Linear Algebra : Matrices

In linear algebra, a commonly encountered situation is where we need to solve matrix equation of the form

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

where $\mathbf{A}$ is an $n \times n$ matrix, $\mathbf{x}$, $\mathbf{b}$ can either be $n \times 1$ vectors (as in solving system of linear equations) or $n \times n$ matrices (as in inverse).

The solution implies determining the vector $\mathbf{x}$

$$\mathbf{x} = \mathbf{A}^{-1}\,\mathbf{b} \quad \text{where} \quad \mathbf{A}^{-1} = \mathbf{A}_c^T/|\mathbf{A}|$$

where $|\mathbf{A}|$ is the determinant and $\mathbf{A}_c$ are the cofactors. Although numerically doable, far more efficient methods exist to calculate inverse.

- Direct elimination : Gauss-Jordan, LU decomposition, Cholesky decomposition which are used for small, dense matrices and are memory intensive, possibly slower and difficult to parallelize.
- Iterative methods : Gauss-Jacobi, Gauss-Seidel, Conjugate gradient, GMRES which are useful for large, particularly sparse matrices and lend themselves to parallelization.

Linear equations: Gauss-Jordon method

Consider solving a linear system of equations, say with 3 variables

$$\mathbf{A}\cdot\mathbf{x}=\mathbf{b}\ \Rightarrow\ \begin{pmatrix} a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\\ a_{31} & a_{32} & a_{33}\end{pmatrix}\cdot\begin{pmatrix} x_1\\ x_2\\ x_3\end{pmatrix}=\begin{pmatrix} b_1\\ b_2\\ b_3\end{pmatrix}$$

To solve (x_1, x_2, x_3), the Gauss-Jordon method suggests to write the above equation in an *augmented matrix* form

$$\left[\mathbf{A}\middle|\mathbf{B}\right]\equiv\left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1\\ a_{21} & a_{22} & a_{23} & b_2\\ a_{31} & a_{32} & a_{33} & b_3\end{array}\right]$$

Using Gauss-Jordon elimination, convert the *augmented matrix* in *reduced row echelon form* (RREF) eventually yielding

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & \tilde{b}_1\\ 0 & 1 & 0 & \tilde{b}_2\\ 0 & 0 & 1 & \tilde{b}_3\end{array}\right]$$

thus solving the linear equations : $x_1=\tilde{b}_1,\ x_2=\tilde{b}_2,\ x_3=\tilde{b}_3$.

RREF

In *reduced row echelon form*,

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 row or column containing a leading 1 are zeros.

$$\text{Example : } \begin{pmatrix} 1 & -2 & 3 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{pmatrix}$$

The way to achieve RREF by Gauss-Jordon involves any one or a combination of three elementary row operations

- swapping two rows
- multiplying a row by a nonzero number
- adding or subtracting a multiple of one row to another row

Operation involving rows is called *partial pivoting*.

Partial pivoting

Say the augmented matrix that we want to do a RREF be

$$\left[\begin{array}{ccc|c} \boxed{0} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array}\right]$$

a_{11} is either 0 or order(s) of magnitude small compared to a_{21}, $a_{31} \rightarrow a_{31}, a_{21} \gg a_{11}$ and let $a_{31} > a_{21}$.

To get leftmost nonzero entry at the top, swap $R_3 \leftrightarrow R_1$

$$\left[\begin{array}{ccc|c} \boxed{a_{31}} & a_{32} & a_{33} & b_3 \\ a_{21} & a_{22} & a_{23} & b_2 \\ 0 & a_{12} & a_{13} & b_1 \end{array}\right] \equiv \left[\begin{array}{ccc|c} \boxed{a_{11}^0} & a_{12}^0 & a_{13}^0 & b_1^0 \\ a_{21}^0 & a_{22}^0 & a_{23}^0 & b_2^0 \\ a_{31}^0 & a_{32}^0 & a_{33}^0 & b_3^0 \end{array}\right]$$

where a_{11}^0 is the *pivot element*, new row R_1^0 is the *pivot row* and new column C_1^0 is the *pivot column*. Reduce $R_1^0/a_{11}^0 \rightarrow R_1^1$,

$$\left[\begin{array}{ccc|c} \boxed{1} & a_{12}^1 & a_{13}^1 & b_1^1 \\ a_{21}^0 & a_{22}^0 & a_{23}^0 & b_2^0 \\ a_{31}^0 & a_{32}^0 & a_{33}^0 & b_3^0 \end{array}\right], \quad a_{1,2,3}^1 = a_{1,2,3}^0/a_{11}^0,\ b_1^1 = b_1^0/a_{11}^0$$

