\partial}{\partial z}\left(\rho \frac{k_z}{\mu}\left(\frac{\partial P}{\partial z} - \rho g\right)\right) + q_{sc}(x, y, z, t) \tag{2}$$

Finite-difference Calculus

The concept of finite-difference forms the foundation of solving differential equations numerically. Instead of solving a differential equation that is supposed to be continuous everywhere in its domain, we look at discrete points and form difference equations. Another way to grasp this concept is that we are not evaluating the behavior of $\Delta f(x)$ as Δx gets closer to zero (see **Eq. 3**) as opposed to the actual definition of derivative (see **Eq. 4**). This method will introduce some error that depends on the chosen value of h.

$$\frac{df}{dx} \approx \frac{f_{x+\Delta x} - f_x}{h} \tag{3}$$

$$\begin{aligned} \frac{df}{dx} &= \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{(x + \Delta x) - (x)} \\ &= \lim_{\Delta x \to 0} \frac{\Delta f(x)}{\Delta x} \end{aligned} \tag{4}$$

App. A presents the derivation of diffusivity equation in cartesian coordinate system.

Python (programming language) and SciPy stack

Python is a programming language designed by Guido van Rossum in 1991. Python is widely known for its neat design, code readability, and its syntax which allows programmers to express concepts in fewer lines of code. It supports object-oriented programming paradigm which enables programmers to express their thinking as objects that interact with each other. Each object may have their own methods to interact with other objects or to perform specific operations.

This simulator makes use of the **SciPy stack**. SciPy stack is a Python-based ecosystem of open-source software for mathematics, science, and engineering. Two of the core packages which are used in this simulator are **NumPy** and **SciPy library**. NumPy provides numerical array objects, and routines to manipulate them. It has also become fundamental package for scientific computing with Python. SciPy library provides numerical routines and is used hand in hand with NumPy.

Physical Assumptions

Frequently, one will be faced with limitations when modelling a physical system. For example, one cannot know the value of porosity (ϕ) or permeability (k) at every coordinate in space since it is impossible to provide core samples of the entire reservoir system. The assumptions made in the development of this simulator are summarized below

Physical assumptions on fluid:

- There is no change in compositions that make up the fluid. This is also known as a black-oil fluid model.
- Fluid has small, constant compressibility. Highly compressible fluid such as gas is not applicable.

Physical assumptions on rock:

- The behavior of porosity stays the same throughout the reservoir.

Other assumptions:

- Temperature throughout the reservoir stays constant. Therefore, the only variable that affects mass conservation is pressure.

Designing the Data Structures

The first question we should address is how do we build the computer model? Or specifically, how do we represent the behavior of a **reservoir system** (consisting of **fluid** and **rock**) in a **3D *cartesian* space** using a computer program? Using this reasoning, the author declare classes as sketched in **Fig. 1**. These classes form the core data structures that model the reservoir. We store the code for these classes in `core.py` file.

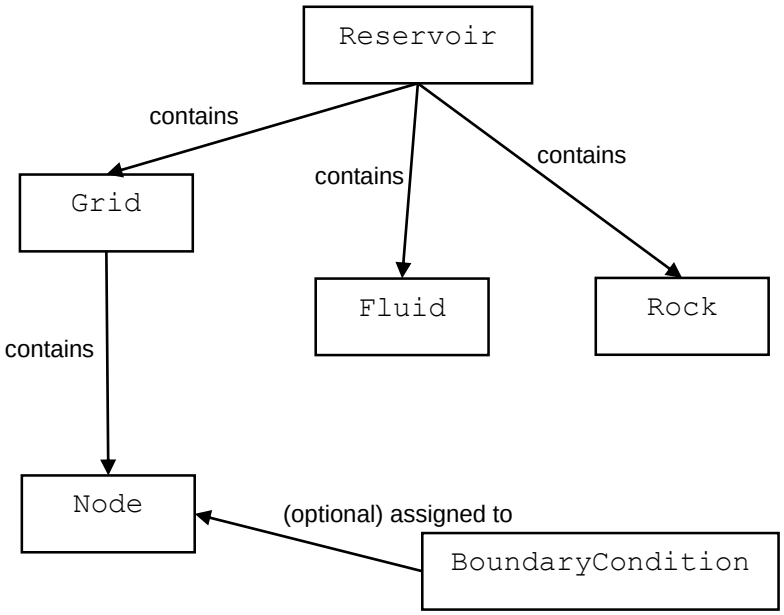

Figure 1 - Schematic of classes

`Node` and `Grid` Classes

`Node` objects will be used to represent a point in space. `Node` will contain information such as the flattened index, the coordinate index (k, j, i notation), whether or not a node is a boundary node, and whether or not a node is a source/sink. Flattened index, in contrast to 3D coordinate index, is a 1D array index. For instance, a `Grid` object with a dimension of (1, 2, 5) will have one gridblock with respect to z direction, two gridblocks with respect to y direction, and five gridblocks with respect to x direction. Each gridblock is bound to a `Node` object. **Fig. 2** explains how flattened index and coordinate index are assigned to each gridblock for a (1, 2, 5) `Grid`. This indexing scheme will later be useful when setting up a 2D matrix that consists of linear equations describing pressure relationship among all points at a time level.

