Linear Algebra : Matrix inversion

In linear algebra we commonly encounter situations where we need to solve matrix equation of the form

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{1}$$

where $\mathbf{A}$ is an $n \times n$ matrix, $\mathbf{x}$ and $\mathbf{b}$ can be either $n \times 1$ vectors or $n \times n$ matrices depending on the problem. Let us just say for now that they are vectors, then explicitly,

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{2}$$

Solving equation (1) *i.e.* (2) implies determining the vector $\mathbf{x}$,

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \tag{3}$$

Therefore, the problem here to address is how to obtain the inverse of $\mathbf{A}$ along with determining its determinant, which is often required to check the existence of inverse. From the computational standpoint, we do not actually attempt to find the inverse in the traditional sense involving the cofactors $\mathbf{A}_c$,

$$\mathbf{A}^{-1} = \frac{\mathbf{A}_c^T}{|\mathbf{A}|} \tag{4}$$

Although numerically such inverse is doable, far more efficient approaches exist which can be put in two categories –

- Direct elimination : Gauss-Jordan, LU decomposition, Cholesky decomposition. These are mostly used for smaller matrices and are memory intensive, possibly slower and cannot be parallelized.
- Iterative methods : Gauss-Jacobi, Gauss-Seidel, Conjugate gradient, GMRES. These are rather useful for larger matrices, particularly *sparse matrices* and lend themselves to parallelization. (Parallelizing Gauss-Seidel is little non-trivial though.)

Here we look at these methods in turn.

Gauss-Jordon method : The method involves rewriting the matrix equation (2) in *augmented matrix form* and then converting into *reduced row echelon form* (RREF),

$$[\mathbf{A}|\mathbf{b}] \;\Rightarrow\; \left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right] \Rightarrow \left[\begin{array}{cccc|c} 1 & 0 & \cdots & 0 & \tilde{b}_1 \\ 0 & 1 & \cdots & 0 & \tilde{b}_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & \tilde{b}_3 \end{array}\right] \tag{5}$$

The RREF is achieved by Gauss-Jordon elimination with *partial* (*full*) *pivoting* through any one or a combination of three elementary row (or row-column) operations

- Swap two rows. In partial pivoting, the algorithm selects the entry with largest absolute value from the pivoting column. This implies swap the rows such that the row with largest leftmost entry is at the top.

- Multiply a row by a nonzero number.
- Add or subtract a multiple of one row to another row

In the RREF,

1. all rows with only zero entries are at the bottom of the matrix
2. the first nonzero entry in a row (called *pivot*) of each nonzero row is to the right of the leading entry of the row above it
3. leading entry *i.e.* pivot in any nonzero row is 1
4. all other entries in the row or column containing a leading 1 are zeros.

It is important to note that no matter what steps and in which order they are applied, the final augmented matrix in (5) is unique. For calculation of determinants, one just needs a *row echelon matrix*, instead of fully RREF. The steps involve are

$$\det \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = \det \begin{pmatrix} a'_{11} & a'_{12} & \cdots & a'_{1n} \\ 0 & a'_{22} & \cdots & a'_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a'_{nn} \end{pmatrix} = (-1)^d \prod_{i=1}^{n} a'_{ii} \tag{6}$$

where d is the number of swaps *i.e.* number of times the rows are interchanged, $a'_{ii} \neq 0$ and none of a'_{ii} has to be 1. For $\mathbf{A}^{-1}$, instead of vector $\mathbf{b}$, we replace it with $\mathbf{b} = \mathbb{1}$ unit matrix and run through the above steps towards RREF. At the end, the matrix $\mathbf{b}$ will contain the inverse.

Computational cost *i.e.* number of arithmetic operations needed is $\mathcal{O}(n^3)$ for an $n \times n$ matrix. Therefore, the cost grows quickly as n increases.

LU Decomposition : The LU decomposition involves factorization of a square matrix A, with proper row and/or column orderings (partially pivoted), into two triangular matrices lower $\mathbf{L}$ and upper $\mathbf{U}$, as shown below for 4×4 $\mathbf{A}$ matrix,

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{U} \text{ where, } \mathbf{L} = \begin{pmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix} \text{ and } \mathbf{U} = \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{pmatrix} \tag{7}$$

Without proper ordering or (at least) partial pivoting, the factorization may fail. The matrix multiplication $\mathbf{L} \cdot \mathbf{U}$ yields

$$\begin{pmatrix} l_{11}u_{11} & l_{11}u_{12} & l_{11}u_{13} & l_{11}u_{14} \\ l_{21}u_{11} & l_{21}u_{12} + l_{22}u_{22} & l_{21}u_{13} + l_{22}u_{23} & l_{21}u_{14} + l_{22}u_{24} \\ l_{31}u_{11} & l_{31}u_{12} + l_{32}u_{22} & l_{31}u_{13} + l_{32}u_{23} + l_{33}u_{33} & l_{31}u_{14} + l_{32}u_{24} + l_{33}u_{34} \\ l_{41}u_{11} & l_{41}u_{12} + l_{42}u_{22} & l_{41}u_{13} + l_{42}u_{23} + l_{43}u_{33} & l_{41}u_{14} + l_{42}u_{24} + l_{43}u_{34} + l_{44}u_{44} \end{pmatrix} \tag{8}$$

To determine l_{ij} and u_{ij} we can equate each elements of $\mathbf{L} \cdot \mathbf{U}$ matrix with the corresponding elements of $\mathbf{A}$ and solve the equations, a few of these are shown below in the equations (9 – 12).

$$l_{11}u_{11} = a_{11} \qquad l_{11}u_{12} = a_{12} \qquad \ldots \tag{9}$$

$$l_{21}u_{11} = a_{21} \qquad l_{21}u_{12} + l_{22}u_{22} = a_{22} \qquad \ldots \tag{10}$$

$$l_{31}u_{12} + l_{32}u_{22} = a_{32} \qquad l_{31}u_{13} + l_{32}u_{23} + l_{33}u_{33} = a_{33} \qquad \ldots \tag{11}$$

$$l_{41}u_{13} + l_{42}u_{23} + l_{43}u_{33} = a_{43} \qquad l_{41}u_{14} + l_{42}u_{24} + l_{43}u_{34} + l_{44}u_{44} = a_{44} \qquad \ldots \tag{12}$$

Now, here is a problem! We have $4 \times 4 = 16$ equations but $4 \times (4+1) = 20$ variables, hence we cannot have a unique solutions for l_{ij} and u_{ij}. The trick here is to put either all four $l_{ii} = 1$, known as *Doolittle decomposition*, or all four $u_{ij} = 1$, known as *Crout decomposition*.

