Report

Gabriela Berenice Diaz Cortes

Abstract

This report is written as a complement of the literature study, with the basic concepts that are important for the research in the project *Physics-based preconditioners for large-scale subsurface flow simulation.*
The idea of this report is to have a document that will be the basis for the introduction chapter of the thesis, and to have a summary of the concepts for future consultations.

Chapter 1

Linear Algebra

In this chapter there are some basic concepts about vector and matrices algebra. More detail about this concepts can be found in the references [**?** **?**].

1.1 Vectors

A vector $x \in \mathbf{C}^n$ is an array of elements $x_i \in \mathbf{C}$ such that:

$$x \in \mathbf{C}^n \qquad x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \qquad x^T = \begin{bmatrix} x_1 & \dots x_n \end{bmatrix} \qquad x_i \in \mathbf{C}$$

The basic operations for a vector are:

Addition $\mathbf{C}^n \times \mathbf{C}^n \to \mathbf{C}^n$:

$$z = x + y,$$

where

$$z_i = x_i + y_i.$$

Scalar-vector multiplication $\mathbf{C} \times \mathbf{C}^n \to \mathbf{C}^n$:

$$z = \alpha x,$$

where

$$z_i = \alpha a_i.$$

Vector multiplication $\mathbf{C}^n \times \mathbf{C}^n \to \mathbf{C}$

$$z = x^T y,$$

where

$$z_i = \sum_{i=1}^{n} x_i y_i.$$

1.1.1 Vector Inner Product

If we have a vector space $\mathbf{X}$, an inner product is defined as a mapping s from $\mathbf{X} \times \mathbf{X}$ into $\mathbf{C}$

$$x, y \in \mathbf{X} \to s(x, y) \in \mathbf{C}. \tag{1.1}$$

This product satisfies the next three conditions:
1. Is linear with respect to x, i.e.,

$$s(\lambda_1 x_1 + \lambda_2 x_2, y) = \lambda_1 s(x_1, y) + \lambda_2 s(x_2, y), \qquad \forall x_1, x_2 \in \mathbf{X}, \forall \lambda_1, \lambda_2 \in \mathbf{C}.$$

2. Is Hermitian $s(x, y)$ i.e.,

$$s(y, x) = \overline{s(x, y)}.$$

3. Is positive definite, i.e.,

$$s(x, x) > 0, \qquad \forall x \neq 0.$$

Any inner product satisfies the Cauchy-Schwartz inequality:

$$|s(x, y)|^2 \leq s(x, x)s(y, y).$$

For the vector space $\mathbf{X} = \mathbf{C}^n$, the canonical inner product is the Euclidean inner product, that is the product of two vectors $(x_i)_{i=1,\ldots,n}$ and $y = (y_i)_i = 1, \ldots, n$ in $\mathbf{C}^n$

$$(x, y) = \sum_{i=1}^{n} x_i \overline{y_i},$$

A property of the Euclidean inner product is

$$(Ax, y) = (x, A^H y), \qquad \forall x, y \in \mathbf{C}^n.$$

The A-inner product for a vector x is defined by

$$(x, x)_A = x^T A x. \tag{1.2}$$

1.2 Matrices

A $n \times m$ matrix A is an array of real numbers $a_{ij} \in \mathbf{C}$ $i = 1, 2, \ldots, n, j = 1, 2, \ldots, m$.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}.$$

The space of the matrices of $n \times m$ is denoted by $\mathbf{C}^{n \times m}$ and has the following operations:

Addition $\mathbf{C}^{n \times m} \times \mathbf{C}^{n \times m} \to \mathbf{C}^{n \times m}$:

$$C = A + B,$$

where

$$c_{ij} = a_{ij} + b_{ij}.$$

Scalar-matrix multiplication $\mathbf{C} \times \mathbf{C}^{n\times m} \rightarrow \mathbf{C}^{n\times m}$:

$$C = \alpha A,$$

where

$$c_{ij} = \alpha a_{ij}.$$

Matrix multiplication $\mathbf{C}^{n\times p} \times \mathbf{C}^{p\times m} \rightarrow \mathbf{C}^{n\times m}$

$$C = AB,$$

where

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}.$$

Some properties of the matrices are the following:

The transpose of a matrix $A \in \mathbf{R}^{n\times m}$ is:

$$C = A^T,$$

$$c_{ij} = a_{ji}, \qquad i = 1, \ldots, m, j = 1, \ldots, n.$$

If $A \in \mathbf{C}^{n\times m}$ is more relevant the transpose conjugate matrix denoted by A^H defined by:

$$A^H = \overline{A}^T = \overline{A^T}.$$

Square matrix. A matrix $A \in \mathbf{C}^{n\times m}$ is square if it has the same number of rows and columns $n = m$. A special matrix is called the identity matrix, this matrix contains ones in the principal diagonal, and zeros in the rest of the entries. This matrix is sometimes written in terms of the Kronecker symbol δ_{ij} that takes the value of 1 if $i = j$ and zero otherwise.

$$I = \{\delta_{ij}\}_{i,j=1,\ldots,n}.$$

The identity matrix satisfies $IA = AI = A$ for any $A \in \mathbf{C}^{n\times m}$.

The inverse (A^{-1}) of a matrix A is such that:

$$AA^{-1} = A^{-1}A = I.$$

The determinant of an $n \times n$ matrix is given by:

$$det(A) = \sum_{j=1}^{n} a_{1j} det(A_{1j})$$

where A_{1j} is a matrix obtained deleting the first row and the $j-th$ column of A, this new matrix contains $(n-1)\times(n-1)$ elements.

A matrix $A \in \mathbf{C}^{n\times m}$ is known as singular if $det(A) = 0$ and nonsingular if $det(A) \neq 0$. Some of the properties of the determinant are the following:

- $det(AB) = det(A)det(B)$.

- $det(A^T) = det(A)$.
- $det(\alpha A) = \alpha^n det(A)$.
- $det(\overline{A}) = \overline{det(A)}$.
- $det(I) = 1$.

A matrix $A \in \mathbf{C}^{n \times m}$ can have a particular structure that can be useful when solving linear systems, some of these structures are

- Symmetric matrices: $A^T = A$.
- Hermitian matrices: $A^H = A$.
- Normal matrices: $A^H A = AA^H$.
- Nonnegative matrices: $a_{ij} \geq 0, i, j = 1, \ldots, n$.
- Unitary matrices: $Q^H Q = I$.
- Diagonal matrices: $a_{ij} = 0$, for $i \neq j$.
- Upper triangular matrices: $a_{ij} = 0$, for $i > j$.
- Lower triangular matrices: $a_{ij} = 0$, for $i < j$.
- Upper bidiagonal matrices: $a_{ij} = 0$, for $j \neq i$ or $j \neq i + 1$.
- Lower bidiagonal matrices: $a_{ij} = 0$, for $j \neq i$ or $j \neq i - 1$.
- Tridiagonal matrices: $a_{ij} = 0$, for $|i - j| > 1$.
- Banded matrices: $a_{ij} \neq 0$ if $i - m_l \leq j \leq i + m_u$, with m_l, m_u two nonnegative integers.
- Upper Hessenberg matrices: $a_{ij} = 0$ for any i, j such that $i > j + 1$.
- Positive Definite: $(Au, u) > 0, \qquad \forall u \in \mathbf{R}^n, \qquad u \neq 0$.
- Symmetric Positive Definite SPD. If A is symmetric and Positive Definite.

1.2.1 Eigenvalues and eigenvectors

The eigenvalues $\lambda \in \mathbf{C}$ of a square matrix $A \in C^{n \times n}$ are complex scalars such that $Au = \lambda u$ for $u \in \mathbf{C}^n$. The vector u is an eigenvector of A associated with λ. The *spectrum* of A, $\sigma(A)$ is the set of all eigenvalues of A. Some definitions and properties related to the eigenvalues of A are listed below.

The characteristic polynomial of A is a function that maps λ to the value

$$p_A(\lambda) = det(A - \lambda I),$$

which is a polynomial of degree n.

The spectral radius , $\rho(A)$, is the maximum modulus of the eigenvalues

$$\rho(A) = \max_{\lambda \in \sigma(A)} |\lambda|.$$

The scalar λ is a root of the characteristic polynomial, and an eigenvalue of A if and only if

$$p_A(\lambda) = det(A - \lambda I) = 0.$$

If λ is an eigenvalue of A, then $\bar{\lambda}$ is an eigenvalue of A^H.

1.2.2 Null space and column space.

Column space. Let $A \in \mathbb{R}^{n\times m}$ be a non square matrix,

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix} = \begin{pmatrix} v_1 & v_2 & \dots & v_m \end{pmatrix}, \qquad v_i = \begin{pmatrix} a_{1i} \\ a_{2i} \\ \dots \\ a_{ni} \end{pmatrix},$$

The set of all possible linear combination of the column vectors v_i of this matrix is called the column space $\mathbf{R}(A)$ or range of A.

$$Range(A) = \{c \in \mathbb{R}^m | Ac = c_1 v_1 + c_2 v_2 + ... + c_m v_m\}$$

The system $Ax = b$ is solvable if and only if the vector b can be expressed as a combination of the columns of A, i.e. if b is in the column space of A. In this case $b = x_1 v_1 + x_2 v_2 + ... + x_m v_m \in Range(A)$, with v_i the $i - th$ column of A. If we take b and b' in the column space, so that $Ax = b$ for some x and $Ax' = b'$ for some x', then $A(x + x') = b + b'$ is also a combination of the columns. The column space of all vectors b is closed under addition. If b is in the column space of A, also is any multiple cb. If some combination of columns produces b, (say $Ax = b$), then multiplying that combination by c will produce cb. In other words, $A(cb) = cb$.
Null space. The null space of a matrix A consist of all the vectors x such that $Ax = 0$. It is denoted by $\mathbf{N}(A)$. If $Ax = 0$ and $Ax' = 0$, then $A(x + x') = 0$. And if $Ax = 0$ then $A(cx) = 0$.

1.2.3 Vector and matrix norms

A *vector norm*, is a function $\mathbb{C}^n \to \mathbb{C}$ that satisfies the following conditions:

1. Positivity
$$||u|| \geq 0 \qquad \forall \qquad u \in \mathbf{C}^n.$$
2. Homogeneity
$$||cu|| = |c|||u||.$$
3. Triangular inequality
$$||u + v|| \leq ||u|| + ||v||.$$

The p norm of a vector $u \in \mathbf{C}^n$ is defined by

$$||u||_p = \left[\sum_{i=1}^{n} |u_i|^p\right]^{\frac{1}{p}}$$

The most important norms are listed below

$$||u||_1 = |u_1| + |u_2| + ... + |u_n|,$$

$$||u||_2 = \left[|u_1|^2 + |u_2|^2 + ... + |u_n|^2\right]^{1/2},$$

$$||u||_\infty = \max_{i=1,\ldots,n} |u_i|.$$

The $A - norm$ of a vector x is defined as:

$$||x||_A = \sqrt{(x,x)_A} = \sqrt{x^T A x}. \tag{1.3}$$

The *p-norm* of a matrix $A \in R^{n\times m}$is given by:

$$||A||_{pq} = \max_{u\in \mathbf{C}^m, u\neq 0} \frac{||Au||_p}{||u||_q}$$

This norm satisfies the usual properties of a norm:

$$||A||_p \geq 0,$$

$$||cA||_p = |c|||A||_p.$$

$$||A_1 + A_2||_p \leq ||A_1||_p + ||A_2||_p.$$

And the *submultiplicative property*

$$||A_1 A_2||_p \leq ||A_1||_p ||A_2||_p.$$

For the case of $p = 1, p = 2$ and $p = \infty$ there are some expressions that allow us to compute the p-norm.

$$||A||_1 = \max_{1\leq j\leq n} \sum_{i=1}^{m} |r_{ij}|$$

$$||A||_2 = \max_{1\leq i\leq n} \lambda_k(A^T A) = \lambda_{max}(A^T A)$$

$$||A||_\infty = \max 1 \leq j \leq n \sum_{j=1}^{n} |r_{ij}|.$$

In the case when A is an $n \times n$ SPD matrix, $||A||_2 = \lambda_n(A)$. The *spectral condition number* of an $n \times n$ matrix A measured in a p-norm is defined as:

$$\kappa_p(A) = ||A||_p ||A^{-1}||_p.$$

For the 2-norm this can be:

$$\kappa_2(A) = \frac{\sqrt{\lambda_{max}(A^T A)}}{\sqrt{\lambda_{min}(A^T A)}}$$

If A is SPD this expression becomes

$$\kappa_2(A) = \frac{\lambda_{max}(A)}{\lambda_{min}(A)}$$

The *spectral radius* $\rho(A)$ of a matrix $A \in C^{n \times n}$ is defined as:

$$\rho(A) = \max_{\lambda_i \in \sigma(A)} |\lambda_i|.$$

A diagonal dominant matrix A satisfies

$$|a_{ii}| > \sum_{j=1, j \neq i}^{n} |a_{ij}|, \qquad i = 1, ..., n.$$

A matrix A is called irreducible iff it does not exist a permutation matrix P such that PAP^T is block upper triangular. The matrix A is called irreducibly strictly diagonal dominant if is irreducible and diagonal dominant. $Gershgorin's\ Theorem$ if $\lambda \in \sigma(A)$, then λ is located in one of the n closed disks in the complex plane centered in a_{ii}, and have a radius

$$\rho_i = \sum_{j=1, j \neq i}^{n} |a_{ij}|$$

$i.e$,

$$\lambda \in \sigma(A) \Rightarrow \exists i, \qquad \text{such that } |a_{ii} - \lambda| \leq \rho_i = \sum_{j=1, j \neq i}^{n} |a_{ij}|$$

The matrix A is an $M - matrix$ iff satisfies the following properties:

1. $a_{ii} > 0 \qquad i = 1, ..., n.$
2. $a_{ij} \leq 0 \qquad i \neq j, \qquad i, j = 1, ..., n.$
3. A is non-singular.
4. $A^{-1} \geq 0$

Chapter 2

System of linear equations

The solution of partial differential equations PDE's is sometimes carried out by numerical methods. Some of these methods, as the finite elements method, transform our problem into a system of equations of the form:

$$\begin{gathered}a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1\\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2\\ \cdots\\ \cdots\\ a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n = b_n.\end{gathered}$$

This system can be written in matrix form as:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n}\\ a_{21} & a_{22} & \dots & a_{2n}\\ \dots & \dots & \dots & \dots\\ a_{n1} & a_{n2} & \dots & a_{nn}\end{pmatrix}\begin{pmatrix}x_1\\ x_2\\ ..\\ x_n\end{pmatrix} = \begin{pmatrix}b_1\\ b_2\\ ..\\ b_n\end{pmatrix}.$$

Or,

$$Ax = b \tag{2.1}$$

This system can be solve with direct (finite) or iterative (infinite) algorithms. The finite algorithms achieve a final solution, meanwhile the infinite ones are stopped if the error is less than a fixed value. The iterative methods not always converge. In this section there is a short description of some direct and iterative methods.