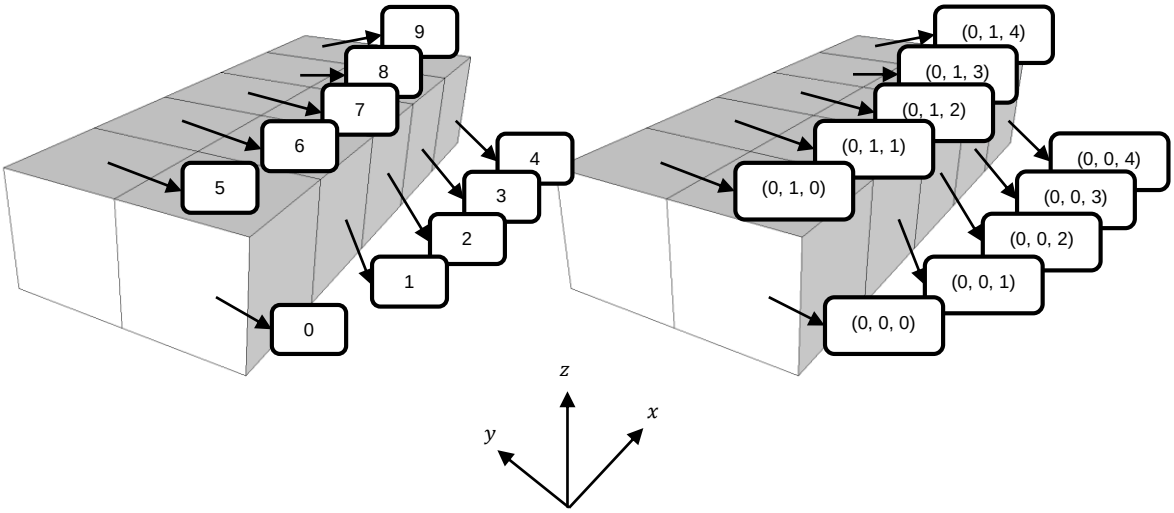

Figure 2 - Flattened index and 3D coordinate index in a 1×2×5 `Grid`

`Fluid` and `Rock` Classes

Fluid in the diffusivity equation presents itself as variables density (ρ) and viscosity (μ). Whereas the porous medium (rock) presents as variables porosity (ϕ) and absolute permeability (k). One should notice that density, viscosity, and porosity are functions of time. Density and porosity can further be combined to form total compressibility (c_T), which is equal to compressibility of fluid and compressibility of rock, $c_T = c_l + c_r$.

$$c_l = \frac{1}{\rho}\frac{d\rho}{dP} \qquad c_r = \frac{1}{\phi}\frac{d\phi}{dP} \tag{5}$$

As given by **Eq. 5**, for any given pressure value, one can determine the value of density and porosity using the following expression:

$$\rho = \rho_0 e^{c_l(P-P_0)} \qquad \phi = \phi_0 e^{c_r(P-P_0)}$$

We need to include a reference density or reference porosity with their reference pressure value. For *slightly compressible* fluid model, this equation of state (EOS) is sufficient. However, for *highly compressible* fluid, such as gas, the model must take into account other factors such as z-factor. Also, for this simulator, we will consider a constant value of viscosity (μ) at any given pressure value.

`Reservoir` class

All the information on `Grid`, `Fluid`, and `Rock` objects will then be passed into a `Reservoir` object. In addition, when instantiating a `Reservoir` object, one must also specify the actual dimension of the reservoir (in feet).

`BoundaryCondition` class

There are two types of boundary condition used in this simulator, Neumann condition (pressure gradient specified) and Dirichlet condition (pressure specified). **App. B** explains how each condition is implemented on a boundary node. By default, this simulator will specify a no-flow Neumann condition ($\frac{\partial P}{\partial s} = 0$) at every boundary node when a `Grid` object is instantiated.

Applying the Finite-difference Diffusivity Equation

The finite-difference form of diffusivity equation as derived in **App. A** is put into Python code in file `differentiator.py`. For each time step, function `oneStepDifferentiator` will go through each coordinate (`Node` object) in `Grid` object and perform what we would like to call as differentiation. The reader may find the code analogous to the derivation in **App. A**.

Function `oneStepDifferentiator` will then form as many linear difference equations as the number of `Node` objects. These linear equations will then be solved implicitly using `scipy.linalg.solve` function to get pressure distribution for the next time step. Pressure values for each flattened coordinate (as described in **Fig. 2**) will be generated into a result file.