Any of the above LU decompositions *i.e.* determination of the $\mathbf{L}$ and $\mathbf{U}$ can proceed iteratively. For Doolittle decomposition,

- Set $l_{ii} = 1$ for all $i = 1, \ldots, N$ implying $u_{1j} = a_{1j}$ for $j = 1, \ldots, N$, see the equations in (9).
- For each $j = 1, 2, \ldots N$ do both the calculations below in the order they appear

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \qquad \text{for } \; i = 2, \ldots j \tag{13}$$

$$l_{ij} = \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right) \qquad \text{for } \; i = j+1, j+2, \ldots, N \tag{14}$$

An interesting point to note that every a_{ij} in (13 and 14) is used only once and never again. This means that the u_{ij} and l_{ij} can be stored in the location that corresponding a_{ij} is to occupy. No need to store $l_{ii} = 1$ at all, just modify the looping over indices accordingly. For Doolittle decomposition this means,

$$\mathbf{LU} = \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ l_{21} & u_{22} & u_{23} & u_{24} \\ l_{31} & l_{32} & u_{33} & u_{34} \\ l_{41} & l_{42} & l_{43} & u_{44} \end{pmatrix} \;\; \rightarrow \;\; \mathbf{A} \tag{15}$$

This is an advantage of LU decomposition over Gauss-Jordan, which requires at least twice the storage for generating RREF, otherwise the numerical cost remains the same *i.e.* $\mathcal{O}(n^3)$.

The calculation of determinant in LU decomposition is trivial since $\det \mathbf{A} = \det \mathbf{LU} = \det \mathbf{L} \times \det \mathbf{U} = (-1)^n \prod_i u_{ii}$. But the inverse follow the similar steps as Guass-Jordon, solving each column of the inverse matrix at a time by solving the a set of linear equations. The procedure can be performed as follows,

1. Consider the equation (7),
$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \;\; \Rightarrow \;\; \mathbf{L} \cdot \mathbf{U} \cdot \mathbf{x} = \mathbf{b} \tag{16}$$
2. Split up the above equation (16) in two equations
$$\mathbf{U} \cdot \mathbf{x} = \mathbf{y} \;\; \Rightarrow \mathbf{L} \cdot \mathbf{y} = \mathbf{b} \tag{17}$$
3. In the first step solve for $\mathbf{y}$ from $\mathbf{L} \cdot \mathbf{y} = \mathbf{b}$ using *forward substitution* and then use it to solve for $\mathbf{x}$.

The *forward* solution proceed exactly the *backward* way except that here we use $\mathbf{L}$. Rewriting the $\mathbf{L}$ and $\mathbf{U}$ matrices again for ready reference,

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 \\ l_{41} & l_{42} & l_{43} & 1 \end{pmatrix} \;\; \text{and} \;\; \mathbf{U} = \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{pmatrix} \tag{18}$$

The solutions for y_i and consequently x_i are thus

$$y_i = b_i - \sum_{j=1}^{i-1} l_{ij} y_j, \qquad \text{where } y_1 = b_1 \text{ and } i = 2, 3, \ldots, N \tag{19}$$

$$x_i = \frac{1}{u_{ii}} \left(y_i - \sum_{j=i+1}^{N} u_{ij} x_j \right), \qquad \text{where } x_N = \frac{y_N}{u_{NN}} \text{ and } i = N-1, N-2, \ldots, 1 \tag{20}$$

For the inverse, use the combination of (19) and (20) to iterate through each column of the identity matrix $\mathbf{b} = \mathbb{1}$.

If the matrix $\mathbf{A}$ has some specific properties and symmetries, then we can exploit that to simplify the above procedure. An useful scheme for factorization of Hermitian, positive definite matrix is *Cholesky decomposition*. Here, the matrix is factorize into a product of lower triangular matrix $\mathbf{L}$ and its conjugate transpose. For obvious reason, Cholesky is about twice as efficient LU decomposition. For a 3×3 system,

$$\mathbf{A} = \mathbf{L}\,\mathbf{L}^{\dagger} \overset{\text{real matrix}}{\longrightarrow} \mathbf{L}\,\mathbf{L}^{T} \Rightarrow \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} l_{11} & l_{12} & l_{13} \\ 0 & l_{22} & l_{23} \\ 0 & 0 & l_{33} \end{pmatrix} \tag{21}$$

When $\mathbf{A}$ is real and positive definite, then

$$l_{ii} = \pm \sqrt{a_{ii} - \sum_{j=1}^{i-1} l_{ij}^2} \tag{22}$$

$$l_{ij} = \frac{1}{l_{ii}} \left(a_{ij} - \sum_{k=1}^{i-1} l_{ik} l_{kj} \right) \quad \text{for } i < j \tag{23}$$

The expression under the square root in (22) is always positive and the sign before the square root is inconsequential. Figure out yourself the corresponding expressions for complex Hermitian matrix. Once the decomposition is performed, the remaining process proceeds by forward and backwards substitution in the same manner as the above Gauss-Jordon / LU decomposition. Apart from being used for numerical solution of linear equations (7), Cholesky decomposition is also used in non-linear optimization for multiple variable, monte carlo simulation for decomposing covariance matrix, inversion of Hermitian matrices etc.

However, Gauss-Jordon and LU decomposition become quickly unmanageable with not only for increasing matrix size but also for increasing number of 0 entries in which case even pivoting itself becomes a costly affair. The strategy for such cases is *iterative* methods and the most common iterative scheme is Gauss-Seidel. But let us first try to understand the steps involved through a simpler but effective method called *Jacobi* method.

Jacobi method : It is an iterative algorithm for solving a matrix equation that has no zeros in its main diagonal. In this method each diagonal element is solved for and an approximate value is put back in. The process goes on until it converges. The basic step consists of decomposing (but not factorizing) $\mathbf{A}$ into a diagonal $\mathbf{D}$, lower triangular L and upper triangular $\mathbf{U}$ matrices

$$\mathbf{A} = \mathbf{D} + (\mathbf{L} + \mathbf{U}) \text{ where } \mathbf{D} = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}, \quad \mathbf{L} + \mathbf{U} = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix} \tag{24}$$

The solution of (1) can be approached by first rewriting the matrix equation

$$\begin{aligned} & \mathbf{A}\cdot\mathbf{x}=\mathbf{b} \\ \Rightarrow \quad & \Big[\mathbf{D}+(\mathbf{L}+\mathbf{U})\Big]\cdot\mathbf{x}=\mathbf{b} \\ \Rightarrow \quad & \mathbf{x}=\mathbf{D}^{-1}\Big[\mathbf{b}-(\mathbf{L}+\mathbf{U})\cdot\mathbf{x}\Big] \end{aligned} \tag{25}$$