If we have a right hand side $b = 0$, we always have the solution $x = 0$, but it is possible to find a infinity of other solutions. If we have a non square matrix $A \in \mathbb{R}^{n\times m}$, $n > m$, there are always infinite solutions.

Triangular Factors

If we have a linear system (2.1), after some operations, we can transform the matrix A into an upper triangular matrix, where all the entries below the diagonal are zero.

If A is an square matrix, and its columns are independent, we get a system like the following:

$$U = \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1(n-1)} & u_{1n} \\ 0 & u_{22} & \dots & u_{2(n-1)} & u_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & u_{(n-1)(n-1)} & u_{(n-1)n} \\ 0 & 0 & \dots & 0 & u_{nn} \end{pmatrix}$$

Where the diagonal elements are the pivots, or the first non zero elements in a row that will be use to create the zeros below them. If the matrix is not square $A \in \mathbb{R}^{n\times m}$, we will arrive to a matrix of the form:

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1(m-3)} & u_{1(m-2)} & u_{1(m-1)} & u_{1m} \\ 0 & 0 & u_{32} & \dots & u_{2(m-3)} & u_{2(m-2)} & u_{2(m-1)} & u_{2m} \\ 0 & 0 & u_{33} & \dots & u_{3(m-3)} & u_{3(m-2)} & u_{3(m-1)} & u_{3m} \\ \dots & \dots & \dots & \dots & \dots & & & \\ 0 & 0 & 0 & \dots & u_{(n-3)(m-3)} & u_{(n-3)(m-2)} & u_{(n-2)(m-1)} & u_{(n-2)m} \\ 0 & 0 & 0 & \dots & 0 & u_{(n-2)(m-2)} & u_{(n-2)(m-1)} & u_{(n-2)m} \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 & u_{(n-1)m} \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \end{pmatrix}.$$

The number of rows linearly independent of a matrix A is called the rank of A. The rank of a matrix A is the number of column vectors linearly independent of A, then A and A' have the same rank. That is the echelon form of the matrix A and has the following characteristics:
1. The pivots are the first non zero entries in their rows.
2. Below each pivot there is a column of zeros.
3. Each pivot lies to the right of the pivot in the row above.
4. The U matrix has a staircase pattern.
5. The last $m - r$ rows are zero rows, with r the rank of the matrix.

LU Decomposition Method

The idea of this method is to decompose a matrix A into two matrices (LU) which have the properties:

$$A = LU. \tag{2.2}$$

Where L is lower triangular, and U is upper triangular and $l_{ii} = 1$. Replacing (2.3) in (2.1), we transform the system into:

$$LUx = b.$$

If we rename $y = Ux$ we get

$$Ly = b.$$

We can solve this equation with *forward substitution* to get the value of y and then solve for x with *back substitution.*

$$Ux = y.$$

2.1 Direct Methods

2.1.1 Gauss Elimination Method

This method is used to get the solution of (2.1) when the matrix A is square, with algebraic operations in the rows of the matrix. This method consists in two steps. The first one is known as *Forward elimination*: we transform the A matrix into a upper triangular matrix *A'*. Then, we perform the *Back substitution*: now that our matrix *A'* is upper triangular, the last equation is of the form $a_{nn}x_n = b_n$. With this equation we can find the value of x_n and by substitution in the previous equations we can solve the complete system. Each time that we perform an operation in a i row of the matrix A we need to modify by the same amount the entry b_i to preserve the original system. To do this in a faster way we can add a column to the matrix A with the vector b, with this new matrix we can ensure that the system will remain the same after the triangulation of the matrix A. Our original system is:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ .. \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ .. \\ b_n \end{pmatrix},$$

where

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}, \qquad B = \begin{pmatrix} b_1 \\ b_2 \\ .. \\ b_n \end{pmatrix}.$$

Adding the vector B to the A matrix, the new matrix will be:

$$A' = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{pmatrix}.$$

We transform this matrix into an upper triangular matrix by making zero the entries below the main diagonal. This can be achieve making the following operations for all the rows below the main diagonal:

$$a_{j:} = a_{j:} - a_{i:} * \frac{a_{ji}}{a_{ii}}. \qquad ^1$$

Sometimes it happens that the elements in the main diagonal a_{ii} are much smaller, or even zero, than other element in the column in which they are $a_{:i}$. This can leads to some inaccurate solutions when we perform the operation $\frac{1}{a_{ii}}$. To avoid these small terms, it's necessary to move the row $a_{j:}$ with the largest coefficient in the column i to the row i, in such a way that the element in the main diagonal a_{ii} is the largest element in the column $a_{:i}$. This movement is called *pivoting*. After the *forward elimination*, using pivoting, we obtain a

[1] The ":" means that we take the hole j-row if we have $a_{j:}$ or the hole j-column if we have $a_{:j}$

matrix A" of the form:

$$A'' = \begin{pmatrix} a'_{11} & a'_{12} & a'_{13} & \dots & a'_{1(n-2)} & a'_{1(n-1)} & a'_{1n} & b'_1 \\ 0 & a'_{22} & a'_{23} & \dots & a'_{2(n-2)} & a'_{2(n-1)} & a'_{2n} & b'_2 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & & 0\dots & 0 & a'_{(n-1)(n-1)} & a'_{(n-1)n} & b'_{n-1} \\ 0 & 0 & & 0\dots & 0 & 0 & a'_{nn} & b'_n \end{pmatrix}.$$

When we have the upper triangular matrix, we can start the *back substitution* solving the system from the n-equation, and upwards. The x_n can be compute in the following way:

$$x_n = \frac{b'_n}{a'_{nn}}.$$

The other $x'_i s$ can be computed from the previous ones that are already known.

$$x_i = \frac{b'_i - \sum_{j=i+1}^{n} a'_{ij} x_j}{a'_{ii}}$$

2.1.2 LU Decomposition Method

The idea of this method is to decompose a matrix A into two matrices (LU) which have the properties:

$$A = LU. \tag{2.3}$$

Where L is lower triangular, and U is upper triangular and $l_{ii} = 1$. Replacing (2.3) in (2.1), we transform the system into:

$$LUx = b.$$

If we rename $y = Ux$ we get

$$Ly = b.$$

We can solve this equation with *forward substitution* to get the value of y and then solve for x with *back substitution.*

$$Ux = y.$$

2.1.3 Cholesky Decomposition for Symmetric Positive Definite Systems

In a linear system (2.1), if the matrix $A \in \mathbf{R}^{n\times n}$ is symmetric positive definite (SPD), there is a lower triangular matrix $L \in \mathbf{R}^{n\times n}$ with positive diagonal entries such that $A = LL^T$. This is called the Cholesky factorization and L is known as the Cholesky factor. The entries of the L matrix can be computed as follow:

$$l_{11} = \sqrt{a_{11}}$$

$$l_{ki} = \frac{a_{ik} - sum_{j=1}^{i-1} l_{ij} l_{kj}}{l_{ii}} \qquad i = 1, 2, \dots k-1, \qquad k = 2, 3, \dots, n, \qquad i \neq k$$

$$l_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2}$$

If the matrix is sparse, some entries that are zero in the lower triangular part of the matrix A might be nonzero in L. These new nonzero entries are known as fill-in. We can construct an incomplete factorization that has the same nonzero entries that the matrix A. This will be an incomplete factorization L_0. As there are some missing entries in the factorization $L_0 L_0^T \neq A$. This incomplete factorization can be improved by allowing some fill-in in the L_0 matrix, *i.e.*, we can construct a new L_k matrix taking some nonzero entries that are zero in the A matrix. The k index, is the level of fill-in allowed for the L_k matrix, the matrix with the greatest k is the L matrix obtained from the Cholesky factorization.

2.2 Iterative Methods

When we want to solve a linear system $Ax = b$, where the coefficient matrix A is *sparse*, the iterative methods are an alternative that uses less computer storage and computation time. In an iterative method, we start with a guess solution $\mathbf{x^0}$ and we try to approximate the solution $\mathbf{x}$. It is possible to decompose the matrix A into it's diagonal D, strict lower $-E$ and strict upper parts $-F$.

$$A = D - E - F$$

The methods presented in this section can be described in terms of these parts. When we want to solve a linear system of the form

$$Ax = b,$$

setting $A = M - N$, we can write the system as follows:

$$Ax = (M - N)x = b,$$

or

$$Mx = (N)x + b = (M - A)x + b.$$

This new system can be used to perform an iterative process

$$Mx^k = (M - A)x^{k-1} + b \qquad 1 \leq k,$$

or

$$x^k = M^{-1}(M - A)x^{k-1} + M^{-1}b. \tag{2.4}$$

The initial vector x^0 is an initial guess of the solution. The iterative method in (2.1) converges if it converges for any initial vector x^0. We can compute a sequence of vectors $x^1, x^2, \ldots, x^n$ from (2.4), and we need to choose M that:

- 1 The sequence $[x^k]$ is easily computed.
- 2 The sequence $[x^k]$ converges rapidly to a solution.

To prove this we can take x from (2.4)

$$x = M^{-1}(M - A)x + M^{-1}b = (I - M^{-1}A)x + M^{-1}b.$$

For an iteration x^k with the solution x we have:

$$x^k - x = (I - M^{-1}A)(x^{k-1} - x),$$

taking the absolute value of this equation:

$$||x^k - x|| = ||(I - M^{-1}A)(x^{k-1} - x)|| \leq ||(I - M^{-1}A)||||(x^{k-1} - x)||,$$

repeating this we have:

$$||x^k - x|| \leq ||(I - M^{-1}A)||^k||(x^0 - x)||,$$

If $||(I - M^{-1}A)|| < 1$, then

$$lim_{k\to\infty}||x^k - x|| = 0.$$

In each iteration we have an error vector, which is the distance to the solution

$$e^k = x - x^k.$$

As the error takes into account the exact solution x, computing the error is equivalent to computing the solution, to have an idea of the error, we can compute the residual, defined as:

$$r^k = b - Ax^k.$$

This leads to the next equation between the residual and the error:

$$Ae^k = r^k.$$

As we see before, we can write x^{k+1} as:

$$x^{k+1} = M^{-1}(M-A)x^k + M^{-1}b = (I - M^{-1}A)x^k + M^{-1}b = x^k + M^{-1}(b - Ax^k),$$

replacing r^k we have

$$x^{k+1} = x^k + M^{-1}r^k.$$

If A is sparse, each iteration requires $\mathcal{O}(cN)$ instead of $\mathcal{O}(N^2)$ flops for dense matrices. The error can be written in recursive form as:

$$\begin{aligned} e^{k+1} &= x - x^k \\ &= x - x^k - M^{-1}r^k \\ &= e^k - M^{-1}Ae^k \\ &= (I - M^{-1}A)e^k \end{aligned}$$

For the residual vector we have:

$$\begin{aligned} r^{k+1} &= b - Ax^{k+1} \\ &= b - Ax^k - AM^{-1}r^k \\ &= r^k - AM^{-1}r^k \\ &= (I - AM^{-1})r^k. \end{aligned}$$

Where $(I - M^{-1}A)$ and $(I - AM^{-1})$ are the error and residual propagation matrices.

2.2.1 Richardson Method

If M in (2.4) is the identity matrix, then:

$$\begin{aligned} x^k &= (I-A)x^{k-1} + b = x^{k-1} + r^{k-1}, \\ r^{k-1} &= b - Ax^{k-1}. \end{aligned}$$

2.2.2 Jacobi Method

If the matrix A that we want to solve is sparse, the solution x can be obtained solving the system for each x_i replacing the initial guess $\mathbf{x^0}$ in the original system (2.1).

$$x_i^1 = \frac{b_i - \sum_{j=1}^{n} a_{ij} x_j^0}{a_{ii}}, \qquad for \qquad a_{ij} \neq 0, \qquad i = 1, \ldots, n.$$

This can be rewritten as:

$$x^{k+1} = D^{-1}(E+F)x^k + D^{-1}b.$$

Where $M = D$ and $N = E + F$.This process is repeated, using the new values of $\mathbf{x}$ which are $\mathbf{x^1}$ as initial guess, until the x^{k+1} obtained with the iterative method converge to the solution within a specified tolerance ϵ.

$$max|\frac{x_i^{k+1} - x_i^k}{x_i^{k+1}}| \times 100\% < \epsilon.$$

This method allows to update all components of x^k independently from each other, so, it can be computed in parallel form. The error propagation of this method ($M = D$) is:

$$\begin{aligned} B_{JAC} &= I - M^{-1}A = I - D^{-1}(D - E - F) \\ &= I - I + D^{-1}E + D^{-1}F \\ &= L + U. \end{aligned}$$

Damped Jacobi Method

In the damped Jacobi method, there is a damping parameter ω and $M_{JAC\omega} = \frac{1}{\omega}D$.

2.2.3 Gauss-Seidel Iteration Method

This method is similar to the Jacobi method. In this case the convergence of the method is faster. Each iteration gives some new values for the approximation $x_i's$. If we want to compute a new x_i^k and the previous $x_{1\ldots(i-1)}^k$ have been computed, we can use these values to get the solution faster, using them to compute the x_i^k with this values obtained in this iteration, as well as the values obtained in the previous one that have not been computed yet.

$$x_i^k = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^k - \sum_{j=i+1}^{n} a_{ij} x_j^{k-1}), \qquad i = 1, \ldots, n.$$

or,

$$x_{k+1} = (D-E)^{-1}Fx_k + (D-E)^{-1}b.$$

In this case $M = D - E$ and $N = F$ This method uses the recently computed x^{k+1} vectors, therefore it converges faster than the Jacobi's Method.