3D Data Visualization

MRST's `plotCellData` function is used to visualize pressure distribution at a time level.

Simulation Examples

We show how to solve for pressure distribution by presenting four example problems. Each problem shows different flow direction. Using the same physical properties, one can observe how the choice of number of grids affect the calculation of pressure distribution. **Example 1** specifically address the same problem as given in Ertekin et al. (2001) (Example 5. 11).

***Example 1: 1D-flow* ($1 \times 1 \times 5$ `Grid` *dimension*)**

Suppose we have a porous medium that can be approximated by a cube shape with $75\ \text{ft} \times 1000\ \text{ft} \times 5000\ \text{ft}$ spatial dimension (see **Fig. 3**). There exists a constant-rate sink term at $x = 3{,}500\ \text{ft}$, $y = 500\ \text{ft}$. The rock and fluid properties for this problem are: $\rho_l = 62\,\frac{\text{lb}_\text{m}}{\text{ft}^3}$, $c_l = 3.5 \times 10^{-6}\ \text{psi}^{-1}$, $k_x = 15\ \text{mD}$, $\phi = 0.18$, and $\mu_l = 10\ \text{cP}$. With an initial pressure of $6{,}000\ \text{psi}$ and $\Delta t = 15\ \text{days}$, determine the pressure distribution during the first year of production.

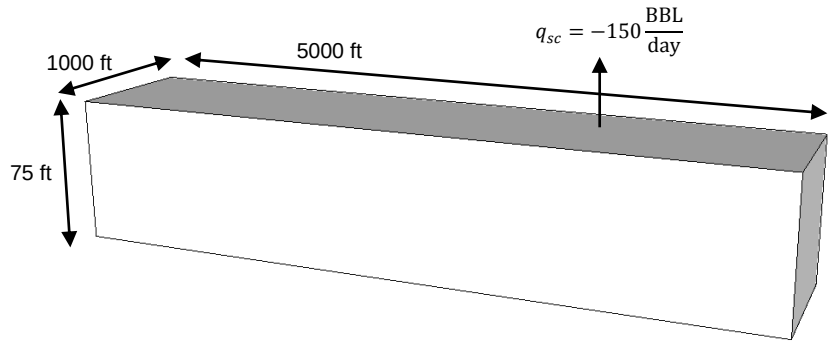

Figure 3 - Sketch of reservoir

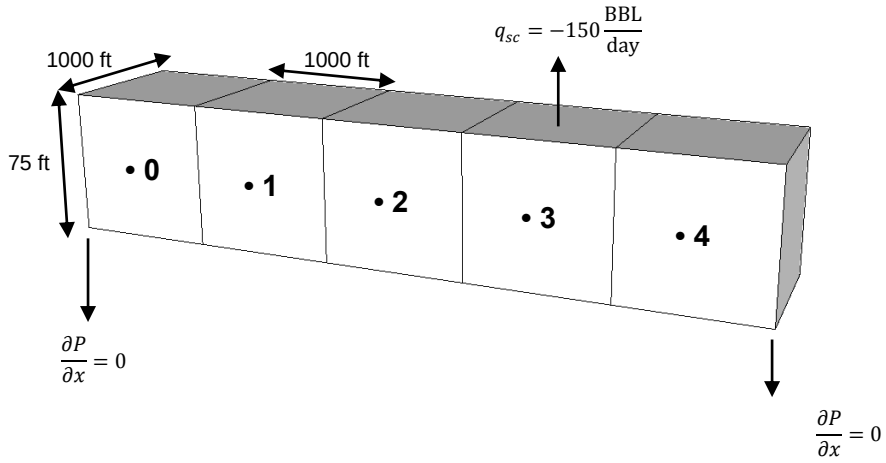

Figure 4 - Porous medium and grid block system for Example 1

For this problem, we position the sink term at grid coordinate (0, 0, 3). The resulting pressure distribution by the end of the year is visualized by **Fig. 5**. It can be seen that the results computed by this simulator agree with the example from Ertekin et al. (2001).