The above expression can be evaluated iteratively for $\mathbf{x}$ until our error tolerance is met $||\mathbf{x}^{k+1}-\mathbf{x}^{k}|| < \epsilon$,

$$\mathbf{x}^{(k+1)}=\mathbf{D}^{-1}\left(\mathbf{b}-(\mathbf{L}+\mathbf{U})\mathbf{x}^{(k)}\right) \;\Rightarrow\; x_i^{(k+1)}xi=\frac{1}{a_{ii}}\left(b_i-\sum_{j\neq i}a_{ij}x_j^{(k)}\right) \tag{26}$$

Although this method is fairly fast, the computation of $x_i^{(k+1)}$ element requires all other elements $x_{j\neq i}^{(k)}$ except itself and, therefore, we cannot overwrite $x_i^{(k)}$ with $x_i^{(k+1)}$ which necessitate a minimum storage space for two n-size vectors. A sufficient condition for the method to converge is that the matrix A is *strictly diagonally dominant*,

$$|a_{ii}| > \sum_{j\neq i}|a_{ij}| \tag{27}$$

Although Jacobi method is relatively slower to converge to solutions, its main advantage is that it can be easily parallelized and hence can handle large matrices, which reduces both memory requirement for matrix storage and time taken.

Gauss-Seidel method : Like Jacobi method, it is also an iterative method but it can be applied to any matrices with non-zero diagonal elements. However, its convergence is only guaranteed if the matrix is either *strictly diagonally dominant* or *symmetric and positive definite*. It is defined by the iteration

$$\mathbf{L}_\star\mathbf{x}^{(k+1)}=\mathbf{b}-\mathbf{U}\mathbf{x}^{(k)} \tag{28}$$

where the matrix $\mathbf{A}$ is decomposed as

$$\mathbf{A}=\mathbf{L}_\star+\mathbf{U} \;\text{ where, }\; \mathbf{L}_\star=\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix},\; \mathbf{U}=\begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \tag{29}$$

The iteration equation (28) leads to computation of the elements of $\mathbf{x}^{(k+1)}$ by

$$x_i^{(k+1)}=\frac{1}{a_{ii}}\left(b_i-\sum_{j=1}^{i-1}a_{ij}x_j^{(k+1)}-\sum_{j=i+1}^{n}a_{ij}x_j^{(k)}\right),\;\; i=1,2,\ldots,n \tag{30}$$

As in all iterative procedure, the process continues until the changes $||\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}||$ is below tolerance.

The main advantage Gauss-Seidel has over Gauss-Jacobi is that only one storage vector for $\mathbf{x}^{(k)}$ is required. This because, as seen in equation (30), the computation of $\mathbf{x}^{(k+1)}$ uses elements of $\mathbf{x}^{(k+1)}$ that have already been computed and those $\mathbf{x}^{(k)}$ that are computed in the $k+1$ iteration. However, its major short coming is, compared to Jacobi, that it is much harder to parallelize the algorithm. Overall, Gauss-Seidel is found advantageous for problems with large and sparse matrices.

Sparse matrix : Both the Jacobi and Gauss-Seidel emphasize on non-zero diagonal matrix but do not talk much about off-diagonal elements. Therefore, if we have large sparse matrices, whose most of the off-diagonal elements are zero, then iterative methods are excellent choice. Consider for instance, the simplified Ising model Hamiltonian

$$H = -J \sum_{m,n} \sigma_m \sigma_n - h \sum_m \sigma_m \tag{31}$$

where (m, n) pairs are nearest neighbors, J is the spin-spin interaction strength, h represents the external field and σ_m are the individual spins on each of (discrete) lattice sites. For a 6×6 periodic lattice, the matrix form of the Hamiltonian is

$$H = - \begin{pmatrix} h\sigma_1 & J\sigma_1\sigma_2 & 0 & 0 & 0 & 0 \\ J\sigma_2\sigma_1 & h\sigma_2 & J\sigma_2\sigma_3 & 0 & 0 & 0 \\ 0 & J\sigma_3\sigma_2 & h\sigma_3 & J\sigma_3\sigma_4 & 0 & 0 \\ 0 & 0 & J\sigma_4\sigma_3 & h\sigma_4 & J\sigma_4\sigma_5 & 0 \\ 0 & 0 & 0 & J\sigma_5\sigma_4 & h\sigma_5 & J\sigma_5\sigma_6 \\ J\sigma_6\sigma_1 & 0 & 0 & 0 & J\sigma_6\sigma_5 & h\sigma_6 \end{pmatrix} \tag{32}$$

Therefore, more than 50% of the entries are zero. Almost similar situation arises with fermion in discrete Euclidean space-time lattice. For simplicity consider the free Euclidean Dirac action,

$$\mathcal{S}_{\rm cont} = \overline{\psi}(x) \Big(\gamma_\mu \partial_\mu + m \Big) \psi(x) \;\; \Rightarrow \;\; \mathcal{S}_{\rm disc} = \frac{1}{2} \overline{\psi}_x \, \gamma_\mu \Big(\psi_{x+\mu} - \psi_{x-\mu} \Big) + m \, \overline{\psi}_x \psi_x \tag{33}$$

where replace continuum derivative with finite difference, which in most simplest form

$$\partial_\mu \psi \;\; \rightarrow \;\; \frac{1}{2} \Big(\delta_{x+\mu,y} - \delta_{x-\mu,y} \Big) \tag{34}$$

assuming the spacing between any two neighboring space-time point (lattice spacing) is unity. This expression immediately suggests sparse form of the fermion matrix. The fermion propagator in QFT is the inverse of the fermion matrix. The problem here is not only inversion of a sparse matrix defined in equation (33) but also the size of the matrix. On a modest 8^4 lattice, fermion fields ψ_x have 4 Dirac indices and 3 color indices giving a total of $8^4 \times 4 \times 3 \times 2 = 98,304$ degrees of freedom and, hence, the problem involves inverting a $10^5 \times 10^5$ sparse matrix! Calculation of memory requirement of such a matrix is left as an exercise. Clearly, we need a completely different strategy to solve such matrix equations. The two most widely used methods are Conjugate Gradient and Minimal Residue, and somewhat less popular is Gauss-Seidel. The Conjugate Gradient method discussed below is liberally borrowed from the original CG paper [1] and [2, 3]

Conjugate gradient method : This is a very popular method for solving large systems of linear equations as in (1), when $\mathbf{A}$ is symmetric, positive-definite matrix *i.e.* for every nonzero vector $\mathbf{x}$,

$$\langle \mathbf{x}, \mathbf{A}\mathbf{x} \rangle = \mathbf{x}^T \; \mathbf{A} \, \mathbf{x} > 0 \tag{35}$$