2.2.4 Successive Over-relaxation Method

If we want to increase the convergence of the method, this can be done by adding a weight factor in the solutions computed in the previous iteration and the current iteration. This method is known as successive over-relaxation, and the weight factor is ω.

$$x_i^k = \frac{1}{a_{ii}}\omega(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^k) - (1-\omega)\sum_{j=i+1}^{n} a_{ij}x_j^{k-1}), \qquad i = 1, \dots, n.$$

The error propagation of this method $(M = D - E)$ is:

$$\begin{aligned} B_{GS} &= I - M^{-1}A = I - (D-E)^{-1}(D-E-F) \\ &= I - I + (D-E)^{-1}F \\ &= (D-E)^{-1}DD^{-1}F \\ &= (I-L)^{-1}U. \end{aligned}$$

2.2.5 Chebyshev Method

If we use an iterative method to solve the equation (2.1), we get the solution $u^1, u^2, \dots, u^k$. A way to improve the convergence of an iterative method is to find a new iteration that will accelerate the convergence of the previous method. This can be done by determining coefficients $\gamma_j(k), j = 0, \dots, k$ such that:

$$y^k = \sum_{j=0}^{k} \gamma_j(k)u^j,$$

these coefficients must satisfy:

$$\sum_{j=0}^{k} \gamma_j(k) = 1.$$

The problem now is to choose the $\gamma_j(k)$ so that the error $y^k - u$ is minimized. Taking the error of the k-iteration as $e^k = u^k - u$, such that $e^{(k+1)} = Q^k e^0$, the error will be given by:

$$y^k - u = \sum_{j=0}^{k} \gamma_j(k)(u^j - u) = \sum_{j=0}^{k} \gamma_j Q^j e^0.$$

Taking $p_k(z) = \sum_{j=0}^{k} \gamma_j(k)z^j$ with $p_k(1) = 1$, we get the 2-norm of the error:

$$||y^k - u||_2 = ||\sum_{j=0}^{k} \gamma_j Q^j e^0||_2 = ||p_k(Q)e^0||_2 \leq ||p_k(Q)||_2||e^0||_2$$

Taking $e^0 = u^0 - u$ this equation is:

$$||y^k - u||_2 \leq ||p_k(Q)||_2 ||u^0 - u||_2$$

The goal is to minimize $||p_k(Q)||_2$ for all polynomials. If Q is symmetric with eigenvalues λ_i, such that $\alpha \leq \lambda_n \leq \cdots \leq \lambda_1 \leq \beta < 1$, then:

$$||p_k(Q)||_2 = \max_{\lambda_i} |p_k(\lambda_i)| \leq \max_{a<\lambda<\beta} |p_k(\lambda)|.$$

If we want that $p_k(Q)$ be small, it's necessary a polynomial $p_k(z)$ that is small on $[\alpha, \beta]$ with the condition of $p_k(1) = 1$. The Chebyshev polynomials are of the form:

$$\begin{aligned} c_0(z) &= 1, \\ c_1(z) &= z, \\ c_j(z) &= 2zc_{j-1}(z) - c_{j-2}(z). \end{aligned}$$

With these polynomials we can construct new ones of he form:

$$p_k(z) = \frac{c_k(-1 + 2\frac{z-\alpha}{\beta-\alpha})}{c_k(1 + 2\frac{1-\beta}{\beta-\alpha})} \tag{2.5}$$

Which satisfies $p_k(1) = 1$ and leads to

$$||y^k - u||_2 \leq \frac{||u - u^0||_2}{|c_k(1 + 2\frac{1-\beta}{\beta-\alpha})|}$$

It is possible to find a three term recurrence from the Chebyshev recursive polynomials:

$$y^0 = u^0$$

solve z^0 from $Bz^0 = b - Ay^0$ then y^1 is given by

$$y^1 = y^0 + \frac{2}{2-\alpha-\beta} z^0$$

solve z^1 from $Bz^k = b - Ay^k$ then y^{k+1} is given by

$$y^{k+1} = \frac{4 - 2\beta - 2\alpha}{\beta - \alpha} \frac{c_k(1 + 2\frac{1-\beta}{\beta-\alpha})}{c_{k+1}(1 + 2\frac{1-\beta}{\beta-\alpha})} (y^k - y^{(k-1)} + \frac{2}{2-\alpha-\beta} z^k) + y^{(k-1)}.$$

2.3 Krylov subspaces

If we have two subspaces $\mathcal{K}_k, \mathcal{L}_k$ of R^n and we want to solve the equation $Ax = b$, with $A \in R^{n\times n}$ we can use a projection method onto $\mathcal{K}_k$. This method allows us, from an arbitrary initial condition x^0, to find an approximate solution x^k in the subspace $\mathcal{K}_k + x^0$, such that the residual $r^0 = b - Ax^0$ is orthogonal to the subspace $\mathcal{L}_k$:

$$x^k \in \mathcal{K}_k + x^0, \qquad r^k = b - Ax^k \perp \mathcal{L}_k. \tag{2.6}$$

From (2.4) we have the recurrence relation for an iterative method:

$$x^{k+1} = x^k + B^{-1}r^k, \qquad r^k = b - Ax^k.$$

With the initial guess solution x^0 we can compute the approximations as follows:

$$\begin{aligned}
x^1 &= x^0 + (B^{-1}r^0),\\
x^2 &= x^1 + (B^{-1}r^1) = x^0 + (B^{-1}r^0) + x^0 + B^{-1}(b - Ax^0 - AB^{-1}r^0)\\
&= x^0 + 2B^{-1}r^0 - B^{-1}AB^{-1}r^0,\\
&\vdots
\end{aligned}$$

We can see that this iterative method can be written as:

$$x^k \in x^0 + span\{B^{-1}r^0, B^{-1}A(B^{-1}r^0), \dots, (B^{-1}A)^{k-1}(B^{-1}r^0)\}.$$

If the x^k and the r^k satisfy (2.6), the x^k is in subspace $\mathcal{K}_k(A, r^0)$, which is called the Krylov subspace of dimension k, of the matrix A and residual r^0.

2.3.1 Arnoldi's Method

With the Arnoldi's method, we can construct an orthonormal basis $V = [v_1, v_2, \dots, v_k]$ from the Krylov subspace $\mathcal{K}_k(A, r^0)$. The vector v_i will be computed by performing the multiplication of the previous vector v_{i-1} with the matrix A, and the orthonormalization of the resulting vector w_i against the previous v's by the Gram-Schmidt method. The basis is computed as follows:

$$\begin{gathered}
h_{ij} = (Av_j, v_i) \qquad j = 1, 2, \dots, k, \qquad i = 1, 2, \dots, j,\\
w_j = Av_j - \sum_{i=1}^{j} h_{ij}v_i,\\
h_{j+1,j} = ||w_j||_2,\\
v_{j+1} = \frac{w_j}{h_{j+1,j}}.
\end{gathered}$$

The resulting matrix H with coefficients h_{ij} is a Hessenberg matrix whose eigenvalues can provide an approximation to the eigenvalues of the matrix A.

$$H = \begin{pmatrix} h_{11} & h_{12} & \dots & h_{1k} \\ h_{21} & h_{22} & \dots & h_{2k} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & h_{(k-1)k} \\ 0 & 0 & \dots & h_{mm} \end{pmatrix}.$$

The basis V can be constructed from the initial residual $r^0 = b - Ax^0$, taking the first vector v_1 as $v_1 = \frac{r^0}{||r^0||_2}$.

2.3.2 Lanczos Method

If the matrix A used in the Arnoldi's method is real and symmetric, the Hessenberg matrix H obtained is tridiagonal and symmetric T_m.

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ \dots & \dots & \dots & \dots & \dots \\ & & \beta_{k-1} & \alpha_{k-1} & \beta_m \\ & & & \beta_m & \alpha_k \end{pmatrix}.$$

Where

$$\alpha_j = h_{jj}, \qquad \beta_j = h_{j-1,j} = h_{j,j-1}.$$

2.3.3 Conjugate Gradient Method

If we have a *SPD* matrix A, we can construct a basis $V_k = [v_1, ..., v_k]$, where

$$x^{k+1} \in x^0 + \mathbf{K}_k(A, r^0),$$

$r^0 = b - Ax^0$, such that

$$||x - x^k||_A, ^2$$

is minimal. In this method we need to find search directions p^k, such that they are conjugated with respect to A, i.e., $(Ap^k, p^j)^3 = 0$, $k \neq j$, and the residuals form an orthogonal set, i.e., $(r^k, r^j) = 0$, $k \neq j$.
Given an initial guess x^0 the following iterations can be computed following the search direction p^i

$$x^{k+1} = x^k + \alpha_k p^k.$$

For the first iteration x^1, if we take $x^0 = 0$ for simplicity, we can compute the A-norm of $||x - x^1||_A^2$ as follows:

$$\begin{aligned} ||x - x^1||_A^2 = ||x - \alpha_0 p^0||_A^2 = (x - \alpha_0 Ap^0)^T(Ax - \alpha_0 Ap^0) = \\ = x^T Ax - 2\alpha_0 (p^0)^T Ax + \alpha_0^2 (p^0)^T A(p^0), \end{aligned}$$

which is minimal if $\frac{\partial ||x-x^1||_A^2}{\partial \alpha_0} = 0$, then

$$-2(p^0)^T Ax + 2\alpha_0 (p^0)^T A(p^0) = 0,$$

resulting in

$$\alpha_0 = \frac{(p^0)^T Ax}{(p^0)^T Ap^0} = \frac{(p^0)^T b}{(p^0)^T Ap^0} = \frac{(p^0)^T b}{(p^0, p^0)_A}$$

And the new search directions can be computed via the residuals,

$$p^{k+1} = r^{k+1} + \beta_k p^k,$$

where

$$\beta_k = \frac{(r^{k+1}, r^{k+1})}{(r^k, r^k)}.$$

After $k + 1$ iterations of the *CG* method, the error of the iteration will be bounded by

$$||x - x^{k+1}||_A \leq 2||x - x^0||_A \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^{k+1}.$$

[2] See eq. (1.3) for the A-norm definition.
[3] See eq. (1.2).

2.3.4 Preconditioning

The preconditioning transforms the original linear system, into a new one with the same solution and easier to solve with an iterative solver. For the preconditioning it is necessary to find a preconditioning matrix M. From (2.4) we can see that

$$x^{k+1} = M^{-1}(M - A)x^k + M^{-1}b,$$

This iteration can be regarded as the system:

$$(I - M^{-1}(M - A))x = M^{-1}(A)x = M^{-1}b.$$

Which has the same solution as the original one, and is called a *preconditioned system*, where M is the *preconditioning matrix.* This *preconditioning matrix* can be applied in three different ways:

- From the left
$$M^{-1}Ax = M^{-1}b$$
- To the right
$$AM^{-1}u = b, \qquad x \equiv M^{-1}u.$$
- Splitting the preconditioner. If the preconditioning matrix M can be factored in triangular matrices M_L, M_R:
$$M_L^{-1}AM_R^{-1}u = M_L^{-1}b, \qquad x \equiv M_R^{-1}$$

After $i+1$ iterations, the error of the iteration will be bounded by

$$||x - x^{i+1}||_A \leq 2||x - x^0||_A \left(\frac{\sqrt{\kappa(M^{-1}A)} - 1}{\sqrt{\kappa(M^{-1}A)} + 1} \right)^{i+1}.$$

2.3.5 Deflation

Given an SPSD matrix $A \in \mathbf{R}^{n\times n}$, the deflation matrix (P) is defined as follows:

$$P = I - AQ, \qquad P \in \mathbf{R}^{n\times n}, \qquad Q \in \mathbf{R}^{n\times n},$$

where

$$Q = ZE^{-1}Z^T, \qquad Z \in \mathbf{R}^{n\times k}, \qquad E \in \mathbf{R}^{k\times k},$$

with

$$E = Z^TAZ.$$

The matrix E is known as the *Galerkin* or $coarse$ matrix. The full rank matrix Z is called the $deflation - subspace$ matrix, and it's $k < n$ columns are the $deflation$ vectors or *projection* vectors. Some properties of the previous matrices are [**?**] :

a) $E^T = E, \qquad symmetric.$

b) $Q^T = Q = QAQ, \qquad symmetric.$

c) $QAZ = Z.$

d) $PAQ = \mathbf{0}_{n\times n}$.

e) $P^2 = P$.

f) $AP^T = PA$.

g) $(I - P^T)x = Qb$.

h) $PAZ = \mathbf{0}_{n\times k}$.

i) $P^T Z = \mathbf{0}_{n\times k}$.

j) PA is SPSD.

Deflated CG Method

If we have the linear system $Ax = b$, we can use the deflation matrix to find the solution of the system. We start from the expression $x = Ix - (P^T + P^T)x$, rearranging terms we get:

$$x = (I - P^T)x + P^T x = Qb + P^T x,$$

multiplying by A the previous expression and using the fact that $Q = I - P^T$, after some algebra, it becomes

$$b = AQb + PAx,$$

then

$$(I - AQ)b = Pb = PAx.$$

Which is not necessarily a solution of the original system, but a solution of the deflated system:

$$PA\hat{x} = Pb.$$

This system can be solved with the CG, for $\hat{x}$, the deflated solution, which satisfies $P^T\hat{x} = P^T x$, with x the solution of the linear system. This solution can be obtained from [**?**]:

$$x = Qb + P^T\hat{x}.$$

Deflated PCG Method

If the deflated linear system is preconditioned by a matrix M, such that

$$\tilde{A} = M^{-\frac{1}{2}}AM^{-\frac{1}{2}}, \qquad \hat{\tilde{x}} = M^{\frac{1}{2}}\hat{x}, \qquad \tilde{b} = M^{-\frac{1}{2}}b$$

The deflated preconditioned system that should be solved with the CG is:

$$\tilde{P}\tilde{A}\hat{\tilde{x}} = \tilde{P}\tilde{b},$$

where:

$$\tilde{P} = I - \tilde{A}\tilde{Q}, \qquad \tilde{Q} = \tilde{Z}\tilde{E}^{-1}\tilde{Z}^T, \qquad \tilde{E} = \tilde{Z}^T\tilde{A}\tilde{Z}.$$

This method is called the deflated preconditioned conjugate gradient *DPCG*, and the error is bounded by:

$$||x - x^{i+1}||_A \le 2||x - x^0||_A \left(\frac{\sqrt{\kappa(M^{-1}PA)} - 1}{\sqrt{\kappa(M^{-1}PA)} + 1}\right)^{i+1},$$

with $\kappa = \frac{\lambda_{max}(M^{-1}PA)}{\lambda_{min}(M^{-1}PA)}$, the effective condition number, and $\lambda_{min}(M^{-1}PA)$ the smallest non zero eigenvalue.
If M^{-1} have eigenvalues $\{\lambda_i\}$ with corresponding orthonormal eigenvectors $\{v_i\}$, and $Z = [v_1 v_2 ... v_k]$, some properties of the $DPCG$ method are:

1. $\tilde{P}\tilde{A}\tilde{v_i} = \begin{cases} 0, & i = 1, ..., k \\ \lambda_i \tilde{v_i}, & i = k+1, .., n \end{cases}$
2. $\sigma(M^{-1}PA) = \{0, .., 0, \lambda_{k+1}, ..., \lambda_n\}$
3. $\kappa(M^{-1}PA) \leq \kappa(M^{-1}A)$.