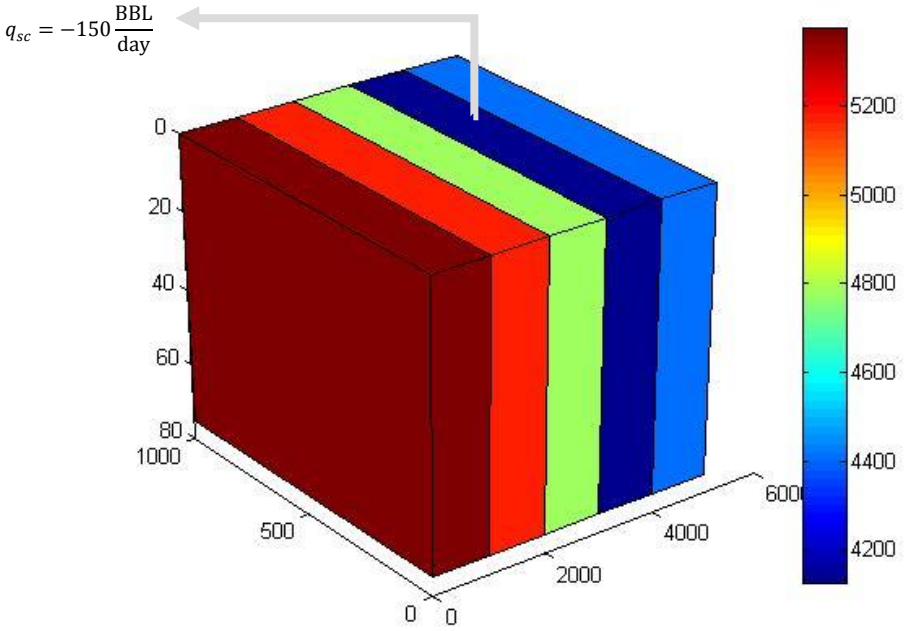

Figure 5 - Pressure distribution by the end of the year for Example 1

***Example 2: 2D-flow* ($1 \times 10 \times 20$ `Grid` *dimension*)**

For this problem, we use the same specifications as **Example 1**. We approximate the location of the sink term at grid coordinate (0, 4, 13). The 3D visualization of pressure distribution is given by **Fig. 6**.

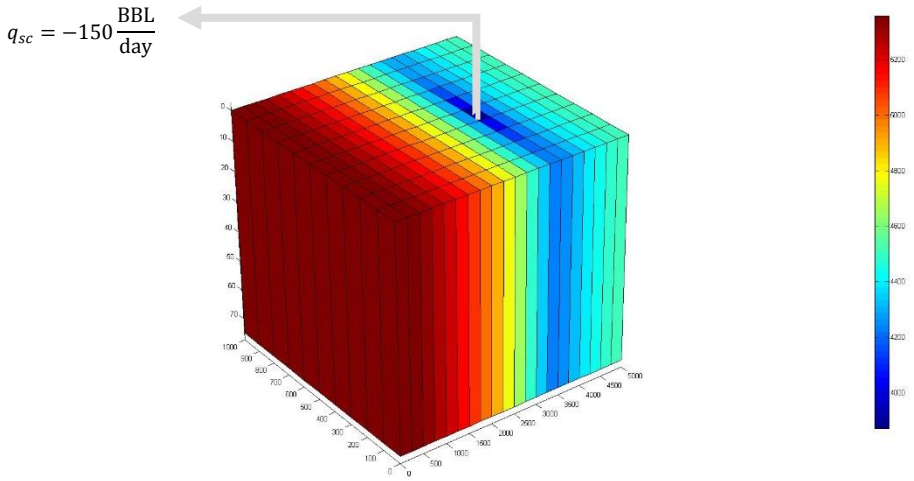

Figure 6 - Pressure distribution by the end of the year for Example 2

***Example 3: 3D-flow* ($5 \times 40 \times 50$ `Grid` *dimension*)**

For this problem, we use the same specifications as **Example 1.** We approximate the location of the sink term at grid coordinate (0, 19, 34). The 3D visualization of pressure distribution is given by **Fig. 7**.

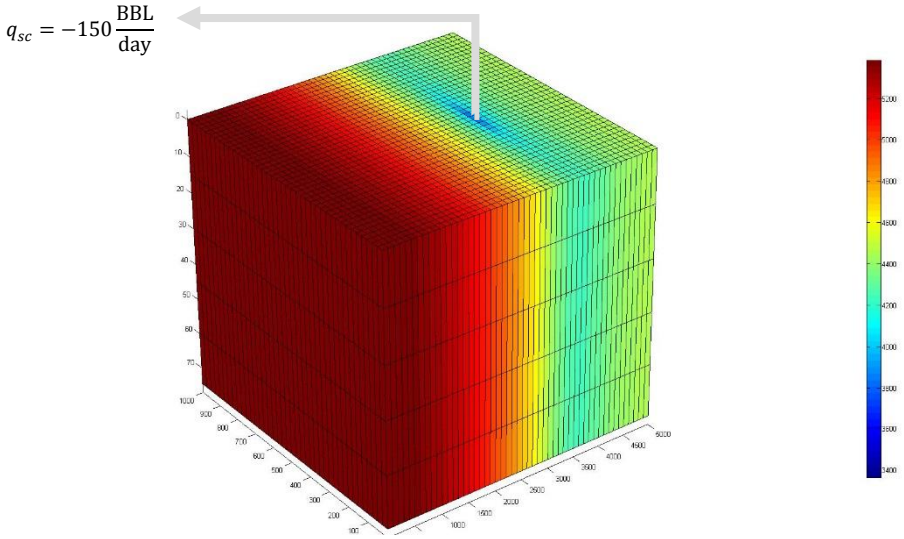

Figure 7 - Pressure distribution by the end of the year for Example 3

***Example 4: 3D-flow* ($5 \times 51 \times 51$ `Grid` *dimension*)**

In this example, we consider a cube-shaped physical reservoir with $75\ \text{ft} \times 5000\ \text{ft} \times 5000\ \text{ft}$ spatial dimension. And we set the location of the sink term at grid coordinate (0, 24, 24). The sink term has the same constant-rate value as before, $q_{sc} = -150 \frac{\text{BBL}}{\text{day}}$. Other specifications remain the same. The 3D visualization of pressure distribution is given by **Fig. 8**.