A Hermitian matrix is a positive definite matrix. Let us take for granted the following statement :
`The solution vector` $\mathbf{x}^\star$ `of the equation (1),` $\mathbf{A}\mathbf{x} = \mathbf{b}$ `where` $\mathbf{A}$ `is positive definite and symmetric, is the minimal value of the quadratic form`

$$f(\mathbf{x}) = \frac{1}{2} \langle \mathbf{x}, \mathbf{A}\mathbf{x} \rangle - \langle \mathbf{x}, \mathbf{b} \rangle \tag{36}$$

`Prove the above statement for a` 2×2 `system.`

As an illustration, consider a simple set of linear equations in two variables

$$\left.\begin{array}{rcl} 2x_1 + x_2 &=& 3 \\ x_1 + x_2 &=& 2 \end{array}\right| \Rightarrow \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix} \tag{37}$$

Using equation (36), the explicit expression for the quadratic form $f(x_1, x_2)$ is,

$$f(x_1, x_2) = \frac{1}{2}\Big(2x_1^2 + 2x_1x_2 + x_2^2\Big) - 3x_1 - 2x_2 \tag{38}$$

The function $f(x_1, x_2)$ trace a paraboloid into 3-dimension and the vertex is the minimum value of the paraboloid which is at $(1, 1)$ and is the solution of (37). This method yields potential solutions through iterative steps utilizing *steepest descent* process. In steepest descent, the iterative solution starts at an arbitrary point $\mathbf{x}_0$ and then take a series of iterations $\mathbf{x}_1$, $\mathbf{x}_2$, ... The process stops when $\mathbf{x}_n$ is sufficiently close to the solution $\mathbf{x}^\star$.

For any estimate $\mathbf{x}_n$, the solution $\mathbf{x}^\star$ can be found by taking the direction of steepest descent *i.e.* the quadratic form (36) decreases most rapidly. This direction is given by $-\nabla f(\mathbf{x})$.

$$\begin{aligned} -\nabla f(\mathbf{x}) &= -\nabla\Big(\frac{1}{2}\langle \mathbf{x}, \mathbf{A}\mathbf{x}\rangle - \langle \mathbf{b}\mathbf{x}\rangle\Big) \\ &= -\nabla\Big(\frac{1}{2}\,\mathbf{x}^T \cdot \mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}\Big) = -\Big(\frac{1}{2}\mathbf{A}^T\mathbf{x} + \frac{1}{2}\mathbf{A}\mathbf{x} - \mathbf{b}\Big) \end{aligned} \tag{39}$$

where we have used the properties

$$\frac{\partial}{\partial \mathbf{x}}\left(\mathbf{b}^T\mathbf{x}\right) = \mathbf{b} \qquad \text{and} \qquad \frac{\partial}{\partial \mathbf{x}}\left(\mathbf{x}^T\mathbf{A}\mathbf{x}\right) = \left(\mathbf{A}^T + \mathbf{A}\right)\mathbf{x} \tag{40}$$

Since $\mathbf{A}$ is symmetric, the direction of steepest descent (39) becomes,

$$-\nabla f(\mathbf{x}) = -(\mathbf{A}\mathbf{x} - \mathbf{b}) = \mathbf{b} - \mathbf{A}\mathbf{x} = \mathbf{r} \tag{41}$$

which is equal to the definition of *residual vector*

$$\mathbf{r}_n = \mathbf{b} - \mathbf{A}\mathbf{x}_n \tag{42}$$

i.e. how far the estimated value of $\mathbf{A}\mathbf{x}_n$ (and hence the solution) is away from $\mathbf{b}$ ($\mathbf{x}^\star$). Therefore, the residual vector $\mathbf{r}$ is the direction vector of steepest descent. Since we should ideally be traveling in the direction of steepest descent for absolute convergence, each consecutive step is some scalar multiple of the direction of steepest descent. If we start with guess vector $\mathbf{x}_0$ and the direction of steepest descent is $\mathbf{r}_0$, then the first step is

$$\mathbf{x}_1 = \mathbf{x}_0 + \boldsymbol{\alpha}_0\mathbf{r}_0 \tag{43}$$

where the coefficient or scaling factor $\boldsymbol{\alpha}$ determines how we descend in the direction steepest descent in each iteration. An important point is not to treat $\boldsymbol{\alpha}$ as a vector but a collection of scaling factors for each component of $\mathbf{r}$. For successive steps, the solution goes as

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \boldsymbol{\alpha}_i\mathbf{r}_i \tag{44}$$

The coefficients $\boldsymbol{\alpha}$ are chosen so as to minimize the $f(\mathbf{x})$ along the direction of steepest descent. For

the first step taken $-\nabla f(\mathbf{x}_1) = -\mathbf{r}_1$ and hence

$$\begin{aligned}
\frac{d}{d\boldsymbol{\alpha}} f(\mathbf{x}_1) &= 0 \\
\left[\nabla f(\mathbf{x}_1)\right]^T \frac{d}{d\boldsymbol{\alpha}} \mathbf{x}_1 &= 0 \\
\mathbf{r}_1^T \mathbf{r}_0 &= 0 \\
\left(\mathbf{b} - \mathbf{A}\mathbf{x}_1\right)^T \mathbf{r}_0 &= 0 \\
\Big(\mathbf{b} - \mathbf{A}\left(\mathbf{x}_0 + \boldsymbol{\alpha}\mathbf{r}_0\right)\Big)^T \mathbf{r}_0 &= 0 \\
\left(\mathbf{b} - \mathbf{A}\mathbf{x}_0\right)^T \mathbf{r}_0 &= \boldsymbol{\alpha}\left(\mathbf{A}\mathbf{r}_0\right)^T \mathbf{r}_0 \\
\mathbf{r}_0^T \mathbf{r}_0 &= \boldsymbol{\alpha}\mathbf{r}_0^T \left(\mathbf{A}\mathbf{r}_0\right) \\
\boldsymbol{\alpha} &= \frac{\mathbf{r}_0^T \mathbf{r}_0}{\mathbf{r}_0^T \left(\mathbf{A}\mathbf{r}_0\right)} = \frac{\langle \mathbf{r}_0, \mathbf{r}_0 \rangle}{\langle \mathbf{r}_0, \mathbf{A}\mathbf{r}_0 \rangle}
\end{aligned} \tag{45}$$