2.4 Choices of Deflation Vectors

The deflation method is used to treat the most unfavorable eigenvalues of $M^{-1}A$, by taking into account only the favorable eigenvalues, in such a way that the convergence is achieve faster. The choice of the matrix Z is of great importance for the deflation method, in the ideal case, it consist of the eigenvectors associated with the unfavorable eigenvalues. Sometimes, the computation of these eigenvectors can be very expensive and they can be inefficient in use. Therefore, is necessary to find deflation vectors that approximate the unfavorable eigenspace.
There is no optimal choice for the deflation vectors because it depends on each problem and the available information. Most of the techniques to choose deflation vectors are based in the choices given in the next section.

2.4.1 Approximated Eigenvector Deflation

As the name states, this technique is based on approximate vectors, that can be obtained in different ways. For example, these eigenvalues can be derived from the solutions of the original PDE's on specific subdomains [**?**].

2.4.2 Recycling Deflation

With this technique, a set of vectors previously used is reused to build the deflation-subspace matrix [**?**]. The vectors could be, for example, the first $q-1$ solution vectors of the linear system, and the matrix Z will be:

$$Z = [x^{(1)}, x^{(12)}, ..., x^{(q-1)}].$$

This method has the drawback that the matrix can be dense and can lead to implementation and memory difficulties.

2.4.3 Subdomain Deflation

Here the domain is divided into several subdomains, where each subdomain corresponds to one or more deflation vectors. For each subdomain, there is a deflation vector that has ones for points in the subdomain and zeros for points outside[**?**].

2.4.4 Multi Grid and Multilevel Deflation

For the multigrid and multilevel methods, there are matrices called prolongation and restriction matrices, that allow us to pass from one level or grid to another. These matrices can also be used as the deflation-subspace matrix [**?**].

Chapter 3

Partial Differential Equations

A differential equation is called partial if its dependent variable varies with two or more independent variables, for example:

$$-\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 0.$$

Where u is the dependent variable, and x, y are the independent. The partial differential equations can be classified by their order: first order if they only have the first partial derivative, second order if they have the second partial, etc. They can also be classified as *linear* or *nonlinear*. The equation is *linear*, if the coefficients are constant or function of the independent variables.

$$u^m \left(\frac{\partial^n u}{\partial x^n}\right)^r, \qquad m = 0, \qquad r = 1.$$

3.1 Types of Equations

The second order partial differential equations can be written in a general form as:

$$a_{11}\frac{\partial^2 u(x,y)}{\partial x^2} + 2a_{12}\frac{\partial^2 u(x,y)}{\partial x \partial y} + 2a_{22}\frac{\partial^2 u(x,y)}{\partial y^2} + b_1\frac{\partial u(x,y)}{\partial x} + b_2\frac{\partial u(x,y)}{\partial y} + cu(x,y) = f$$

This equation can be classified based in the sign of the determinant D,

$$D = \begin{vmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}^2.$$

- If $D = 0$ the differential equation is parabolic.
- If $D < 0$ the differential equation is hyperbolic.
- If $D > 0$ the differential equation is elliptic.

The conditions that can be used to solve the partial differential equations can be of two types:

- The Neumann boundary condition: $\nabla u(\mathbf{x}) \cdot n = f(\mathbf{x})$.
- The Dirichlet boundary conditions: $u(\mathbf{x}) = g(\mathbf{x})$.

If the function $f(\mathbf{x})$ or $g(\mathbf{x})$ are zero, the boundary conditions are called homogeneous.

3.2 Finite Difference Discretization

The method of finite difference discretization is used to solve elliptic partial differential equations ($D > 0$). In this method, the domain Ω is discretized with a mesh of $N+1$, $(N+1)^2$ or $(N+1)^3$ nodes, for $1D, 2D$ and $3D$, respectively. The size of the mesh is $h = 1/N$. The partial differential equation is transformed into a system of algebraic equations with the help of the Taylor series expansion. Some theory concerning the transformation is given in the next section.

3.2.1 Numerical Differentiation

The derivative of a function $f(x)$ in a point x_1 is given by:

$$\frac{df(x)}{dx} = \lim_{\Delta x \to 0} \frac{f(x_1 + \Delta x) - f(x_1)}{\Delta x}.$$

However, most of the functions are complicated and their exact derivatives cannot be determined easily. The Taylor's series are used to find an approximate derivative of this functions. The Taylor's series for a function $f(x)$ at a point x_{i+1} is:

$$f(x_{i+1}) = f(x_i) + hf'(x_i) + \frac{h^2}{2!}f''(x_i) + \dots, \tag{3.1}$$

where h is the distance between x_i and x_{i+1}. The derivative of the function at x_i can be obtained from (3.1) as:

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{h}{2!}f''(x_i) + \dots,$$

which is called first forward divided differences. If we consider the error of order h as $O(h)$,

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} + O(h).$$

In a similar way, using the Taylor's series for the point x_{i-1}, the first order derivative can be written as:

$$f(x_{i-1}) = f(x_i) - hf'(x_i) + \frac{h^2}{2!}f''(x_i) + \dots \tag{3.2}$$

and

$$f'(x_i) = \frac{f(x_{i-1}) - f(x_i)}{h} + O(h).$$

Which is the first backward divided differences with error $O(h)$. The central divided differences, which has an error of order h^2 is obtained by subtracting (3.2) from (3.1) and is:

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} + O(h^2). \tag{3.3}$$

High order derivatives of $f(x)$ can be derived in a similar way. The second order derivatives with central divided differences have an error of order h^2 and is obtained as:

$$f''(x_i) = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2} + O(h^2). \tag{3.4}$$

This approximations are used for the discretization of the finite differences method, taking the central divided differences scheme, where the point x_i represents the i point of the grid, and the x_{i+1}, x_{i-1} points are grid points located a distance h from the i point.

3.2.2 Poisson's Equation

The Poisson's equation in $2D$ is given by:

$$-\left(\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2}\right) = f(x,y). \tag{3.5}$$

If the function $f(x,y) = 0$ this equation is called Laplace's equation. In the previous section is mentioned that the second derivative of a function $u(x)$ can be written as (3.4). If the function depends on two independent variables, the partial derivative with respect to each variable can be approximated by:

$$\frac{\partial^2 u(x,y)}{\partial x^2} \simeq \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}$$
$$\frac{\partial^2 u(x,y)}{\partial y^2} \simeq \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}$$

with

$$u_{i,j} = u(x_i, y_j).$$

If we substitute the second derivatives in the Poisson's equation (3.5) we get:

$$-\frac{u_{i+1,j} + u_{i,j+1} - 4u_{i,j} + u_{i-1,j} + u_{i,j-1}}{h^2} = f_{i,j}.$$

For the Dirichlet boundary, the values of u are given, so it's only necessary to compute the solution for the interior nodes. If the interior nodes are $M = N - \Gamma$, where Γ are the nodes in the boundary, we can number these interior nodes from 1 to M, and transform this problem into a linear system of the form $Ax = b$, where

$$A = \begin{pmatrix} 4 & -1 & 0 & 0 & \dots & -1 & \dots & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & \dots & 0 & -1 & \dots & 0 & 0 \\ \vdots & & & & & & & & & \\ 0 & -1 & \dots & -1 & 4 & -1 & \dots & -1 & \dots & 0 \\ \vdots & & & & & & & & & \\ 0 & 0 & \dots & -1 & 0 & \dots & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & \dots & -1 & 0 & \dots & 0 & -1 & 4 \end{pmatrix}, \qquad b = \begin{pmatrix} f_{1,1} \\ f_{2,2} \\ f_{3,3} \\ \vdots \\ f_{M-1,M-1} \\ f_{M,M} \end{pmatrix}.$$

And the boundary conditions will be impose in the nodes that are related with the boundary nodes. For the Dirichlet's conditions we have:

$$\begin{aligned} u(0,y) &= \Gamma_{0,y} \\ u(L,y) &= \Gamma_{L,y} \\ u(x,0) &= \Gamma_{x,0} \\ u(x,L) &= \Gamma_{x,L}. \end{aligned}$$

These conditions will change the side of the equation that contains b for the interior nodes (i,j) that have a boundary node as neighbor. For these terms, the boundary condition will be added to the function f_{ij} For the Neumann conditions $(\frac{\partial u}{\partial x_i}.n)$:

$$\frac{\partial u}{\partial x} = \nu_{i,j}$$

We need to approximate the normal derivative of the boundary points with the central difference scheme. For a point in the boundary i,j and an interior point $i-1,j$ and an imaginary point $i+1,j$, we have:

$$\frac{\partial u}{\partial x} \simeq \frac{u_{i+1,j} - u_{i-1,j}}{2h}, \qquad O(h^2)$$

The temperature at the point $u_{i+1,j} = 2h\frac{\partial u_{i,j}}{\partial x} + u_{i-1,j} = 2h\nu_{i,j} + u_{i-1,j}$. For the boundary point i,j the approximate solution is:

$$\frac{(2h\nu_{i,j} + u_{i-1,j}) + u_{i,j+1} - 4u_{i,j} + u_{i-1,j} + u_{i,j-1}}{h^2} = f_{i,j}.$$

That can be rewritten as:

$$\frac{u_{i,j+1} - 4u_{i,j} + 2u_{i-1,j} + u_{i,j-1}}{h^2} = f_{i,j} - 2\nu_{i,j}.$$

Chapter 4

Concepts in Reservoir Engineering

4.1 Petroleum Reservoir

Petroleum reservoirs are layers of sedimentary rock, which vary in terms of their grain size, mineral and clay contents. These rocks contain grains and empty space, the void is called pore space. The pore space allows the rock to store and transmit fluids. The volume fraction of the rock that is void is called *rock porosity* (ϕ).
Some rocks are compressible and their porosity depends on the pressure, this dependence is called *rock compressibility* (c_r). The ability of the rock to transmit a single fluid when the void space is completely filled with fluid is known as *rock permeability* (K).
Reservoir simulation is a way to analyze and predict the fluid behavior in a reservoir by the analysis of its behavior in a model. The description of subsurface flow simulation involves two types of models: mathematical and geological models. The geological model is used to describe the reservoir, i.e., the porous rock formation. The mathematical modeling is performed taking into account mass conservation and Darcy's law, corresponding to the momentum conservation. The equations used to describe single-phase flow through a porous medium are:

$$\frac{\partial(\rho\phi)}{\partial t} + \nabla\cdot(\rho v) = q, \qquad v = -\frac{K}{\mu}(\nabla p - \rho g \nabla z), \tag{4.1}$$

or

$$\frac{\partial(\rho\phi)}{\partial t} - \nabla\cdot\left(\frac{\rho K}{\mu}(\nabla p - \rho g \nabla z)\right) = q. \tag{4.2}$$

Where the primary unknown is the pressure p, g is the constant of gravity, d is the reservoir depth, ρ and μ are the fluid density and viscosity and q are the sources. The fluid density $\rho = \rho(p)$ and the rock porosity $\phi = \phi(p)$ can be pressure dependent. Rock porosity is related to the pressure via the rock compressibility. The relation is given by the following expression:

$$c_r = \frac{1}{\phi}\frac{d\phi}{dp} = \frac{d(ln(\phi))}{dp},$$

If the rock compressibility is constant, the previous equation is integrated as:

$$\phi(p) = \phi_0 e^{c_r(p-p_0)}. \tag{4.3}$$

The fluid density and the pressure are related via the fluid compressibility c_f, the relation is given by:

$$c_f = \frac{1}{\rho}\frac{d\rho}{dp} = \frac{d(ln(\rho))}{dp}.$$

If the fluid compressibility is constant, the previous equation is integrated as:

$$\rho(p) = \rho_0 e^{c_f(p-p_0)}. \tag{4.4}$$

To solve Equation (4.2), it is necessary to supply conditions on the boundary of the domain. For parabolic equations, we also need to impose initial conditions. Boundary and initial conditions will be discussed later for each problem.

Incompressible fluid

If the density and the porosity are not pressure-dependent in Equation (4.2), we have an incompressible model, where density and porosity do not change over time. Therefore, the incompressible mode is time-independent. Assuming no gravity terms and a fluid with constant viscosity, Equation (4.2) becomes:

$$-\frac{\rho}{\mu}\nabla \cdot (K\nabla p) = q. \tag{4.5}$$

Discretization

The spatial derivatives are approximated using a finite difference scheme with cell central differences. For a 3D model, taking a mesh with a uniform grid size Δx, Δy, Δz where (i, j, l) is the center of the cell in the position i in the x direction and j in the y direction and l in the z direction (x_i, y_j, z_l) and $p_{i,j,l} = p(x_i, y_j, z_l)$ is the pressure at this point.

For the x direction, we have (see [**?**]):

$$\begin{aligned}\frac{\partial}{\partial x}\left(k\frac{\partial p}{\partial x}\right) &= \frac{\Delta}{\Delta x}\left(k\frac{\Delta p}{\Delta x}\right) + \mathcal{O}(\Delta x^2)\\ &= \frac{k_{i+\frac{1}{2},j,l}(p_{i+1,j,l} - p_{i,j,l}) - k_{i-\frac{1}{2},j,l}(p_{i,j,l} - p_{i-1,j,l})}{(\Delta x)^2} + \mathcal{O}(\Delta x^2),\end{aligned}$$

where $k_{i-\frac{1}{2},j,l}$ is the harmonic average of the permeability for cells $(i-1, j, l)$ and (i, j, l):

$$k_{i-\frac{1}{2},j,l} = \frac{2}{\frac{1}{k_{i-1,j,l}} + \frac{1}{k_{i,j,l}}}. \tag{4.6}$$

After discretization, Equation (4.5), together with boundary conditions, can be written as:

$$\mathbf{Tp} = \mathbf{q}, \tag{4.7}$$

where $\mathbf{T}$ is known as the transmissibility matrix with elements in adjacent grid cells. The *transmissibility* $(T_{i-\frac{1}{2},j,l})$ between grid cells $(i-1, j, l)$ and (i, j, l) is defined as [**?**]:

$$T_{i-\frac{1}{2},j,l} = \frac{2\Delta y\Delta z}{\mu\Delta x}k_{i-\frac{1}{2},j,l}, \tag{4.8}$$

System (4.7) is a linear system that can be solved with iterative or direct methods. For the solution of this system, it is necessary to define boundary conditions in all boundaries of the domain. These conditions can be prescribed pressures (Dirichlet conditions), flow rates (Neumann conditions) or a combination of these (Robin conditions).