Partial pivoting

Reduce the pivot column C_1 by adding / subtracting multiples of pivot row from the following rows

$$a^1_{2(1,2,3)} = a^0_{2(1,2,3)} - a^0_{21} * R^1_1, \qquad b^1_2 = b^0_2 - a^0_{21} * b^1_1$$
$$a^1_{3(1,2,3)} = a^0_{3(1,2,3)} - a^0_{31} * R^1_1, \qquad b^1_3 = b^0_3 - a^0_{31} * b^1_1$$

$$\Rightarrow \left[\begin{array}{ccc|c} 1 & a^1_{12} & a^1_{13} & b^1_1 \\ 0 & \boxed{a^1_{22}} & a^1_{23} & b^1_2 \\ 0 & a^1_{32} & a^1_{33} & b^1_3 \end{array}\right]$$

a^1_{22} is new *pivot element* and undergoes the same test of smallness. The test has to be performed with the rows that are below the new pivot row. If needed, partial pivoting done; eventually yielding

$$\text{RREF} \Rightarrow \left[\begin{array}{ccc|c} 1 & 0 & 0 & b^3_1 \\ 0 & 1 & 0 & b^3_2 \\ 0 & 0 & 1 & b^3_3 \end{array}\right]$$

This is Gauss-Jordan elimination using *partial pivoting*. No matter what steps and the order they are applied, final augmented matrix is unique.

An example :

Consider the equations and its corresponding augmented matrix

$$\left.\begin{array}{rcr} 2y+5z & = & 1 \\ 3x-y+2z & = & -2 \\ x-y+3z & = & 3 \end{array}\right\} \Rightarrow \left[\begin{array}{ccc|c} 0 & 2 & 5 & 1 \\ 3 & -1 & 2 & -2 \\ 1 & -1 & 3 & 3 \end{array}\right]$$

Since $a_{11} = 0$ and $a_{21} > a_{31}$, we begin by swapping $R_1 \leftrightarrow R_2$,

$$\left[\begin{array}{ccc|c} 3 & -1 & 2 & -2 \\ 0 & 2 & 5 & 1 \\ 1 & -1 & 3 & 3 \end{array}\right] \xrightarrow{R_1^0/3 \to R_1^1} \left[\begin{array}{ccc|c} 1 & -1/3 & 2/3 & -2/3 \\ 0 & 2 & 5 & 1 \\ 1 & -1 & 3 & 3 \end{array}\right]$$

$$\xrightarrow{R_3^0 - R_1^1 \to R_3^1} \left[\begin{array}{ccc|c} 1 & -1/3 & 2/3 & -2/3 \\ 0 & 2 & 5 & 1 \\ 0 & -2/3 & 7/3 & 11/3 \end{array}\right]$$

$$\xrightarrow{R_2^1/2 \to R_2^2} \left[\begin{array}{ccc|c} 1 & -1/3 & 2/3 & -2/3 \\ 0 & 1 & 5/2 & 1/2 \\ 0 & -2/3 & 7/3 & 11/3 \end{array}\right]$$

$$\xrightarrow{R_1^1 + R_2^2/3 \to R_1^2} \left[\begin{array}{ccc|c} 1 & 0 & 9/6 & -3/6 \\ 0 & 1 & 5/2 & 1/2 \\ 0 & -2/3 & 7/3 & 11/3 \end{array}\right]$$

$$\xrightarrow{R_3^1+2R_2^2/3\to R_3^2}\left[\begin{array}{ccc|c} 1 & 0 & 9/6 & -3/6 \\ 0 & 1 & 5/2 & 1/2 \\ 0 & 0 & 12/3 & 24/6 \end{array}\right]$$

$$\xrightarrow{3R_3^2/12\to R_3^3}\left[\begin{array}{ccc|c} 1 & 0 & 9/6 & -3/6 \\ 0 & 1 & 5/2 & 1/2 \\ 0 & 0 & 1 & 1 \end{array}\right] \xrightarrow{R_2^2-5R_3^3/2\to R_2^3}\left[\begin{array}{ccc|c} 1 & 0 & 9/6 & -3/6 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & 1 \end{array}\right]$$

$$\xrightarrow{R_1^2-9R_3^3/6\to R_1^3}\left[\begin{array}{ccc|c} 1 & 0 & 0 & -2 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & 1 \end{array}\right]$$

Therefore, the solution is $x = -2$, $y = -2$, $z = 1$.