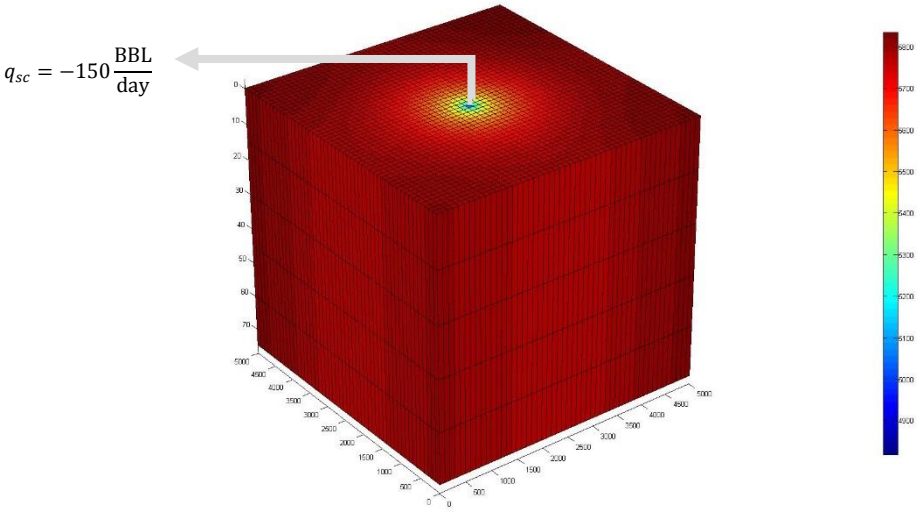

Figure 8 - Pressure distribution by the end of the year for Example 4

Concluding Remarks

This paper presented a detailed implementation of building an implicit, single-phase, slightly compressible, black-oil fluid using finite-difference approach in the form of Python code. One can revisit the code for study purpose and may extend or add new features.

Acknowledgments

The author wish to thank Zuher Syihab for helpful discussions and recommendations.

References

Ertekin, T., Abou-Kassem, J. H., and King, G. R. 2001. ***Basic Applied Reservoir Simulation***. Richardson, Texas: Society of Petroleum Engineers, Inc.

Lie, K.-A. 2014. ***An Introduction to Reservoir Simulation Using MATLAB***. Oslo, Norway: SINTEF ICT, Department of Applied Mathematics.

Single-phase Fluid Finite-difference Simulator using Python. https://github.com/benjdewantara/fdressim

Appendix A - Finite-difference Derivation of Diffusivity Equation

We begin by observing the conservation of mass equation,

$$\frac{d(m_{cv})}{dt} = \dot{m}_{in} - \dot{m}_{out} + \dot{m}_{sc} \qquad \textbf{(A1)}$$

Evaluating the right-hand side, for simplicity, we consider mass rate ($\dot{m}_x$) that only flows in x direction, with term $\dot{m}$ being $\rho u A$,

$$\begin{aligned} \dot{m}_{in} &= \rho u_{x-\frac{\Delta x}{2}} A_{x-\frac{\Delta x}{2}} \\ \dot{m}_{out} &= \rho u_{x+\frac{\Delta x}{2}} A_{x+\frac{\Delta x}{2}} \\ \dot{m}_{in} - \dot{m}_{out} + \dot{m}_{sc} &= \rho u_{x-\frac{\Delta x}{2}} A_{x-\frac{\Delta x}{2}} - \rho u_{x+\frac{\Delta x}{2}} A_{x+\frac{\Delta x}{2}} + \dot{m}_{sc} \end{aligned}$$

Using the derivative definition as follows,

$$\begin{aligned} \frac{\partial(\rho u_x A_x)}{\partial x} &= \lim_{\Delta x \to 0} \frac{(\rho u_x A_x)_{x-\frac{\Delta x}{2}} - (\rho u_x A_x)_{x+\frac{\Delta x}{2}}}{\left(x - \frac{\Delta x}{2}\right) - \left(x + \frac{\Delta x}{2}\right)} \\ \lim_{\Delta x \to 0} -\Delta x \frac{\partial(\rho u_x A_x)}{\partial x} &= \lim_{\Delta x \to 0} (\rho u_x A_x)_{x-\frac{\Delta x}{2}} - (\rho u_x A_x)_{x+\frac{\Delta x}{2}} \end{aligned}$$