Compressible fluid
If the fluid is compressible with a constant compressibility, the density depends on the pressure (see Equation (4.4)). Therefore, Equations (4.1) become:

$$\frac{\partial(\rho(p)\phi)}{\partial t} + \nabla \cdot (\rho(p)v) = q, \qquad v = -\frac{K}{\mu}(\nabla p - \rho(p)g\nabla z), \tag{4.9}$$

Discretization
Using backward Euler time discretization, Equations (4.9) are approximated by:

$$\frac{(\phi\rho(p))^{n+1} - (\phi\rho(p))^n}{\Delta t^n} + \nabla\cdot(\rho(p)v)^{n+1} = q^{n+1}, \qquad v^{n+1} = -\frac{K}{\mu^{n+1}}(\nabla(p^{n+1}) - g\rho^{n+1}\nabla z). \tag{4.10}$$

Assuming no gravity terms, constant fluid viscosity and constant rock porosity, Equations (4.10) become:

$$\phi\frac{\rho(p^{n+1}) - \rho(p^n)}{\Delta t^n} - \frac{1}{\mu}\nabla \cdot (\rho(p^{n+1})K\nabla p^{n+1}) + q^{n+1} = 0. \tag{4.11}$$

Due to the dependence of ρ on the pressure, the latter is a nonlinear equation for p that can be linearized with, e.g., the Newton-Raphson (NR) method. Equation (4.11) can be discretized in space, using a finite differences scheme. After spatial discretization, Equation (4.11) reads:

$$\phi\frac{\rho(\mathbf{p}^{n+1}) - \rho(\mathbf{p}^n)}{\Delta t^n} - \frac{1}{\mu}\nabla \cdot (\rho(\mathbf{p}^{n+1})\mathbf{K}\nabla \mathbf{p}^{n+1}) + \mathbf{q}^{n+1} = 0. \tag{4.12}$$

As in the incompressible case, we need to define boundary condition to solve Equation (4.12). Dirichlet, Neumann or Robin boundary conditions can be used. For this problem, we also have a derivative with respect to time. Therefore, it is also necessary to specify the initial conditions that are the pressure values of the reservoir at the beginning of the simulation.

Well model
In reservoirs, wells are typically drilled to extract or inject fluids. Fluids are injected into a well at constant surface rate or constant bottom-hole pressure (bhp) and are produced at constant bhp or a constant surface rate.
When the bhp is known, some models are developed to accurately compute the flow rate into the wells. A widely used model is Peaceman's model, that takes into account the bhp and the average grid pressure in the block containing the well. This model is a linear relationship between the bhp and the surface flow rate in a well, for a cell (i, j, l) that contains a well, this relationship is given by:

$$q_{(i,j,l)} = I_{(i,j,l)}(p_{(i,j,l)} - p_{bhp(i,j,l)}), \tag{4.13}$$

where $I_{(i,j,l)}$ is the productivity or injectivity index of the well, $p_{(i,j,l)}$ is the reservoir pressure in the cell where the well is located, and $p_{bhp(i,j,l)}$ is a prescribed pressure inside the well.

Incompressible fluid
Using the well model for an incompressible fluid, Equation (4.7) transforms into:

$$\mathbf{T}\mathbf{p} = \mathbf{I}_w(\mathbf{p} - \mathbf{p}_{bhp}). \tag{4.14}$$

Where $\mathbf{I}_w$ is a vector containing the productivity or injetivity indices of the wells present in the reservoir. It is zero for cells without wells and the value of the well index for each cell containing a well.

Compressible fluid
For a compressible fluid, using the well model, Equation (4.12) reads:

$$\phi\frac{\rho(\mathbf{p}^{n+1}) - \rho(\mathbf{p}^n)}{\Delta t^n} - \frac{1}{\mu}\nabla\cdot(\rho(\mathbf{p}^{n+1})\mathbf{K}\nabla\mathbf{p}^{n+1}) + \mathbf{I}_w^{n+1}(\mathbf{p}^{n+1} - \mathbf{p}_{bhp}^{n+1}) = 0. \tag{4.15}$$

Solution procedure for compressible flow
As mentioned before, for the compressible problem, we have a nonlinear system that depends on the pressure at the time step n and the pressure at the time step $n+1$. Therefore, Equation (4.15) can be seen as a function that depends on $\mathbf{p}^{n+1}$ and $\mathbf{p}^n$, i.e.,

$$\mathbf{F}(\mathbf{p}^{n+1}; \mathbf{p}^n) = 0. \tag{4.16}$$

This nonlinear system can be solved with the NR method, the system for the $(k+1)$-th NR iteration is:

$$\mathbf{J}(\mathbf{p}^k)\delta\mathbf{p}^{k+1} = -\mathbf{F}(\mathbf{p}^k; \mathbf{p}^n), \qquad \mathbf{p}^{k+1} = \mathbf{p}^k + \delta\mathbf{p}^{k+1},$$

where $\mathbf{J}(\mathbf{p}^k) = \frac{\partial\mathbf{F}(\mathbf{p}^k;\mathbf{p}^n)}{\partial\mathbf{p}^k}$ is the Jacobian matrix, and $\delta\mathbf{p}^{k+1}$ is the NR update at iteration step $k+1$.
Therefore, the linear system to solve is:

$$\mathbf{J}(\mathbf{p}^k)\delta\mathbf{p}^{k+1} = \mathbf{b}(\mathbf{p}^k). \tag{4.17}$$

with $\mathbf{b}(\mathbf{p}^k)$ being the function evaluated at iteration step k, $\mathbf{b}(\mathbf{p}^k) = -\mathbf{F}(\mathbf{p}^k; \mathbf{p}^n)$.
The procedure to solve a compressible flow problem consists of three stages. During the first stage, we select a time and solve Equation (4.15) for this particular time, i.e., we have a solution for each time step. In the second stage, we linearize the equations with the NR method, i.e., we perform a series of iterations to find the zeros of Equation (4.16). For every NR iteration the linear system in Equation (4.17) is solved. In this work, the solution of the linear system is performed with iterative methods (see Section **??**). A summary of this procedure is presented in Algorithm 1.

Mass Conservative formulation.

The reservoir simulation are usually run to predict the recoverable hydrocarbon volumes, in such a case is necessary to represent accurately the mass conservation equation. The equation (**??**) can be interpreted as the mass balance equation for a grid point i, if the rows are added, we get:

$$\sum_{i=1}^{n_x}\sum_{j=1}^{n_y}[Vc_t(\phi_0)_{i,j}\dot{p_{i,j}} + q_{i,j}], \tag{4.18}$$

Algorithm 1

```
for $t = 0, ...,$                                              %Time integration
    Select time step
  for $NR\_iter = 0, ...,$                                     %NR iteration
      Find zeros of $\mathbf{F}(\mathbf{p}^{n+1}; \mathbf{p}^n) = 0$
    for $lin\_iter = 0, ...,$                                  %Linear iteration
        Solve $\mathbf{J}(\mathbf{p}^k)\delta\mathbf{p}^{k+1} = \mathbf{b}(\mathbf{p}^k)$ for each NR iteration
    end
  end
end
```

which can be interpreted as the mass-balance equation for the system. Therefore any mass-balance error in the numerical solution results from errors in the accumulation terms $Vc_t(\phi_0)_{i,j}\dot{p_{i,j}}$. The discretization of the accumulation term can be done in the form:

$$Vc_t(\phi_0)\frac{\partial p}{\partial t} = Vc_t(\phi_0)\frac{p_{k+1} - p_k}{\Delta t}. \tag{4.19}$$

But this discretization, in general, is not mass conservative. For mass conservative discretization it is necessary to take the original accumulation term:

$$\frac{\partial(\rho\phi)}{\partial t} = \frac{1}{\Delta t}\left[\frac{1}{\phi}\frac{\partial \phi}{\partial p} + \frac{1}{\rho}\frac{\partial \rho}{\partial p}\right]\frac{\partial p}{\partial t},$$

'Comparison with equation (1.38) shows that the constant coefficient Vct [U+F066] 0 has been replaced by a state-dependent coefficient, which, moreover, contains an element [U+F072] k+1 that should be computed at the new time step k+1. A mass-conservative implementation therefore always requires some form of implicit time integration. For liquid flow, and as long as the pressure changes in the reservoir remain small compared to the total pressure, the effect of mass-balance errors is small, and therefore we do not make use of the strict mass-conserving formulation in our numerical examples. However if compressibility plays a role, e.g. when free gas is present, the use of a mass-conservative formulation is essential.'

4.1.1 Two-phase flow.

When simulating two-phases within a porous medium, we often consider them as separated phases, i.e., they are inmiscible and there is no mass transfer between the phases. The contact between phases is known as the interface between two reservoir fluids phases.
While modeling two phases we usually consider one of the fluids as the wetting phase (w) that wets the porous medium more than the other. The other phase is consider as non-wetting phase (n). In the case of a water-oil system, water is often the wetting phase.
For a reservoir, the density of each phase varies, the gas is less dense than the oil and water. Therefore, the gas is found on top of the reservoir. For the oil-water system, water's density is larger than oil's, then water is found on the bottom of the reservoir.
The saturation of a phase S_α, is the fraction of void space filled with that phase in a porous medium. If there are two phases present in the porous medium, these fluids fill completely the empty space, which is expressed by the following relation.

$$S_n + S_w = 1 \tag{4.20}$$

The surface tension and the curvature of the interface between the fluids causes a difference in pressure between the two phases. The pressure in the wetting fluid is less than in the nonwetting fluid. This difference in pressures is known as the capillary pressure, p_c, and it's function of saturation:

$$p_c(S_w) = p_n - p_w. \tag{4.21}$$

As mentioned before, the pressure in the non-wetting fluid is higher than the pressure in the wetting fluid, therefore the capillary pressure is always positive. The relation between the capillary pressure and saturation is obtained as an empirical model based on experiments. However, each core sample will have different capillary curve due to the difference in pore-size distributions, porosity and permeability. To normalize the measured data, it's common to use a so-called Leverett J-function, which takes the following form:

$$J(S_w) = \frac{P_c}{\sigma cos\theta}\sqrt{\frac{K}{\phi}}, \tag{4.22}$$

where σ is the surface tension and θ the contact angle measured in the laboratory for an specific rock and fluid system. A model that relates the capillary pressure and water saturation in a partially saturated media (water and air media) is:

$$\hat{S}_w = \begin{cases} (p_c/p_e)^{-n_b} & \text{if } p_c > p_e \\ 1 & p_c \leq p_e \end{cases}$$

where p_e is the entry pressure of air, and n_b is related to the pore-size distribution. This model was proposed by Brooks and Corey. Another model was proposed by Genuchten:

$$\hat{S}_w = \left(1 + (\beta_g p_c)_g^n)^{-m_g}\right), \tag{4.23}$$

where β_g is related to the average size of pores and n_g and m_g are related to the pore size distribution.

Relative permeability
When more than one phase is present in the pore space, each phase α will experience an effective permeability K^e_αthat is less than the absolute permeability K. Due to interfacial tensions, the sum of all the phase permeabilities is less than one.

$$\sum_\alpha K^e_\alpha < K.$$

The relative permeability for an isotropic medium is defined as: $k_{r\alpha} = K^e_\alpha / K$. The relative permeabilities will generally be functions of saturation, generally nonlinear. It is common to use analytic relationships to represent relative permeabilities. These are usually stated using normalized or effective saturations $\hat{S}_w$. The simplest model possible is called the Corey model:

$$\begin{aligned} k_{rw} &= (\hat{S}_w)^{n_w} k^0_w, \\ k_{ro} &= (1 - \hat{S}_w)^{n_n} k^0_o. \end{aligned} \tag{4.24}$$

where $n_w > 1$, $n_o > 1$ and k^0_α are fitting parameters. The Brooks-Corey functions are also used.

$$\begin{aligned} k_{rw} &= (\hat{S}_w)^{n_1 + n_2 n_3}, \\ k_{ro} &= (1 - \hat{S}_w)^{n_1} [1 - (\hat{S}_w)^{n_2}]^{n_3}. \end{aligned}$$

where, $n_1 = 2$, $n_2 = 1 + 2/n_b$ and $n4 = 1$ for the Brooks–Corey–Burdine model and $n_1 = \eta$, $n_2 = 1 + 1/n_b$ and $n3 = 2$ for the Brooks–Corey–Mualem model. For the van Genuchten capillary functions $m_g = 1 - 1/n_g)$ we have:

$$\begin{aligned} k_{rw} &= \hat{S}^2_w [1 - (1 - \hat{S}^{1/m_g}_w)^{m_g}], \\ k_{ro} &= (1 - \hat{S}_w)^2 [1 - (\hat{S}^{1/m_g}_w)]^{m_g}, \end{aligned}$$

which is called the Genuchten-Burdine model.
As in the single-phase case, the governing equations for two-phase flow in a porous medium are the mass conservation and Darcy's law. The mass balance equations for each phase are given by:

$$\frac{\partial(\phi \rho_\alpha S_\alpha)}{\partial t} + \nabla \cdot (\rho_\alpha \mathbf{v}_\alpha) = \rho_\alpha q_\alpha,$$

Darcy's law is:

$$\mathbf{v}_\alpha = -\frac{k_{r\alpha}}{\mu_\alpha} K (\nabla p_\alpha - \rho_\alpha g \nabla z).$$

To simplify notation, the phase mobilities ($\lambda_\alpha = K k_{r\alpha}/\mu_\alpha$) or relative phase mobilities ($\lambda_\alpha = \lambda_\alpha K$) are used. Using the discrete derivative operators and backward discretization of temporal derivatives, the resulting system of discrete equations is:

$$\frac{(\phi \rho_\alpha \mathbf{S}_\alpha)^{n+1} - (\phi \rho_\alpha \mathbf{S}_\alpha)^n}{\Delta t^n} + div(\rho_\alpha \mathbf{v}_\alpha)^{n+1} = \mathbf{q}^{n+1}_\alpha,$$

$$\mathbf{v}^{n+1}_\alpha = -\frac{\mathbf{k}_{r\alpha}}{\mu_\alpha} \mathbf{K}[grad(\mathbf{p}^{n+1}_\alpha) - g \rho^{n+1}_\alpha grad(\mathbf{z})].$$