The expression for the i-th step gives us

$$\boldsymbol{\alpha}_i = \frac{\langle \mathbf{r}_i, \mathbf{r}_i \rangle}{\langle \mathbf{A}\mathbf{r}_i, \mathbf{r}_i \rangle} \tag{46}$$

The above steepest descent algorithm involves two matrix products $\mathbf{A}\mathbf{x}_n$ and $\mathbf{A}\mathbf{r}_n$. Of these the product $\mathbf{A}\mathbf{x}_n$ can be eliminated by

$$\begin{aligned}
\mathbf{x}_{i+1} &= \mathbf{x}_i + \boldsymbol{\alpha}_i \mathbf{r}_i \\
-\mathbf{A}\,\mathbf{x}_{i+1} &= -\mathbf{A}\mathbf{x}_i - \boldsymbol{\alpha}_i \mathbf{A}\mathbf{r}_i \\
-\mathbf{A}\,\mathbf{x}_{i+1} + \mathbf{b} &= -\mathbf{A}\mathbf{x}_i + \mathbf{b} - \boldsymbol{\alpha}_i \mathbf{A}\mathbf{r}_i \\
\mathbf{r}_{i+1} &= \mathbf{r}_i - \boldsymbol{\alpha}_i \mathbf{A}\mathbf{r}_i
\end{aligned} \tag{47}$$

Therefore, we need to compute only $\mathbf{A}\mathbf{r}_i$ (except of course the first computation of $\mathbf{r}_0$). Although this decreases computational cost, it increases the error on the other hand. It looks like we may have to compute new $\mathbf{x}_i$ at regular intervals to help reduce the error. It also converges slowly as the algorithm always follows the negative gradient and thereby the direction of steepest descent is mostly in the previously searched directions. In order to travel in directions that are never taken in the past, one chooses a set of nonzero vectors $\{\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}\}$ such that

$$\langle \mathbf{d}_i,\, \mathbf{A}\mathbf{d}_j \rangle = \delta_{ij} \tag{48}$$

This is known as the *A-orthogonality condition*. This amounts in re-defining the scaling coefficient $\boldsymbol{\alpha}$ in (46),

$$\boldsymbol{\alpha}_i = \frac{\langle \mathbf{d}_i, \mathbf{r}_i \rangle}{\langle \mathbf{d}_i, \mathbf{A}\mathbf{d}_i \rangle} \tag{49}$$

In terms of $\mathbf{d}$'s thus defined, the iterative steps (44) and (47) are rewritten as

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \boldsymbol{\alpha}_i \mathbf{d}_i \tag{50}$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \boldsymbol{\alpha}_i \mathbf{A}\mathbf{d}_i \tag{51}$$

Considering a set of A-orthogonal set of nonzero vectors $\{\mathbf{d}_0, \ldots, \mathbf{d}_{n-1}\}$, satisfying (48) associated with positive definite symmetric matrix $\mathbf{A}$, then the above scaling factor (49) and iteration steps (50) also leads to the solution $\mathbf{A}\mathbf{x}_n = \mathbf{b}$. These $\mathbf{d}$'s are called *Conjugate Direction*.

The Conjugate Gradient is a Conjugate Direction method that selects mutually orthogonal the residual vectors $\mathbf{r}_i$, as defined in (42), $\langle \mathbf{r}_i, \mathbf{r}_j \rangle = \delta_{ij}$. The algorithm starts with the initial guess $\mathbf{x}_0$ and the corresponding conjugate direction and residual are $\mathbf{d}_0 = \mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$. The subsequent search directions are computed using the residual from the previous iteration,

$$\mathbf{d}_{i+1} = \mathbf{r}_i + \boldsymbol{\beta}_i \mathbf{d}_i \tag{52}$$

For the A-orthogonal condition to satisfy $\boldsymbol{\beta}_i$ is so chosen such that

$$\langle \mathbf{d}_{i+1}, \mathbf{A}\mathbf{d}_i \rangle = 0 \quad \text{and} \quad \langle \mathbf{r}_i, \mathbf{r}_j \rangle = \delta_{ij} = \langle \mathbf{d}_i, \mathbf{r}_j \rangle \tag{53}$$

The $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}_i$ are given by

$$\boldsymbol{\alpha}_i = \frac{\langle \mathbf{d}_i, \mathbf{A}\mathbf{r}_i \rangle}{\langle \mathbf{d}_i, \mathbf{A}\mathbf{d}_i \rangle} = \frac{\langle \mathbf{r}_{i-1}, \mathbf{r}_{i-1} \rangle}{\langle \mathbf{d}_i, \mathbf{A}\mathbf{d}_i \rangle} \tag{54}$$

$$\boldsymbol{\beta}_i = -\frac{\langle \mathbf{d}_i, \mathbf{A}\mathbf{r}_i \rangle}{\langle \mathbf{d}_i, \mathbf{A}\mathbf{d}_i \rangle} = \frac{\langle \mathbf{r}_i, \mathbf{r}_i \rangle}{\langle \mathbf{r}_{i-1}, \mathbf{r}_{i-1} \rangle} \tag{55}$$

The following is the pseudo-code for Conjugate Gradient algorithm,

1. Initial guess $\mathbf{x}_0$:
   $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ and $\mathbf{d}_0 = \mathbf{r}_0$ and set $i = 1$
2. While $i \leq n$, where n is that of $n \times n$ matrix,

$$\begin{aligned} \boldsymbol{\alpha}_i &= \frac{\langle \mathbf{r}_{i-1}, \mathbf{r}_{i-1} \rangle}{\langle \mathbf{d}_i, \mathbf{A}\mathbf{d}_i \rangle} \\ \mathbf{x}_{i+1} &= \mathbf{x}_i + \boldsymbol{\alpha}_i \mathbf{d}_i \\ \mathbf{r}_{i+1} &= \mathbf{r}_i - \boldsymbol{\alpha}_i \mathbf{A}\mathbf{d}_i \end{aligned}$$

   If $||\mathbf{r}_i|| < \epsilon$, where ϵ is the tolerance, quit.
3. Else

$$\begin{aligned} \boldsymbol{\beta}_i &= \frac{\langle \mathbf{r}_i, \mathbf{r}_i \rangle}{\langle \mathbf{r}_{i-1}, \mathbf{r}_{i-1} \rangle} \\ \mathbf{d}_{i+1} &= \mathbf{r}_i + \boldsymbol{\beta}_i \mathbf{d}_i \\ i &= i + 1 \end{aligned}$$

The convergence is guaranteed within n steps unless round-off error plays spoilsport. To ensure convergence, *precondition* the matrix $\mathbf{A}$ using methods like *incomplete Cholesky factorization* or *even-odd preconditioning*. There are many variants of preconditioning and flavors of conjugate gradient, the most important of which is *(stabilized) bi-conjugate gradient* employed when the matrix $\mathbf{A}$ is not necessarily positive definite or symmetric.