Thus, with a slight abuse of notation, the right-hand side of the equation becomes,

$$\dot{m}_{in} - \dot{m}_{out} + \dot{m}_{sc} = -\Delta x \frac{\partial(\rho u_x A_x)}{\partial x} + \dot{m}_{sc}$$

Evaluating the left-hand side of the equation, noticing that $m_{cv} = \phi \rho V_b$, with term V_b being $\Delta x \Delta y \Delta z$,

$$\frac{d(m_{cv})}{dt} = \frac{d(\phi \rho V_b)}{dt}$$

Coming back to the earlier mass conservation equation,

$$\begin{aligned} \frac{d(m_{cv})}{dt} &= \dot{m}_{in} - \dot{m}_{out} + \dot{m}_{sc} \\ \frac{d(\phi \rho V_b)}{dt} &= -\Delta x \frac{\partial(\rho u_x A_x)}{\partial x} + \dot{m}_{sc} \end{aligned}$$

Since V_b and A_x are constants,

$$
\begin{aligned}
V_b \frac{d(\phi\rho)}{dt} &= -A_x \Delta x \frac{\partial(\rho u_x)}{\partial x} + \dot{m}_{sc} \\
\frac{d(\phi\rho)}{dt} &= \frac{-A_x \Delta x}{V_b} \frac{\partial(\rho u_x)}{\partial x} + \frac{\dot{m}_{sc}}{V_b} \\
\frac{d(\phi\rho)}{dP} \frac{\partial P}{\partial t} &= \frac{-A_x \Delta x}{V_b} \frac{\partial(\rho u_x)}{\partial x} + \frac{\dot{m}_{sc}}{V_b} \\
\phi\rho c_T \frac{\partial P}{\partial t} &= \frac{-A_x \Delta x}{V_b} \frac{\partial(\rho u_x)}{\partial x} + \frac{\dot{m}_{sc}}{V_b}
\end{aligned}
$$

We now have the general continuity equation. One can generalize the equation further by taking into account the flow in y and z direction as follows,

$$
\phi\rho c_T \frac{\partial P}{\partial t} = \frac{-A_x \Delta x}{V_b} \frac{\partial(\rho u_x)}{\partial x} + \frac{-A_y \Delta y}{V_b} \frac{\partial(\rho u_y)}{\partial y} + \frac{-A_z \Delta z}{V_b} \frac{\partial(\rho u_z)}{\partial z} + \frac{\dot{m}_{sc}}{V_b} \qquad \textbf{(A2)}
$$

The moment we evaluate the velocity term (u_s), using Darcy's law, we will arrive at the diffusivity equation. Again, for clarity we shall derive the equation further by considering only the flow in x direction. Recall the equation for Darcy's law in some s direction,

$$
u_s = \frac{-k}{\mu}\left(\frac{\partial P}{\partial s} - \rho g \frac{\partial z}{\partial s}\right)
$$

For flow only in x direction, the continuity equation becomes,

$$
\phi\rho c_T \frac{\partial P}{\partial t} = \frac{-A_x \Delta x}{V_b} \frac{\partial}{\partial x}\left(\rho \frac{-k}{\mu} \frac{\partial P}{\partial x}\right) + \frac{\dot{m}_{sc}}{V_b}
$$

We begin translating any $\frac{\partial}{\partial x}$ term (spatial derivative) using finite-difference method,

$$
\begin{aligned}
\phi\rho c_T \frac{\partial P}{\partial t} &= \frac{A_x \Delta x}{V_b} \frac{\partial}{\partial x}\left(\rho \frac{k}{\mu} \frac{\partial P}{\partial x}\right) + \frac{\dot{m}_{sc}}{V_b} \\
&= \frac{A_x \Delta x}{V_b} \frac{1}{\Delta x}\left[\left(\rho \frac{k}{\mu} \frac{\partial P}{\partial x}\right)_{x+\frac{\Delta x}{2}} - \left(\rho \frac{k}{\mu} \frac{\partial P}{\partial x}\right)_{x-\frac{\Delta x}{2}}\right] + \frac{\dot{m}_{sc}}{V_b} \\
&= \frac{A_x}{V_b}\left[\left(\frac{\rho k}{\mu}\right)_{x+\frac{\Delta x}{2}} \left(\frac{\partial P}{\partial x}\right)_{x+\frac{\Delta x}{2}} - \left(\frac{\rho k}{\mu}\right)_{x-\frac{\Delta x}{2}} \left(\frac{\partial P}{\partial x}\right)_{x-\frac{\Delta x}{2}}\right] + \frac{\dot{m}_{sc}}{V_b} \\
&= \left[\left(\frac{\rho k A_x}{\mu V_b}\right)_{x+\frac{\Delta x}{2}} \frac{1}{\Delta x}\,(P_{x+\Delta x} - P_x) - \left(\frac{\rho k A_x}{\mu V_b}\right)_{x-\frac{\Delta x}{2}} \frac{1}{\Delta x}\,(P_x - P_{x-\Delta x})\right] + \frac{\dot{m}_{sc}}{V_b} \\
\phi\rho c_T \frac{\partial P}{\partial t} &= \left[\left(\frac{\rho k A_x}{\mu V_b \Delta x}\right)_{x+\frac{\Delta x}{2}} (P_{x+\Delta x} - P_x) - \left(\frac{\rho k A_x}{\mu V_b \Delta x}\right)_{x-\frac{\Delta x}{2}} (P_x - P_{x-\Delta x})\right] + \frac{\dot{m}_{sc}}{V_b}
\end{aligned}
$$