Immiscible two-phase flow
For the case of immiscible fluids, the flow equations are:

$$\frac{\partial(\phi\rho_w S_w)}{\partial t} + \nabla\cdot(\rho_w \mathbf{v}_w) = \rho_w q_w, \tag{4.25}$$

$$\frac{\partial(\phi\rho_n S_.)}{\partial t} + \nabla\cdot(\rho_n \mathbf{v}_n) = \rho_n q_n. \tag{4.26}$$

Pressure formulation
We can choose the pressures as the primary unknowns, then the saturations are expressed as functions of pressure. Assuming the capillary pressure has a unique inverse function $\hat{S}_w = P_c^{-1}(p_c)$, then we have:

$$S_w = \hat{S}_w(p_n - p_w) \qquad S_n = 1 - \hat{S}_w(p_n - p_w),$$

then equations (4.25) and (4.26) can be written as:

$$\frac{\partial(\phi\rho_w \hat{S}_w)}{\partial t} + \nabla\cdot(\rho_w \frac{Kk_{rw}(\hat{S}_w)}{\mu_w}(\nabla p_w - \rho_w g\nabla z)) = \rho_w q_w, \tag{4.27}$$

$$\frac{\partial(\phi\rho_n(1-\hat{S}_w))}{\partial t} + \nabla\cdot(\rho_n \frac{Kk_{rn}(\hat{S}_w)}{\mu_n}(\nabla p_n - \rho_n g\nabla z)) = \rho_n q_n. \tag{4.28}$$

Previous system is highly coupled and strongly nonlinear.
Fractional flow formulation
Part of the non linearity of previous formulation can be eliminated if the system is expressed in terms of one phase pressure and one phase saturation. A common choice is to use p_n and S_w which gives the following system

$$\frac{\partial(\phi\rho_w S_w)}{\partial t} + \nabla\cdot(\rho_w \frac{Kk_{rw}}{\mu_w}(\nabla p_n - \nabla P_c(S_w) - \rho_w g\nabla z)) = \rho_w q_w, \tag{4.29}$$

$$\frac{\partial(\phi\rho_n(1-S_w))}{\partial t} + \nabla\cdot(\rho_n \frac{Kk_{rn}}{\mu_n}(\nabla p_n - \rho_n g\nabla z)) = \rho_n q_n. \tag{4.30}$$

Incompressible flow
For incompressible flow, the porosity is only a function of time and the fluid densities ρ_α are constant. Therefore, the mass-balance equations are:

$$\phi\frac{\partial(S_\alpha)}{\partial t} + \nabla\cdot(\mathbf{v}_\alpha) = q_\alpha. \tag{4.31}$$

The total Darcy velocity is defined as the sum of the velocity in the wetting and non wetting phases, it can be expressed in terms of the non-wetting phase as:

$$\begin{aligned}\mathbf{v} = \mathbf{v}_w + \mathbf{v}_n &= -\lambda_n\nabla p_n - \lambda_w\nabla p_w + (\lambda_n\rho_n + \lambda_w\rho_w)g\nabla z\\ &= -(\lambda_n + \lambda_w)\nabla p_n + \lambda_w\nabla p_c + (\lambda_n\rho_n + \lambda_w\rho_w)g\nabla z\end{aligned}$$

if we add the two continuity equations and use the relationship $S_n + S_w = 1$ we have:

$$\phi\frac{\partial(S_w + S_n)}{\partial t} + \nabla\cdot(\mathbf{v}_w + \mathbf{v}_n) = q_n + q_w.$$

Using $\lambda = \lambda_n + \lambda_w = \lambda K$ as the total mobility and $q = q_n + q_w$ as the total source, for the total Darcy velocity we have:

$$-\nabla \cdot (\lambda K \nabla p_n) = q - \nabla[\lambda_w \nabla p_c + (\lambda_n \rho_n + \lambda_w \rho_w) g \nabla z].$$

Multiplying each phase velocity by the relative mobility of the other phase and subtracting the result we obtain:

$$\begin{aligned} \lambda_n \mathbf{v}_w - \lambda_w \mathbf{v}_n &= \lambda \mathbf{v}_w - \lambda_w \mathbf{v} \\ &= \lambda_w \lambda_n K[\nabla p_c + (\rho_w - \rho_n) g \nabla z]. \end{aligned}$$

Therefore, for $\mathbf{v}_w$ we have

$$\mathbf{v}_w = \frac{\lambda_w}{\lambda}\mathbf{v} + \frac{\lambda_w \lambda_n}{\lambda} K[\nabla p_c + (\rho_w - \rho_n) g \nabla z].$$

Using the velocity computed above, for the wetting phase we have:

$$\phi \frac{\partial(S_w)}{\partial t} + \nabla \cdot [f_w(\mathbf{v} + \lambda_n \Delta\rho g \nabla z)] = q_w - \nabla \cdot (f_w \lambda_n p_c \nabla S_w). \tag{4.32}$$

with $\Delta\rho = \rho_w - \rho_n$ and the fractional flow function f_w:

$$f_w = \frac{\lambda_w}{\lambda_n + \lambda_w},$$

is the fraction of the total flow that consists of the wetting fluid, and

$$\mathbf{v} = -\lambda(\nabla p_n - f_w \nabla p_w - (f_n \rho_n + f_w \rho_w) g \nabla z)$$

The coupling between the elliptic pressure equation and the parabolic saturation equation is much weaker than the coupling between the two continuity equations in the two-pressure formulation. In the pressure equation, the coupling to saturation appears explicitly in the effective mobility that makes up the variable coefficient in the Poisson problem and on the right-hand side through the phase mobilities and the derivative of the capillary function. In Equation (4.32), the saturation is only indirectly coupled to the pressure through the total Darcy velocity. incompressible fluids.
The system of PDEs can then be reformulated so that it consists of an elliptic equation for fluid pressure and one or more trans- port equations. These transport equations are generally parabolic, but have a strong hyperbolic character. Since the pressure and saturations equations have very different mathematical characteristics, it is natural to solve them in consecutive substeps. Sequential solution procedures and in- compressible flow models are popular in academia and for research purposes, but are less used in industry. To the extent simulations are used for practical reservoir engineering, they are mainly based on compressible equations and solution procedures in which flow and transport are solved as a coupled system. Such approaches are very robust and particularly useful for problems with large variations in time constants or strong coupling between different types of flow mechanisms.

Sequential solution procedures

The two-phase, incompressible model will be solved using the fractional-flow formulation. This fractional flow model consists of an elliptic pressure equation

$$\nabla \cdot \mathbf{v} = q, \qquad \mathbf{v} = -\lambda(\nabla p_n - f_w \nabla p_c - (f_n \rho_n + f_w \rho_w) g \nabla z) \tag{4.33}$$

and a parabolic transport equation (4.32)

$$\phi\frac{\partial(S_w)}{\partial t} + \nabla\cdot[f_w(\mathbf{v} + \lambda_n(\Delta\rho g\nabla z) + \nabla P_c)] = q_w. \tag{4.34}$$

Where the capillary pressure $pc = p_w - p_n$ is assumed to be a known function P_c of the wetting saturation S_w , and the transport equation becomes hyperbolic whenever P_c' is zero.
In the standard sequential solution procedure, the system above, is evolved in time using a set of discrete time steps Δt_i . We assume that $p, \mathbf{v}$, and S_w are all known at time t and that we want to evolve the solution to time $t + \Delta t$.
At the beginning of the time step, we first assume that the saturation S_w is fixed. This means that the parameters λ, f_w, and f_n are functions of the spatial variable $\mathbf{x}$, and hence we can use this equation only to update the pressure p_n and the Darcy velocity $\mathbf{v}$. Then, $\mathbf{v}$ and p_n are held fixed while equation 4.34 is evolved a time step Δt to define an updated saturation $S_w(\mathbf{x}, t + \Delta t)$. This saturation is then held fixed when we update p_n and $\mathbf{v}$ in the next time step, and so on. Some authors refer to this solution procedure as an operator splitting method since the solution procedure effectively splits the overall solution operator of the flow model into two parts that are evolved in consecutive substeps. Likewise, some authors refer to the sequential solution procedure as IMPES, which is short-hand for implicit pressure, explicit saturation. Using the name IMPES is strictly speaking only correct if the saturation evolution is approximated by a single time step of an explicit transport solver.

Pressure solvers
The pressure Equation (4.33) for incompressible, multiphase flow is time dependent. This time dependence comes as the result of three factors:

1. K/μ being replaced by the total mobility $\lambda(S_w)$, which depends on time through the saturation $S_w(\mathbf{x}, t + \Delta t)$,
2. the constant density ρ being replaced by a saturation-dependent quantity $\rho_w f_w(S_w) + \rho_n f_n(S_w)$, and
3. the source term q being replaced by a saturation-dependent source term $q - \nabla\lambda_w(S_w)\nabla P_c(S_w)$.

However, once S_w is held fixed in time, all three quantities become functions of $\mathbf{x}$ only, and we hence end up again with an elliptic Poisson-type equation having the same spatial variation.
Saturation solvers
The saturation equation depends on the time, using backward Euler discretization for the time derivative in Equation 4.34, we have:

$$\phi\frac{(S_w^{n+1} - S_w^n)}{\Delta t} + \nabla\cdot[f_w(S_w)(\mathbf{v} + \lambda_n(S_w)(\Delta\rho g\nabla z + \nabla P_c(S_w)))] = q_w, \tag{4.35}$$

or

$$S_w^{n+1} = S_w^n - \frac{\Delta t}{\phi}\nabla\cdot[f_w(S_w)(\mathbf{v} + \lambda_n(S_w)(\Delta\rho g\nabla z + \nabla P_c(S_w)))] + q_w,$$

which can be computed explicitly:

$$S_w^{n+1} = S_w^n - \mathcal{F}(S_w^n, S_w^n),$$

or implicitly:

$$S_w^{n+1} = S_w^n - \mathcal{F}(S_w^{n+1}, S_w^n).$$

If we use the implicit scheme, the system is nonlinear and depends on the saturation at time step n and $n+1$.
The discretization is implemented with the following residual form for each cell Ω_i,

$$F_i(s,r) = s_i - r_i + \frac{\Delta t}{\phi_i|\Omega_i|}[H_i(s) - max(q_i, 0) - min(q_i, 0)f(S_i)], \qquad (4.36)$$

where s and r are cell-averaged quantities and subscript i refers to the cell the average is evaluated in. The sum of the interface fluxes for cell i

$$H_i(s) = \sum_k \frac{\lambda_w^u(s_i, s_k)}{\lambda_w^u(si, sk) + \lambda_n^u(si, sk)}[v_{ik} + \lambda_n^u(s_i, s_k)(g_{ik} + P_{ik})], \qquad (4.37)$$

is computed using the single-point upstream mobility-weighting, whereas the fractional flow function f in the source term is evaluated from the cell average of S in cell Ω_i . The explicit scheme is given as $S^{n+1} = S^n - F(S^n, S^n)$ and the implicit solution is obtained solving $F(S^{n+1}, S^n) = 0$. The residual equations (4.36) for all cells in vector form

$$\mathbf{F}(\mathbf{s}) = \mathbf{s} - \mathbf{S} + \frac{\Delta t}{\phi_i|\Omega|}[\mathbf{H}(\mathbf{s}) - \mathbf{Q}^+ - \mathbf{Q}^- \mathbf{f}(\mathbf{S})] = \mathbf{0}. \qquad (4.38)$$

Here, s is the unknown state at time tf and S is the known state at the start of the time step. The nonlinear system can be solved with NR method, where, for the $(k+1)$-th iteration we have:

$$\mathbf{J}(\mathbf{S}^k)\delta\mathbf{S}^{k+1} = -\mathbf{F}(\mathbf{S}^k; \mathbf{S}^n), \qquad \mathbf{S}^{k+1} = \mathbf{S}^k + \delta\mathbf{S}^{k+1},$$

where $\mathbf{J}(\mathbf{S}^k) = \frac{\partial \mathbf{F}(\mathbf{S}^k;\mathbf{S}^n)}{\partial \mathbf{S}^k}$ is the Jacobian matrix, and $\delta\mathbf{S}^{k+1}$ is the NR update at iteration step $k+1$.
Therefore, the linear system to solve is:

$$\mathbf{J}(\mathbf{pS}^k)\delta\mathbf{S}^{k+1} = \mathbf{b}(\mathbf{S}^k). \qquad (4.39)$$

with $\mathbf{b}(\mathbf{S}^k)$ being the function evaluated at iteration step k, $\mathbf{b}(\mathbf{S}^k) = -\mathbf{F}(\mathbf{S}^k; \mathbf{S}^n)$.

Chapter 5

Proper Orthogonal Decomposition (POD)

Model Order Reduction.