Generalized Minimal Residue : It might appear a bit odd that much emphasis is put on the residuals $\mathbf{r}$, as in (54) and (55) or even in (52), instead of the possible solution $\mathbf{x}$. One reason is that $\mathbf{r}$'s are orthogonal to the previous search directions $\mathbf{d}$ as shown in (53), hence each step always produce a new, linearly independent search direction unless the residual is zero, in which case the problem is solved. The residuals are also orthogonal to all the previous residuals (53). From (52), we can construct an i-dimensional subspace $\mathcal{K}_i$,

$$\mathcal{K}_i = \text{span } \{\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{i-1}\} \tag{56}$$

From (51) we know that each residual $\mathbf{r}_i$ is a linear combination of the previous residual and $\mathbf{Ad}$, they too form a subspace which is equal to $\mathcal{K}$

$$\text{span } \{\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{i-1}\} = \mathcal{K}_i \tag{57}$$

Since $\mathbf{d}_{i-1} \in \mathcal{K}_i$, each new subspace $\mathcal{K}_{i+1}$ is formed from the union of the previous subspace $\mathcal{K}_i$ and the subspace $\mathbf{A}\mathcal{K}_i$. Hence,

$$\begin{aligned} \mathcal{K}_i &= \text{span } \left\{\mathbf{d}_0, \mathbf{A}\mathbf{d}_0, \mathbf{A}^2\mathbf{d}_0, \ldots, \mathbf{A}^{i-1}\mathbf{d}_0\right\} &(58)\\ &= \text{span } \left\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \ldots, \mathbf{A}^{i-1}\mathbf{r}_0\right\} &(59)\end{aligned}$$

This subspace is called *Krylov subspace*, a subspace created by repeatedly applying a matrix to a vector. Because $\mathbf{A}\mathcal{K}_i$ is included in $\mathcal{K}_{i+1}$ and $\mathbf{r}_{i+1}$ is orthogonal to $\mathcal{K}_{i+1}$ (53) imply that $\mathbf{r}_{i+1}$ is A-orthogonal to $\mathcal{K}_i$.

The idea behind GMRES method is to solve a least square problem [4] *i.e.* find an approximate solution $\mathbf{x}_i \in \mathcal{K}_i$ such that the norm of the residual $\mathbf{r}_i$,

$$||\mathbf{r}_i|| = ||\mathbf{A}\mathbf{x}_i - \mathbf{b}|| \tag{60}$$

is minimized. We start with the i-th order Krylov subspace, whose matrix representation is $i \times n$ matrix, formed by repeated multiplication of $\mathbf{A}$ on $\mathbf{b}$,

$$K_i = \left[\mathbf{b}, \mathbf{A}\mathbf{b}, \mathbf{A}^2\mathbf{b}, \ldots, \mathbf{A}^{i-1}\mathbf{b}\right] \text{ where } K \in \mathbb{R}^{i\times n},\ \mathbf{A} \in \mathbb{R}^{n\times n},\ \mathbf{b} \in \mathbb{R}^n \tag{61}$$

and the desired solution is

$$\mathbf{x}_i \in \mathcal{K}_i \ \rightarrow\ \mathbf{x}_i = K_i\mathbf{c} \ \Rightarrow\ ||\mathbf{A}K_i\mathbf{c} - \mathbf{b}||_{\min}, \text{ where } \mathbf{c} \in \mathbb{R}^n \tag{62}$$

and $\mathbf{c}$ is some scalar multipliers (similar to $\boldsymbol{\alpha}$). The columns of K_i are ill-conditioned *i.e.* close to linearly dependent. Hence *Arnoldi iteration* is used to find orthonormal basis vectors $\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n\}$ that forms the column of a matrix $\mathcal{Q}_n$, where the orthogonal columns of $\mathcal{Q}_i$ span $\mathcal{K}_i$. Therefore, instead of $\mathbf{x}_i = K_i\mathbf{c}$ we use,

$$\mathbf{x}_i = \mathcal{Q}_i\mathbf{y} \ \Rightarrow\ ||\mathbf{A}\mathcal{Q}_i\mathbf{y} - \mathbf{b}||_{\min}, \text{ where } \mathbf{y} \in \mathbb{R}^n \tag{63}$$

We need a bit of diversion at this point to discuss a few things that will be helpful to understand the working of GMRES algorithm.

QR decomposition : Like LU and Cholesky decomposition or factorization, QR decomposition can be used to solve system of linear equation (1). The idea is,

$$\mathbf{A} = \mathbf{Q}\cdot\mathbf{R}, \text{ where } \mathbf{Q}^T\mathbf{Q} \text{ or } \mathbf{Q}^\dagger\mathbf{Q} = 1 \tag{64}$$

i.e. $\mathbf{Q}$ is orthogonal, meaning its columns are orthogonal unit vectors. $\mathbf{R}$ is a upper triangular matrix (7). The decomposition is unique if $\mathbf{R}_{ii} > 0$. Then the linear equation (1) is solved by back substitution,

$$\mathbf{QR}\cdot\mathbf{x} = \mathbf{b} \ \Rightarrow\ \mathbf{R}\cdot\mathbf{x} = \mathbf{Q}^T\mathbf{b} \tag{65}$$

Since QR is two times more expensive than LU, it is usually not used for solving equations as above. QR factorization is mostly obtained using *Householder transformation* but also with *Gram Schmidt* algorithm. Although numerically unstable, we will describe Gram-Schmidt very briefly since it has got some bearings to what happens next. This computes $\mathbf{Q}$ and $\mathbf{R}$ column by column. Denote the column of the matrix $\mathbf{A}$ as,

$$\mathbf{A} = \left[\mathbf{a}_1\,|\mathbf{a}_2|\ldots|\mathbf{a}_n\,\right] \tag{66}$$

Then the QR factorization proceeds as

$$\begin{aligned}
\mathbf{u}_1 &= \mathbf{a}_1, & \mathbf{e}_1 &= \frac{\mathbf{u}_1}{||\mathbf{u}_1||} \\
\mathbf{u}_2 &= \mathbf{a}_2 - (\mathbf{a}_2 \cdot \mathbf{e}_1)\,\mathbf{e}_1, & \mathbf{e}_2 &= \frac{\mathbf{u}_2}{||\mathbf{u}_2||} \\
&\vdots & &\vdots \\
\mathbf{u}_{i+1} &= \mathbf{a}_{i+1} - (\mathbf{a}_{i+1} \cdot \mathbf{e}_1)\,\mathbf{e}_1 - \ldots - (\mathbf{a}_{i+1} \cdot \mathbf{e}_i)\,\mathbf{e}_i, & \mathbf{e}_{i+1} &= \frac{\mathbf{u}_{i+1}}{||\mathbf{u}_{i+1}||}
\end{aligned} \tag{67}$$