We can group the term $\left(\frac{\rho k A_x}{\mu V_b \Delta x}\right)_x$ and define them as transmissibility, T_x,

$$\phi\rho c_T \frac{\partial P}{\partial t} = \left[T_{x+\frac{\Delta x}{2}}(P_{x+\Delta x} - P_x) - T_{x-\frac{\Delta x}{2}}(P_x - P_{x-\Delta x})\right] + \frac{\dot{m}_{sc}}{V_b}$$
$$= \left[T_{x+\frac{\Delta x}{2}}P_{x+\Delta x} - \left(T_{x+\frac{\Delta x}{2}} + T_{x-\frac{\Delta x}{2}}\right)P_x + T_{x-\frac{\Delta x}{2}}P_{x-\Delta x}\right] + \frac{\dot{m}_{sc}}{V_b}$$

We then proceed by translating the $\frac{\partial}{\partial t}$ term (time derivative) using finite-difference. We also assign superscript t or $t + \Delta t$ to any variable to specify the time level.

$$\phi\rho c_T\left(\frac{P_x^{t+\Delta t} - P_x^t}{\Delta t}\right) = \left[T^t_{x+\frac{\Delta x}{2}}P^{t+\Delta t}_{x+\Delta x} - \left(T^t_{x+\frac{\Delta x}{2}} + T^t_{x-\frac{\Delta x}{2}}\right)P_x^{t+\Delta t} + T^t_{x-\frac{\Delta x}{2}}P^{t+\Delta t}_{x-\Delta x}\right] + \frac{\dot{m}_{sc}}{V_b} \quad \textbf{(A3)}$$
$$\frac{\phi\rho c_T}{\Delta t}\left(P_x^{t+\Delta t} - P_x^t\right) = \left[T^t_{x+\frac{\Delta x}{2}}P^{t+\Delta t}_{x+\Delta x} - \left(T^t_{x+\frac{\Delta x}{2}} + T^t_{x-\frac{\Delta x}{2}}\right)P_x^{t+\Delta t} + T^t_{x-\frac{\Delta x}{2}}P^{t+\Delta t}_{x-\Delta x}\right] + \frac{\dot{m}_{sc}}{V_b}$$

We have arrived at the final finite-difference form of diffusivity equation for flow only in x direction. One should notice that any P_x term encountered so far is actually P at some x, y, z coordinate (i.e. $P_{x,y,z}$). Similarly, the term $P_{x+\Delta x}$ actually denotes $P_{x+\Delta x,y,z}$. The subscript y, z is left out for brevity when considering the differential with respect to x. We can further generalize this form and factor in the flow terms for y and z direction as follows,

$$\begin{bmatrix}\text{differential}\\ \text{in } t\end{bmatrix} = \begin{bmatrix}\text{differential}\\ \text{in } x \text{ direction}\end{bmatrix} + \begin{bmatrix}\text{differential}\\ \text{in } y \text{ direction}\end{bmatrix} + \begin{bmatrix}\text{differential}\\ \text{in } z \text{ direction}\end{bmatrix} + \frac{\dot{m}_{sc}}{V_b} \quad \textbf{(A4)}$$

where:

$$\begin{bmatrix}\text{differential}\\ \text{in } x \text{ direction}\end{bmatrix} = T^t_{x+\frac{\Delta x}{2}}P^{t+\Delta t}_{x+\Delta x} - \left(T^t_{x+\frac{\Delta x}{2}} + T^t_{x-\frac{\Delta x}{2}}\right)P_x^{t+\Delta t} + T^t_{x-\frac{\Delta x}{2}}P^{t+\Delta t}_{x-\Delta x}$$
$$\begin{bmatrix}\text{differential}\\ \text{in } y \text{ direction}\end{bmatrix} = T^t_{y+\frac{\Delta y}{2}}P^{t+\Delta t}_{y+\Delta y} - \left(T^t_{y+\frac{\Delta y}{2}} + T^t_{y-\frac{\Delta y}{2}}\right)P_y^{t+\Delta t} + T^t_{y-\frac{\Delta y}{2}}P^{t+\Delta t}_{y-\Delta y}$$
$$\begin{bmatrix}\text{differential}\\ \text{in } z \text{ direction}\end{bmatrix} = T^t_{z+\frac{\Delta z}{2}}P^{t+\Delta t}_{z+\Delta z} - \left(T^t_{z+\frac{\Delta z}{2}} + T^t_{z-\frac{\Delta z}{2}}\right)P_z^{t+\Delta t} + T^t_{z-\frac{\Delta z}{2}}P^{t+\Delta t}_{z-\Delta z} + \left(-T_{z+\frac{\Delta z}{2}} + T_{z-\frac{\Delta z}{2}}\right)\rho g\Delta z$$