The Model Order Reduction (MOR) methods are based in a projection - framework. Where, the original high-order system is transformed into a much lower order one, preserving the most important part of the system dynamics [**?** **?**]. The dynamical system can be written as [**?**]:

$$\mathbf{E}\sigma\mathbf{x}(t) = \mathbf{f}(\mathbf{x}, \mathbf{u}), \tag{5.1}$$

where σ can be the continuous derivative operator $\sigma\mathbf{x}(t) = d\mathbf{x}(t)/dt$, $t \in \mathcal{R}$, or the shift $\sigma\mathbf{x}(t) = x(t+1)$, $t \in \mathbb{Z}$ if the system is discretized in time. The vector $\mathbf{x}$ is an n-dimensional state of the system, for reservoir simulation is the grid-block pressures and saturations vector. The vector $\mathbf{u}$ is a p-dimensional vector-valued input signal, and $\mathbf{E}$ is the descriptor matrix, that contains the physical parameters. For reservoir simulation, E is the accumulation matrix.
Equation (5.1) can be written as:

$$\sigma\mathbf{x}(t) = \mathbf{E}^{-1}\mathbf{f}(\mathbf{x}, \mathbf{u}) = \tilde{\mathbf{f}}(\mathbf{x}, \mathbf{u}). \tag{5.2}$$

The projection-based model reduction models try to find l dimensional ($l << n$) subspaces S_1 and S_2 of the state space that will yield a reduced system as result of projection of the state onto S_1, and the residual onto S_2. For matrices V and W of size $n \times l$ spanning S_1 and S_2, respectively, the reduced order models is:

$$\mathbf{W}^T\mathbf{E}\sigma[\mathbf{V}\mathbf{z}(t)] = \mathbf{W}^T\mathbf{f}(\mathbf{V}\mathbf{z}(t), \mathbf{u}), \tag{5.3}$$

where W and V are chosen to minimize the error in the state-solution approximation $\mathbf{x} \cong \mathbf{V}\mathbf{z}$ or the residual of the system equation:

$$\epsilon(t) := \mathbf{E}\sigma[\mathbf{V}\mathbf{z}(\mathbf{t})] - f(\mathbf{V}\mathbf{z}(t), \mathbf{u}). \tag{5.4}$$

When the subspaces S_1 and S_2 are equal, the projection is called orthogonal, otherwise it is oblique.
In the MOR, the high order model is projected onto the space spanned by a

set of orthonormal basis functions. The high dimensional variable $\mathbf{x} \in \mathbb{R}^n$ is approximated by a linear combination for $l << n$ orthonormal basis functions [**?**]:

$$\mathbf{x} \approx \sum_{i=1}^{l} z_i \phi_i, \tag{5.5}$$

where $\phi_i \in \mathbf{R}^n$ are the basis functions and z_i are its corresponding coefficients. In matrix notation equation (5.5) is:

$$\mathbf{x} \approx \Phi \mathbf{z},$$

where $\Phi = [\phi_1 \ \phi_2 \ .. \ \phi_l]$, $\Phi \in \mathbf{R}^{n \times l}$ are the basis matrix of the basis functions, and $\mathbf{z} \in \mathbf{R}^l$ is the vector of basis functions coefficients. ROM can be applied to speed-up the solution of a linear system,

$$A\mathbf{x} = \mathbf{b}. \tag{5.6}$$

This linear system is projected onto the subspace spanned by the basis Φ:

$$\Phi^T A \Phi \mathbf{z} = \Phi^T \mathbf{b},$$

and the system is rewritten as:

$$A_r \mathbf{z} = b_r.$$

The original sparse matrix $A \in \mathbf{R}^{n \times n}$ is transformed into a much smaller matrix, $A_r \in \mathbf{R}^{l \times l}$.
Although the reduced-order matrix is dense, it is sufficiently small to solve it efficiently using direct inversion methods. The solution z of the reduced order model, is used to compute an approximated solution of equation (5.6) ($\mathbf{x} \approx \Phi \mathbf{z}$).

Proper Orthogonal Decomposition (POD).

The Proper Orthogonal Decomposition method is also known with several names, some of them are: 'Principal components analysis', 'Karhunen-Loeve expansion', and the 'Method of empirical functions'.
This method has been used in several applications, one of these is the modeling or the analysis of flow through porous media [**? ? ?**].
The Proper Orthogonal Decomposition (POD) method is a ROM where the basis functions are based on 'snapshots' which are expected to be 'sufficiently accurate' for the success of the method. This 'snapshots' $\{\mathbf{x_i}\}_{i \in \mathbb{N}}$ are obtained by simulation or experiment and a matrix $n \times m$ is created with this vectors [**?**].

$$X := [\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_m]. \tag{5.7}$$

It is possible to determine, a set of l, $l \leq m << n$, orthogonal vectors, denoted by $\{\phi_j\}_{j=1}^{l}$, with POD.
These orthonormal eigenvectors are the largest eigenvalues $\{\lambda_j\}_{j=1}^{l}$ of the data snapshot correlation matrix,

$$\mathbf{R} := \frac{1}{m}\mathbf{X}\mathbf{X}^T \equiv \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}_i \mathbf{x}_i^T. \tag{5.8}$$

Sometimes, instead of $\mathbf{R}$, the covariance matrix used is:

$$\overline{\mathbf{R}} := \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x}_i - \overline{\mathbf{x}})^T, \tag{5.9}$$

where $\overline{\mathbf{x}} = \sum_{i=1}^{m}\mathbf{x}_i$, is the mean of the snapshots.
This vectors satisfy that the distance between the snapshots and its projections on the subspace defined by $\Phi = [\phi_1, \phi_2, ...\phi_l] \in \mathcal{R}^{n\times l}$, is minimized for any l,

$$Q := \frac{1}{m}\sum_{i=1}^{m}||\mathbf{x}_i - \Phi\Phi^T\mathbf{x}_i||^2. \tag{5.10}$$

The number of dominant eigenvectors l is chosen to be the largest number satisfying:

$$\frac{\sum_{j=1}^{l}\lambda_j}{\sum_{j=1}^{m}\lambda_j} \leq \alpha, \qquad 0 < \alpha \leq 1. \tag{5.11}$$

The standard POD approach involves approximating the solution $\mathbf{x}$ by the linear combination,

$$\mathbf{x} \simeq \Phi\mathbf{z}. \tag{5.12}$$

The resulting l-dimensional reduced order model of (5.3) becomes:

$$\mathbf{\Phi}^T\mathbf{E}\Phi\frac{d}{dt}\mathbf{z}(t) = \mathbf{\Phi}^T\mathbf{f}(\mathbf{\Phi}\mathbf{z}(t), \mathbf{u}(t)). \tag{5.13}$$

The vectors ϕ_j and the matrix Φ are in general dense, therefore $\mathbf{\Phi}^T\mathbf{E}\Phi$ and $\mathbf{\Phi}^T\mathbf{f}$ are dense as well, i.e. the possible sparse pattern is lost. Then a gain in the simulation speed is possible only if the reduced order is much smaller than the original one.

Applications of POD for reservoir simulation.

Improving initial guess.

In [**?**], reduced order models are used to accelerate the solution of systems of equations using iterative solvers in time stepping schemes for large scale numerical solutions. The solution is accelerated by improving the initial guess for the iterative process based on solutions obtained in the previous time steps.
The algorithm consist of two projection steps:
1) Projecting the governing equations onto a subspace spanned by a low number of global empirical basis functions extracted from the previous time step solutions.
2) Solving the governing equations in the reduced space and projecting the solution back to the original one. This algorithm is used for simulation of two-phase flow through heterogeneous porous media. It is modeled with the IMPES implicit pressure explicit saturation scheme and the iterative solution of the pressure equation is investigated. The largest problem had 93500 variables and the maximum reduction in computed time was of 67%. The iterative solvers used where preconditioned conjugate gradient (PCG), minimal residual (MinRes) and successive overrelaxation (SOR) methods.

Preconditioning.

Astrid [**?**] used the Proper Orthogonal Decomposition to speed up the pressure solution during a two stage preconditioning procedure in a fully implicit integration scheme. The method is also applicable to IMPES, IMPEC (implicit pressure explicit accumulation), and AIM (adaptive implicit methods). The reservoir model is a three dimensional, two-phase model, developed by van Essen [**?**]. The spatial domain consists of 25200 grid blocks, 18553 grid cells active, 8 injection wells and 4 production wells, viscosities and fluid compresibilities are equal for both phases. Rock compressibility is negligible, permeabilities are isotropic and no capillary forces were taken into account. Snapshots are collected by simulating the reservoir while turning on and off the wells one by one. Well control settings may change during the full simulation, so the POD basis functions should at least contain the solutions obtained by flowing each of the wells separately or in at least as many linear combinations of well control settings.

In a MsC thesis [**?**], it is developed and investigated a linear-solver preconditioner for the pressure systems in reservoir simulation based on Proper Orthogonal Decomposition. The goal of this work is to investigate the POD method as preconditioner for the linear system of the pressure equation. The POD based approximation used in this work is:

$$\tilde{x} = M_{POD}^{-1} b,$$

where

$$M_{POD}^{-1} = \Phi(\Phi^T A \Phi)^{-1} \Phi^T \qquad \text{Galerkin as default} \tag{5.14}$$

$$M_{POD,LS}^{-1} = \Phi(\Phi^T A^T A \Phi)^{-1} \Phi^T A^T \qquad \text{LSP} \tag{5.15}$$

For the POD method, $(\Phi^T A \Phi)^{-1}$ and $(\Phi^T A^T A \Phi)^{-1}$ are of much smaller dimension than A. The rank of the preconditioning operator M^{-1} is less than or equal to l.
A 10x10 homogeneous oil/water reservoir model using the IMPES scheme was used to study the performance of the POD pressure preconditioner. The POD preconditioner was used with ILU and it was found that it was faster for larger values of l, compared with ILU-only preconditioner. In rhis work they also observed a possible dependence of the RHS using this preconditioner.

Appendix A

MRST [?].

Grid construction.

The MRST (MATLAB Reservoir Simulation Toolbox) allows the use of various grid types, which are stored using a general unstructured format, in which cells, faces, vertices's and connections between cells and faces are explicitly represented.
There is a wide range of structured and unstructured grids that can be constructed in MRST. For the structured grids, a pattern is chosen and repeated. The most typical structured grids are based on quadrilaterals in 2D and hexahedral 3D.

Cartesian grids.

The simplest structured grid is based in a square in 2D and a cube in 3D. to construct these grids in the MRST we need to specify the domain $[0\ Lx,\ 0\ Ly]$ represented as $[Lx\ Ly]$, and number of cells nx, ny, for cartesian grids the construction is in the next form:

$$\begin{aligned} G &= cartGrid([nx, ny], [Lx, Ly]) \qquad 2D, \\ G &= cartGrid([nx, ny], [Lx, Ly, Lz]) \qquad 3D. \end{aligned}$$

If we want to build a 3D cartesian grid with 10 cells in each direction in a cube with $[0\,10] \times [0\,10] \times [0\,1]$, we use the next function:

```
G = cartGrid([10  10  10],[1  1  10]);
plotGrid(G);        to plot the grid
```

Rectilinear grids.

These are also called tensor grids, are rectilinear shapes (rectangles or parallelepipeds) not necessarily congruent to each other.
An example is a rectilinear grid in the domain $[-1, 1] \times [0, 1]$, with an increase of size.

```
dx=1-0.5*cos((-1:0.1:1)*pi);
x=-1.15+0.1*cumsum(dx);
y=0:0.05:1;
G=tensorGrid(x,sqrt(y));
plotGrid(G);
axis([-1.05 1.05 -0.05 1.05])
```

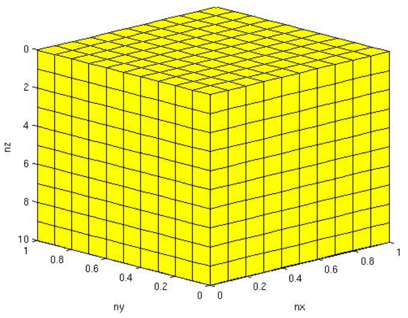

Figure A.1: Cartesian grid in 3D.

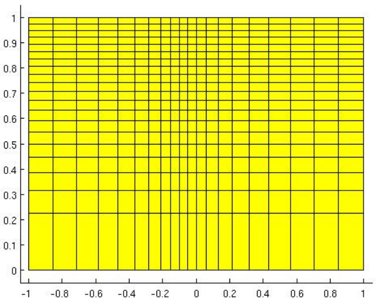

Figure A.2: Rectangular grid in 2D.

All grid types are stored using a general unstructured grid format that represents cells, faces, vertices, and connections between cells and faces. The main grid structure G contains three fields: cells, faces, and nodes, that specify individual properties for each individual cell/face/vertex in the grid.

A grid is a tessellation of a planar or volumetric object by a set of contiguous simple shapes referred to as cells. Grids can be described and distinguished by their geometry, reflected by the shape of the cells that form the grid, and their topology that tells how the individual cells are connected. In 2D, a cell is in general a closed polygon for which the geometry is defined by a set of vertices and a set of edges that connect pairs of vertices and define the interface between two neighboring cells. In 3D, a cell is a closed polyhedron for which the geometry is defined by a set of vertices, a set of edges that connect pairs of vertices, and a set of faces (surfaces delimited by a subset of the edges) that define the interface between two different cells, see . All cells in a grid are non-overlapping, so that each point in the planar/volumetric object represented by the grid is either inside a single cell, lies on an interface or edge, or is a vertex. Two cells that share a common face are said to be connected. Likewise, one can also define connections based on edges and vertices. The topology of a grid is defined by the total set of connections, which is sometimes also called the connectivity of the grid.

The cell structure, G.cells, consists of the following mandatory fields:

- num: the number n_c of cells in the global grid.
- facePos: an indirection map of size [num+1,1] into the faces array. Specifically, the face information of cell i is found in the submatrix faces(facePos(i) : facePos(i+1)−1, :) The number of faces of each cell may be computed using the state- ment diff(facePos) and the total number of faces is given as nf = facePos(end)−1.

- faces: an $n_f \times 3$ array that gives the global faces connected to a given cell. If faces(i,1)==j, the face with global number faces(i,2) is connected to cell number j. The last component, faces(i,3), is optional and can for certain types of grids contain a tag used to distinguish face directions: East, West, South, North, Bottom, Top.
- indexMap: an optional $n_c \times 1$ array that maps internal cell indices to external cell indices. For models with no inactive cells, indexMap equals 1 : nc . For cases with inactive cells, indexMap contains the indices of the active cells sorted in ascending order.

The face structure, G.faces, consists of the following mandatory fields:

- num: the number n_f of global faces in the grid.
- nodePos: an indirection map of size [num+1,1] into the nodes array. The node information of face i is found in the submatrix nodes(nodePos(i) : nodePos(i+1)−1, :) The number of nodes of each face may be computed using the state- ment diff(nodePos). Likewise, the total number of nodes is given as $nn = nodePos(end) - 1$.
- nodes: an $N_n \times 2$ array of vertices in the grid. If nodes(i,1)==j, the local vertex i is part of global face number j and corresponds to global vertex nodes(i,2). For each face the nodes are assumed to be oriented such that a right-hand rule determines the direction of the face normal. As for cells.faces, the first column of nodes is redundant and can be easily reconstructed.
- neighbors: an $n_f \times 2$ array of neighboring information. Global face i is shared by global cells neighbors(i,1) and neighbors(i,2). One of the entries in neighbors(i,:), but not both, can be zero, to indicate that face i is an external face that belongs to only one cell (the nonzero entry).

The vertex structure, G.nodes, consists of two fields: – num: number N_n of global nodes (vertices) in the grid, – coords: an $N_n \times d$ array of physical nodal coordinates in $\mathbb{R}^d$. Global node i is at physical coordinate coords(i,:).

Rock modeling in MRST.