Gauss-Jordon can be trivially extend to obtain inverse of an invertible matrix. First to test invertibility, determine the determinant.

$$\det\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \det\begin{pmatrix} a'_{11} & a'_{12} & a'_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & 0 & a'_{33} \end{pmatrix} = (-1)^n a'_{11}\, a'_{22}\, a'_{33}$$

where n is the number of times the rows are interchanged, $a'_{ii} \neq 0$ and none of a'_{ii} has to be 1.

Matrix inversion with Gauss-Jordon

Matrix inversion follows the method almost exactly as system of linear equations we seen before, except that we replace vector $\mathbf{b}$ with unit matrix $\mathbf{b} = \mathbb{1}$ and run through the previous steps towards RREF. At the end, the matrix $\mathbf{b}$ will contain the inverse.

$$\mathbf{A} \cdot \mathbf{b} = \mathbb{1} \rightarrow \text{augmented matrix} : \left[\mathbf{A} \middle| \mathbb{1}\right] \overset{G-J}{\Longrightarrow} \left[\mathbb{1} \middle| \mathbf{A}^{-1}\right]$$

Memory requirement : $N \times (N + 1)$ for solving linear system of equations and $N \times (N + N)$ for inverse, ignoring swapping operation (involving sorting).

Mathematical operations : $(N - 1) \times 2$ (multiplication / division and addition / subtraction) for each row reduction.

Accuracy : Rounding errors, proportional to N, quickly built up in solution

For large N not very suitable – memory requirement $\propto N^2$ and slow.

Typically restricted to $\sim N = 10$

Linear equations: LU decomposition

The method involves factorising of a square matrix **A**, with proper row and/or column pivoting, into two triangular matrices namely lower **L** and upper **L** matrices.

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \quad \rightarrow \quad \mathbf{A} = \mathbf{L} \cdot \mathbf{U}$$

Consider solving a linear system of equations, say with 3 variables

$$\mathbf{L} = \begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \qquad \mathbf{U} = \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}$$

Looks formidable but very useful, not nearly as hard. On multiplying,

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} l_{11}u_{11} & l_{11}u_{12} & l_{11}u_{13} \\ l_{21}u_{11} & l_{21}u_{12} + l_{22}u_{22} & l_{21}u_{13} + l_{22}u_{23} \\ l_{31}u_{11} & l_{31}u_{12} + l_{32}u_{22} & l_{31}u_{13} + l_{32}u_{23} + l_{33}u_{33} \end{pmatrix}$$

Without proper ordering, the factorisation may fail!. The matrix multiplication $\mathbf{L} \cdot \mathbf{U}$ yields

$$\begin{array}{lll} l_{11}u_{11} = a_{11} & l_{11}u_{12} = a_{12} & l_{11}u_{13} = a_{13} \\ l_{21}u_{11} = a_{21} & l_{21}u_{12} + l_{22}u_{22} = a_{22} & l_{21}u_{13} + l_{22}u_{23} = a_{23} \\ l_{31}u_{11} = a_{31} & l_{31}u_{12} + l_{32}u_{22} = a_{32} & l_{31}u_{13} + l_{32}u_{23} + l_{33}u_{33} = a_{33} \end{array}$$

A catch : $3 \times 3 = 9$ equations but $3 \times (3+1)$ variables!! Hence, no unique solutions for l_{ij} and u_{ij}.

Trick is to put either all three $l_{ii} = 1$, called *Doolittle* or all three $u_{ii} = 1$, called *Crout* decomposition. Any of the decomposition of **L** and **U** can proceed iteratively.