$$\begin{bmatrix}\text{differential}\\ \text{in } t\end{bmatrix} = \frac{\phi\rho c_T}{\Delta t}\left(P^{t+\Delta t}_{x,y,z} - P^t_{x,y,z}\right)$$

$$T_s = \left(\frac{\rho k A_s}{\mu V_b \Delta s}\right)_s$$

Appendix B - Implementing Boundary Condition

The finite-difference derivation as described in **App. A** is based on the central difference scheme. This type of scheme is only applicable to a gridblock which has neighboring gridblocks in its x, y, and z direction. However, this is not the case for a boundary gridblock. This situation is best described using a 3D grid sketch (**Fig. B1**). For a boundary gridblock, a special treatment based on either Dirichlet or Neumann boundary condition needs to be made.

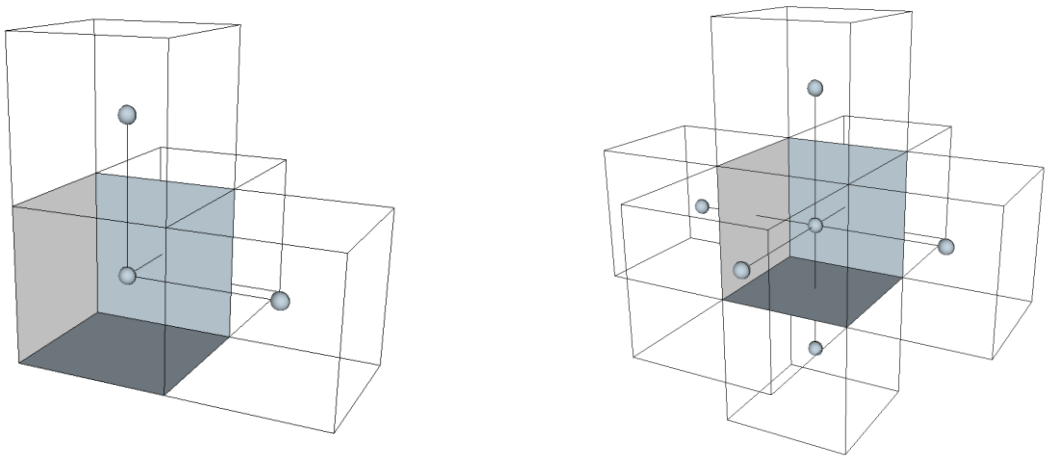

Figure B1 - Sketch of a boundary node (left) and a regular node (right) with their neighboring gridblocks

We restrict further discussion on the flow equation only looking at flow in x direction for simplicity. For instance, consider a boundary gridblock at x that does not have both gridblocks at $x - \frac{\Delta x}{2}$ and $x + \frac{\Delta x}{2}$ to interact with. This means with respect to x, either $P_{x+\Delta x}$ or $P_{x-\Delta x}$ is missing and the diffusivity equation (**Eq. A4**) cannot be completed. Either one of them is the boundary pressure. We need to evaluate how Dirichlet or Neumann condition is imposed on this boundary pressure.

1. **Dirichlet condition (pressure specified)**

   Suppose the boundary pressure is $P_{x-\Delta x}$, for Dirichlet condition, one can directly specify the boundary pressure,

$$P_{x-\Delta x} = C$$

The value of C is some constant but can also be a function of time. But we will assume it is constant throughout the time.

2. Neumann condition (pressure gradient specified)

For this condition, we are given the gradient value, $\frac{\partial P}{\partial x}$, which is equal to C. We can then translate this gradient into the following,

$$\frac{\partial P}{\partial x} = C$$

$$\frac{\partial P}{\partial x} = \frac{P_x - P_{x-\Delta x}}{\Delta x}$$

$$P_{x-\Delta x} = P_x - C\Delta x$$

Similarly, if the boundary pressure is $P_{x+\Delta x}$, the gradient expression can be approximated as follows,

$$\frac{\partial P}{\partial x} = \frac{P_{x+\Delta x} - P_x}{\Delta x}$$

$$P_{x+\Delta x} = P_x + C\Delta x$$

For a no-flow condition, the gradient is zero ($C = 0$), thus the boundary pressure is directly equal to P_x.