The rock parameters are represented as fields in a structure. All the solvers in MRST work with this assumption. The name of the structure is **rock**, and the fields for porosity and permeability are **poro** (**rock.poro**) and **perm**, (**rock.perm**). In the MRST software, the permeability can be a vector for isotropic permeability or a symmetric tensor.
If we want to model an homogeneous 10x10 grid with uniform porosity of 0.2 and isotropic permeability of 200 mD, we just fill the corresponding fields with these values.
MRST works in SI units, so we need to convert from 'Darcy' to '$meters^2$', this can be done by multiplication of the functions **milli** and **darcy**, that return the corresponding conversion factors. There is another conversion function $convertFrom(200, milli * darcy)$.

```
G=cartGrid([10 10]);
rock.poro=repmat(0.2,[G.cells.num,1]);
rock.perm=repmat(200*milli*darcy,[G.cells.num,1]);
```

For an homogeneous, anisotropic permeability, the field is constructed in a similar way.

```
rock.perm=repmat([100 100 10]*milli*darcy,[G.cells.num,1]);
```

It is not easy to measure the rock properties, so it is common to use geostatistical methods for the porosity and permeability fields. MRST contains two methods for generating geostatistical realizations. The first one has porosity ϕ as a Gaussian field (Figure A.3), and the second one is with layered realizations (Figure A.4).
In the software, there are also some models, with pre-defined structure, permeability and porosity, that can be used. Some of these models are:

1. 10th SPE.
2. The Johansen formation.
3. SAIGUP model.
4. Etc.

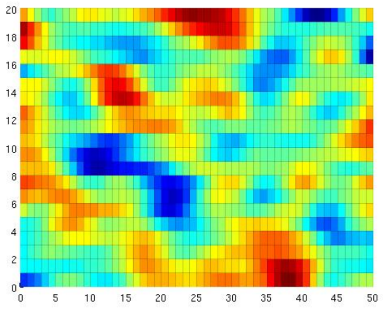

Figure A.3: Gaussian field in 2D.

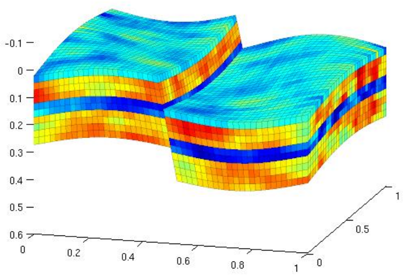

Figure A.4: Layered permeability in 3D.

Fluid properties.

For the basic single-phase flow equation, the fluid properties needed are the viscosity and the fluid density for incompressible models, and fluid compressibility for compressible models. MRST uses fluid objects that contain basic fluid properties and some function handles used to evaluate rock-fluid properties for multiphase flow. If we want to determine the viscosity and density of a fluid, we need to write the values with units in the fluid object.

```
fluid=initSingleFluid('mu',1*centi*poise,'rho',1014*kilogram/meter^3).
```

After initialization, the fluid object will contain pointers to functions that can be used to evaluate petrophysical properties of the fluid:

```
fluid =
properties:@(varargin)properties(opt,varargin:)
saturation:@(x,varargin)x.s
relperm:@(s,varargin)relperm(s,opt,varargin:)
```

The first function returns the viscosity when called with a single output argument and the viscosity and the density when called with two output arguments. For single-phase flow this is the only relevant function.
For the saturation function, the argument is the reservoir state and returns the corresponding saturation. The relperm function accepts a fluid saturation as argument and returns the relative permeability, i.e., the reduction in permeability due to the presence of other fluid phases. For single-phase flow is identical to one.

Reservoir States.

MRST uses a structure for the dynamic state of the reservoir, it is called state object. This structure contains 3 elements: a pressure and a saturation vector, with the pressure and saturation of each cell in the model. And a vector flux, with one flux per grid face in the model. The state object is initialized by a call to the function:

$\boxed{\mathit{state} = \mathit{initResSol}(G, p_0, s_0),}$

where p_0 is the initial pressure, and s_0 is the initial saturation, which is 1 for single-phase models. If the reservoir has wells, the state object should contain an additional field wellSol, that is a vector with the size of the number of wells. Each element of the previous vector is a structure with two fields $wellSol.pressure$ and $wellSol.flux$.

$\boxed{\mathit{state} = \mathit{initResSol}(G, W, p_0, s_0).}$

Wells.

The wells describe injection or extraction of fluids from the reservoir. They provide volumetric flux rate and a model that couples the flux rate to the difference between the average in the grid cell and the pressure inside the wellbore. The structure used to represent wells is W, and consist of the following fields:

1. cells: an array index to cells perforated by this well.
2. type: string describing which variable is controlled, either 'bhp' (default, the well is controlled by bottom-hole pressure), or 'rate' (the well is rate controlled).
3. val: the target value of the well control (Pascal for 'bhp' or [m^3/sec]'rate'). Default value is 0.
4. radius: the wellbore radius (default 0.1 m).

5. dir: a char describing the direction of the perforation, one of the cardinal directions 'x', 'y' or 'z'.
6. WI: the well index: either the productivity index or the well injectivity index depending on whether the well is producing or injecting.
7. name: string giving the name of the well.
8. compi: fluid composition, only used for injectors.
9. refDepth: reference depth of control mode.
10. sign: define if the well is intended to be producer or injector.

The well structures are created with the function:

$\boxed{W = addWell(W, G, rock, CellInx,' pn', pv, ...}$

Here, *cellInx* is a vector of the indices's to the cells perforated by the well and 'pn' indicate more parameters that can influence the well. There is also the function:

$\boxed{W = verticalWell(W, G, rock, I, J, K),}$

that can be used to specify vertical wells in models described by Cartesian grids. I,J gives the horizontal location of the well.

Sources.

The flow into or out the interior points of the reservoir can be also added using the following function:

$$\boxed{\begin{array}{l} src = addSource(scr, cells, rates); \\ src = addSource(scr, cells, rates,' sat', sat). \end{array}}$$

Src is a structure that contains the following fields:

1. *cell* : cells for which explicit sources are provided,
2. *rate* : rates for these explicit sources,
3. *value* : pressure or flux value for the given condition,
4. *sat* : specifies the composition of injected fluids in cell.

Boundary conditions.

For specifying Dirichlet or Neumann conditions, the functions used are the following:

```
src=addBC(bc,faces,type,values);
src=addBC(bc,faces,type,values'sat',sat),
```

The input values are: *bc* : structure that contains the following fields:

1. *face* : external faces for which explicit conditions are set,

2. *type* : cell array of strings denoting type of condition,
3. *value* : pressure or flux value for the given condition,
4. *sat* : fluid composition.

Incompressible two-point pressure solver.

Using a simplified single-phase flow equation

$$\nabla \cdot \vec{v} = q, \qquad \vec{v} = -\frac{K}{\nabla} p, \tag{A.1}$$

Using a finite-volume discretization for Equation (A.1), the equation is rewritten in integral form using a single cell Ω_i in the discrete grid as control volume.

$$\int_{\partial\Omega_i} \vec{v} \cdot \vec{n} ds = \int_{\Omega_i} q dx \tag{A.2}$$

Using Darcy's law to compute the flux across each face of the cell,

$$v_{i,k} = \int_{\partial\Gamma_{ik}} \vec{v} \cdot \vec{n} ds, \qquad \Gamma_{i,k} = \partial\Omega_i \cap \partial\Omega_i. \tag{A.3}$$

$\Gamma_{i,k}$ are the half faces, they are associated with a particular grid cell Ω_i and a certain normal vector $\vec{n}_{i,k}$. Each interior half face will have a twin half-face $\Gamma_{k,i}$ that has identical area $A_{i,k} = A_{k,i}$ but opposite normal vector $\vec{n}_{i,k} = -\vec{n}_{k,i}$. Approximating the integral by the midpoint rule (see figure **??**), the flux can be written as:

$$v_{i,k} \approx A_{i,k} \vec{v}(\vec{x}_{i,k}) \cdot \vec{n}_{i,k} = -A_{i,k}(K\nabla p)(\vec{x}_{i,k}) \cdot \vec{n}_{i,k}, \tag{A.4}$$

where $\vec{x}_{i,k}$ denotes the centroid on $\Gamma_{k,i}$.

Assuming a linear or constant pressure inside the cell i, the gradient can be approximated with finite differences scheme taking the difference between the pressure at the face centroid $\pi_{i,k}$ and the average pressure inside the cell p_i, Equation (A.4) becomes :

$$v_{i,k} \approx A_{i,k} K_i \frac{(p_i - \pi_{i,k}) \vec{c_{i,k}}}{|\vec{c}^2_{i,k}|} \cdot \vec{n}_{i,k} = T_{i,k}(p_i - \pi_{i,k}), \tag{A.5}$$

where $T_{i,k}$ is the transmissibility associated with one single cell and gives a two point relation between the flux across a cell face and the difference between the pressure at the cell and face centroids, they are called half-transmissibilities. Imposing continuity of flux across the faces $v_{i,k} = -v_{k,i} = v_{ik}$ and continuity of face pressures $\pi_{i,k} = \pi_{k,i} = \pi_{ik}$ we obtain two equations

$$T^{-1}_{i,k} v_{ik} = (p_i - \pi_{ik}), \qquad -T^{-1}_{k,i} v_{ik} = (p_k - \pi_{ik}). \tag{A.6}$$

Eliminating the interface pressure π_{ik} we obtain the two-point flux approximation scheme.

$$v_{ik} = [T^{-1}_{i,k} + T^{-1}_{k,i}]^{-1}(p_i - p_k). \tag{A.7}$$

Adding up all the contributions we have:

$$\sum_k T_{ik}(p_i - \pi_k), \qquad \forall \Omega_i \subset \Omega. \tag{A.8}$$

The pressure is specified for a point $p_1 = 0$.

The two-point flux-approximation scheme is used in MRST, the half-face transmissibility of eq. (**??**) is computed with:

$\boxed{ht = computeTrans(G, rock).}$

And the solution is obtained with:

$\boxed{state = imcompTPFA(state, G, ht, fluid,' mech1, obj1, ...),}$

that takes the complete model, with mech the drive mechanism ('src','bc', and or 'wells'). The inflow and outflow must sum up to zero for preventing the violation of the assumption of incompressibility.

Two-phase flow

When going from a single-phase to a multiphase flow model, the most prominent changes take place in the fluid model. It is this model that generally will tell your solver how many phases are present and how these phases affect each other when flowing together in the same porous medium. To describe an incompressible flow model, we need to know the viscosity and the constant density of each fluid phase, as well as the relative permeabilities of the fluid phases. If the fluid model includes capillary forces, we also need one or more functions that specify the capillary pressure as function of saturation. The most basic multiphase fluid model in MRST is the following,

```
fluid    = initSimpleFluid( 'mu', [1, 10] * centi * poise, ...
           ' rho ', [1014, 859]*kilogram /meter ^3, ...
           ' n ' , [2, 2]);
```

which implements a simplified version of the Corey model (see Equation 4.24) in which the residual saturations S_{wr} and S_{nr} are assumed to be zero and the end-points and scaled to unity, so that $k_{rw} = (S_w)^{n_w}$ and $S_{rn} = (1 - S_w)^{n_n}$

```
mu = fluid.properties();% gives mu_w and mu_n
[mu, rho] = fluid.properties(); % .... plus rho_w and rho_n
```

New to multiphase flow is the relperm function, which takes a single fluid saturation or an array of fluid saturations as input and outputs the corresponding values of the relative permeabilities. Plotting the relative permeability curves of the fluid object can be achieved by the following code

The relperm function can also return the first and second derivatives of the relative permeability curves when called with two or three output arguments. The basic fluid model does not have any capillary pressure. To also include this effect, one can use another fluid model, which adds a capillary function that

```
s=linspace(0,1,20)';
kr = fluid.relperm(s);
plot(s,kr (:,1), ' −s' ,s,kr (:,2), ' −o');
```

```
fluid = initSimpleFluidPc('pc scale', 2*barsa);
```

assumes a linear relationship $P_c(S) = C(1 - S)$.
The capillary pressure function pc is evaluated using a state object and not a saturation. This simple function can be extended to include Leverett J- function scaling by using the fluid object generated by $initSimpleFluidJfunc$. The incomp module also implements the general Corey model with end- point scaling k^0_α and nonzero residual saturations S_{wr} and S_{nr}.

Explicit solver
The incomp module offers the following explicit transport solver

$\boxed{state = explicitTransport(state, G, tf, rock, fluid,' mech1', obj1, ...);}$

which evolves the saturation given in the state object a step tf forward in time. The function requires a complete and compatible model description consisting of a grid structure G, petrophysical properties rock, and a fluid model fluid. The state object must contain the correct number of saturations per cell and an incompressible flux field that is consistent with the global drive mechanisms given by the mech argument ('src', 'bc', and/or 'wells') using correctly specified objects obj. The input value of state must be the output value of a previous call to an incompressible solver like incompTPFA, incompMPFA, or incompMimetic. In addition, the function takes a number of optional parameters that determine whether the time steps are prescribed by the user or to be automatically computed by the solver. The solver can also ignore the Darcy flux and work as a pure gravity segregation solver if the optional parameter onlygrav is set to true. Finally, the solver will issue a warning if the updated saturation value is more than satwarn outside the interval [0, 1] of physically meaningful values (default value: sqrt(eps)).
The implicit and the explicit solvers involve many of the same operations and formulas used for the spatial discretizations. To avoid duplication of code we have therefore introduced a private help function that implements the residual

$\boxed{[F, Jac] = twophaseJacobian(G, state, rock, fluid,' pn1', pv1, ...);}$

form (10.3) and its Jacobian matrix J = dF and returns these as two function handles F and Jac. Using this function, the key lines of the explicit saturation solver read,

The function $correct_saturations$ ensures that the computed saturations stay inside the interval of physically valid states.
Implicit solver
The implicit solver has the same user interface and parameter requirement as the explicit solver In addition, there are optional parameters that control the Newton–Raphson method used to solve for S^{n+1}.

```
$F = twophaseJacobian(G, state, rock, fluid,' wells', opt.wells, ...);$
$s = state.s(:, 1);$
$t = 0;$
while t ¡ tf,
    $dt = min(tf - t, getdt(state));$
    $s(:) = s - F(state, state, dt);$
    $t = t + dt;$
    $s = correct_saturations(s, opt.satwarn);$
    $state.s = [s, 1 - s];$
end
```

```
$state = explicitTransport(state, G, tf, rock, fluid,' mech1', obj1, ...);$
```