Therefore, resulting QR factorization by Gram-Schmidt orthogonalization process is

$$\mathbf{A} = \Big[\mathbf{a}_1\,|\mathbf{a}_2|\ldots|\mathbf{a}_n\Big] = \Big[\mathbf{e}_1\,|\mathbf{e}_2|\ldots|\mathbf{e}_n\Big] \times \begin{bmatrix} \mathbf{a}_1\cdot\mathbf{e}_1 & \mathbf{a}_2\cdot\mathbf{e}_1 & \ldots & \mathbf{a}_n\cdot\mathbf{e}_1 \\ 0 & \mathbf{a}_2\cdot\mathbf{e}_2 & \ldots & \mathbf{a}_n\cdot\mathbf{e}_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \mathbf{a}_n\cdot\mathbf{e}_n \end{bmatrix} \equiv \mathbf{QR} \tag{68}$$

Hessenberg matrix : In linear algebra, Hessenberg matrix is a square matrix which is *almost* triangular – *upper Hessenberg matrix* has zero entries below the first subdiagonal and *lower Hessenberg matrix* has zero entries above the first superdiagonal. For an $n \times n$ matrix, in upper Hesenberg we have $a_{ij} = 0$ for $i > j + 1$ and for lower Hessenberg we have $a_{ij} = 0$ for $j > i + 1$,

$$\text{upper Hessenberg} \quad = \quad \begin{bmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1\,n-1} & a_{1n} \\ a_{21} & a_{22} & a_{23} & \ldots & a_{2\,n-1} & a_{2n} \\ 0 & a_{32} & a_{33} & \ldots & a_{3\,n-1} & a_{3n} \\ 0 & 0 & a_{43} & \ldots & a_{4\,n-1} & a_{4n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & a_{n\,n-1} & a_{nn} \end{bmatrix} \tag{69}$$

$$\text{lower Hessenberg} \quad = \quad \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \ldots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2\,1} & a_{n-2\,2} & a_{n-2\,3} & \ldots & a_{n-2\,n-1} & 0 \\ a_{n-1\,1} & a_{n-1\,2} & a_{n-1\,3} & \ldots & a_{n-1\,n-1} & a_{n-1\,n} \\ a_{n1} & a_{n2} & a_{n3} & \ldots & a_{n\,n-1} & a_{nn} \end{bmatrix} \tag{70}$$

Triangular matrices are computationally less expensive to deal with and this applies to the Hessenberg matrices as well. If for certain constraints a matrix cannot be reduced to a triangular one then the next best option is reduction to Hessenberg form. Reduction of any matrix to Hessenberg form can be achieved in a finite number of steps, most often through Householder transformation.

Arnoldi Iteration : The last piece that we need before we return to GMRES is Arnoldi Iteration. The discussion closely follows the reference [5]. Consider factorization of $\mathbf{A}$ by similarity transformation

$$\mathbf{A} = \mathbf{Q}\,\mathbf{H}\,\mathbf{Q}^* \quad \Rightarrow \quad \mathbf{AQ} = \mathbf{QH} \tag{71}$$

where $\mathbf{H}$ is an upper Hessenberg matrix and $\mathbf{Q}$ an unitary matrix. $\mathbf{A}$, $\mathbf{Q}$ and $\mathbf{H}$ are all $n \times n$ matrices.

$$\mathbf{Q} = \left[\mathbf{q}_1 \middle| \mathbf{q}_2 \middle| \cdots \middle| \mathbf{q}_n \right] \tag{72}$$

$$\mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \cdots & \cdots & h_{1m} & \cdots & \cdots & h_{1n} \\ h_{21} & h_{22} & h_{23} & \cdots & \cdots & h_{2m} & \cdots & \cdots & h_{2n} \\ 0 & h_{32} & h_{33} & \cdots & \cdots & h_{3m} & \cdots & \cdots & h_{3n} \\ 0 & 0 & h_{43} & \cdots & \cdots & h_{4m} & \cdots & \cdots & h_{4n} \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & h_{m\,m-1} & h_{mm} & h_{m\,m+1} & \cdots & h_{mn} \\ 0 & 0 & 0 & \cdots & 0 & h_{m+1\,m} & h_{m+1\,m+1} & \cdots & h_{m+1\,n} \\ \vdots & \vdots & \vdots & \ddots & \ddots & 0 & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \cdots & \cdots & 0 & h_{n\,n-1} & h_{nn} \end{bmatrix} \tag{73}$$

where we have written $\mathbf{Q}$ in terms of its columns $\mathbf{q}_i$ and obviously $m < n$. Consider the following $\mathbf{Q}^{n\times m}$ and $\mathbf{Q}^{n\times m+1}$ matrices,

$$\mathbf{Q}_m = \left[\mathbf{q}_1 \middle| \mathbf{q}_2 \middle| \cdots \middle| \mathbf{q}_m \right] \tag{74}$$

$$\mathbf{Q}_{m+1} = \left[\mathbf{q}_1 \middle| \mathbf{q}_2 \middle| \cdots \middle| \mathbf{q}_m \middle| \mathbf{q}_{m+1} \right] \tag{75}$$

$$\tag{76}$$

The corresponding $(m+1) \times m$ Hessenberg matrix $\tilde{\mathbf{H}}$ is

$$\tilde{\mathbf{H}}_m = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \cdots & \cdots & h_{1m} \\ h_{21} & h_{22} & h_{23} & \cdots & \cdots & h_{2m} \\ 0 & h_{32} & h_{33} & \cdots & \cdots & h_{3m} \\ 0 & 0 & h_{43} & h_{44} & \cdots & h_{4m} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \vdots & h_{m\,m-1} & h_{mm} \\ 0 & 0 & \cdots & \cdots & 0 & h_{m+1\,m} \end{bmatrix} \tag{77}$$

Thus we have the following relation,

$$\mathbf{A}^{n\times n}\,\mathbf{Q}_m^{n\times m} = \mathbf{Q}_{m+1}^{n\times m+1} \cdot \tilde{\mathbf{H}}_m^{m+1\,m} \quad i.e.\ \ \mathbf{A} \cdot \mathbf{Q}_m = \mathbf{Q}_{m+1} \cdot \tilde{\mathbf{H}}_m \tag{78}$$

The m-th column in the RHS of the above equation (78) can be written as,

$$\begin{bmatrix} q_{11}h_{1m} + q_{12}h_{2m} + \cdots + q_{1\,m+1}h_{m+1\,m} \\ q_{21}h_{1m} + q_{22}h_{2m} + \cdots + q_{2\,m+1}h_{m+1\,m} \\ \vdots \\ q_{n1}h_{1m} + q_{n2}h_{2m} + \cdots + q_{n\,m+1}h_{m+1\,m} \end{bmatrix} =$$

$$h_{1m}\begin{pmatrix} q_{11} \\ q_{21} \\ \vdots \\ q_{n1} \end{pmatrix} + h_{2m}\begin{pmatrix} q_{12} \\ q_{22} \\ \vdots \\ q_{n2} \end{pmatrix} + \cdots + h_{m+1\,m}\begin{pmatrix} q_{1\,m+1} \\ q_{2\,m+1} \\ \vdots \\ q_{n\,m+1} \end{pmatrix}$$

$$= \quad h_{1m}\mathbf{q}_1 + h_{2m}\mathbf{q}_2 + \cdots + h_{m+1\,m}\mathbf{q}_{m+1} \tag{79}$$

Comparing m-th column on both sides of the above two equations (78) and (79), we get

$$\mathbf{A}\mathbf{q}_m = h_{1m}\mathbf{q}_1 + h_{2m}\mathbf{q}_2 + \cdots + h_{mm}\mathbf{q}_m + h_{m+1\,m}\mathbf{q}_{m+1} \tag{80}$$

This leads to the recursion relation

$$\mathbf{q}_{m+1} = \frac{\mathbf{A}\mathbf{q}_m - \sum_{j=1}^{m} h_{jm}\mathbf{q}_j}{h_{m+1\,m}} \tag{81}$$

This recursive computation of the columns of the unitary matrix $\mathbf{Q}$ is known as *Arnoldi iteration*. To understand the iteration well, let us start with an $\mathbf{A}$ and an arbitrary normalized vector $\mathbf{q}_1$ *i.e.* $\mathbf{q}_1^*\mathbf{q}_1 = 1$. Then

$$\mathbf{q}_2 = \frac{\mathbf{A}\mathbf{q}_1 - h_{11}\mathbf{q}_1}{h_{21}} \tag{82}$$

Since we need orthogonal columns of $\mathbf{Q}$,

$$\mathbf{q}_1^*\mathbf{q}_2 = 0 \;\;\Rightarrow\;\; \mathbf{q}_1^*\mathbf{A}\mathbf{q}_1 - h_{11}\mathbf{q}_1^*\mathbf{q}_1 \;\;\Rightarrow\;\; h_{11} = \mathbf{q}_1^*\mathbf{A}\mathbf{q}_1 \tag{83}$$

We can normalize $\mathbf{q}_2$ in a straight forward way by setting

$$h_{21} = ||\mathbf{A}\mathbf{q}_1 - h_{11}\mathbf{q}_1|| \tag{84}$$

Arnoldi iteration is thus very much analogous to Gram-Schmidt ortho normalizing process. Its major advantage is it computes columns of the unitary matrix $\mathbf{Q}$ one at a time and, in addition, it performs partial QR factorization (which comes in handy if we are triangularizing a huge $\mathbf{A}$ matrix). However, we have to keep in mind that the most expensive operation in the algorithm is the matrix-vector product $\mathbf{A}\mathbf{q}$ and, therefore, we must use problem specific method to multiply. The pseudo-code for Arnoldi iteration is

```
Choose v an arbitrary initial vector
q_1 = v/||v||
for m = 1, 2, 3, ...
    w = A q_m
    for  j = 1, 2, ... m
         h_jm = q_j^* w
         w = w - h_jm q_j
    end
    h_{m+1 m} = ||w||
    q_{m+1} = w / h_{m+1 m}
end
```

We now get back to our GMRES equation (63), the minimal residual is

$$\Big|\Big|\mathbf{r}_n\Big|\Big| = \Big|\Big|\mathbf{A}\mathcal{Q}_n\mathbf{y} - \mathbf{b}\Big|\Big|_{\min} = \Big|\Big|\mathcal{Q}_{n+1}\tilde{\mathbf{H}}_n\mathbf{y} - \mathbf{b}\Big|\Big|_{\min} \tag{85}$$

Left multiplication by $\mathcal{Q}_{n+1}^*$ does not change the norm, since both vectors are in the column space of $\mathcal{Q}_{n+1}$. Assuming $\mathbf{q}_1 = \mathbf{b}/||\mathbf{b}||$ and using (79) $\mathcal{Q}_{n+1}\mathbf{q}_1 = \mathbf{e}_1$, where $\mathbf{e}_1 = (1, 0, 0, \ldots, 0)^T$, we have

$$\Big|\Big|\tilde{\mathbf{H}}_n\mathbf{y} - \mathcal{Q}_{n+1}^*\mathbf{b}\Big|\Big|_{\min} = \Big|\Big|\tilde{\mathbf{H}}_n\mathbf{y} - \mathcal{Q}_{n+1}^*\mathbf{q}_1||\mathbf{b}||\;\Big|\Big|_{\min} = \Big|\Big|\tilde{\mathbf{H}}_n\mathbf{y} - ||\mathbf{b}||\,\mathbf{e}_1\Big|\Big|_{\min} \tag{86}$$

Hence, $\mathbf{x}_n$ can be found by minimizing the Euclidean norm of the residual $\mathbf{r}_n = \tilde{\mathbf{H}}_n\mathbf{y}_n - ||\mathbf{r}_0||\,\mathbf{e}_1$. So the GMRES algorithm reads

Let $\mathbf{q}_1 = \mathbf{r}_0//||\mathbf{r}_0||$
for $n = 1, 2, 3, \ldots$
    compute $\mathbf{q}_n$ by Arnoldi iteration
    minimize $||\mathbf{r}_n||$ for a $\mathbf{y}_n$
    $\mathbf{x}_n = \mathcal{Q}_n \mathbf{y}_n$
end
Above the minimization has to be carried out by *least square method* like LU forward-backward.

References

[1] M.R. Hestenes and E. Stiefel, *Journal of Research of the National Bureau of Standards*, Vol 49, No 6 (1952) https://nvlpubs.nist.gov/nistpubs/jres/049/jresv49n6p409_A1b.pdf

[2] https://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf

[3] http://math.stmarys-ca.edu/wp-content/uploads/2017/07/Mike-Rambo.pdf

[4] https://web.stanford.edu/class/cme324/saad-schultz.pdf

[5] http://http://www.math.iit.edu/~fass/477577_Chapter_14.pdf