Take Doolittle factorization :

$$
\begin{aligned}
& & l_{11} &= l_{22} = l_{33} = 1 \\
u_{11} &= a_{11} & l_{21} &= a_{21}/u_{11} \\
u_{12} &= a_{12} & & \\
u_{13} &= a_{13} & l_{31} &= a_{31}/u_{11} \\
u_{22} &= a_{22} - l_{21}u_{12} & l_{32} &= \left(a_{32} - l_{31}u_{31}\right)/u_{22} \\
u_{23} &= a_{23} - l_{21}u_{13} & l_{12} &= l_{13} = l_{23} = 0 \\
u_{33} &= a_{33} - l_{31}u_{13} - l_{32}u_{23} & u_{21} &= u_{31} = u_{32} = 0
\end{aligned}
$$

The generic form for each of the **L** and **U** elements are

$$
\begin{aligned}
u_{ij} &= a_{ij} - \textstyle\sum_{k=1}^{i-1} l_{ik}u_{kj} && \text{for } i = 2, \ldots, j \\
l_{ij} &= \left(a_{ij} - \textstyle\sum_{k=1}^{j-1} l_{ik}u_{kj}\right)/u_{jj} && \text{for } i = j+1, j+2, \ldots, N
\end{aligned}
$$

But can we always LU factorise?

- Guaranteed if all leading submatrices have nonzero determinant

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 2 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \Rightarrow \mathbf{A_1} = 1, \mathbf{A_2} = \begin{pmatrix} 1 & 0 \\ -2 & 1 \end{pmatrix} \; \mathbf{A_3} = \begin{pmatrix} 1 & 0 & 2 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

$$\det[\mathbf{A_1}] = 1, \; \det[\mathbf{A_2}] = 1 \; \det[\mathbf{A_3}] = -3$$

- Pivoting if $a_{11} = 0$
- Additional pivoting if determinant of any leading submatrix is zero but the matrix itself is invertible.

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 2 \\ 3 & 0 & 2 \\ -1 & 1 & 2 \end{pmatrix} \Rightarrow \mathbf{A_1} = 1, \mathbf{A_2} = \begin{pmatrix} 1 & 0 \\ 3 & 0 \end{pmatrix} \; \mathbf{A_3} = \begin{pmatrix} 1 & 0 & 2 \\ 3 & 0 & 2 \\ -1 & 1 & 2 \end{pmatrix}$$

$$\det[\mathbf{A_1}] = 1, \; \det[\mathbf{A_2}] = 0 \; \det[\mathbf{A_3}] = 4$$

- If all the above fails, no solution exists and your are doomed!

The LU decomposition has an advantage over Gauss-Jordon – every a_{ij} is used only once and never again $\Rightarrow$ u_{ij}, l_{ij} can be stored in the same location / memory of a_{ij}, hence storage requirement is half.

Otherwise, the numerical costs of Gauss-Jordon and LU are the same $\mathcal{O}(N^3)$.

Storing LU decomposed matrix

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

No need to store $l_{ii} = 1$, modify the looping over indices when needed.

Once LU decomposed, the first step towards solution is splitting the original equation in two,

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \Rightarrow \mathbf{L} \cdot (\mathbf{U} \cdot \mathbf{x}) = \mathbf{b} \;\Big|\; \mathbf{U} \cdot \mathbf{x} = \mathbf{y} \Rightarrow \mathbf{L} \cdot \mathbf{y} = \mathbf{b}$$

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 $\mathbf{x}$ by backward substitution.

Forward substitution :

$$\begin{pmatrix} 1 & 0 & 0 \\ l_{12} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Starting from the first row *i.e.* moving forward we solve for y_i

$$\begin{aligned} y_1 &= b_1 & y_1 &= b_1 \\ l_{21} y_1 + y_2 &= b_2 & y_2 &= b_2 - l_{21} y_1 \\ l_{31} y_1 + l_{32} y_2 + y_3 &= b_3 & y_3 &= b_3 - l_{31} y_1 - l_{32} y_2 \end{aligned}$$

Backward substitution :

$$\begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

Starting from the last row *i.e.* moving backward to solve for x_i

$$\begin{aligned} u_{33}x_3 &= y_3 & x_3 &= y_3/u_{33} \\ u_{22}x_2 + u_{23}x_3 &= y_2 & x_2 &= \left(y_2 - u_{23}x_3\right)/u_{22} \\ u_{11}x_1 + u_{12}x_2 + u_{13}x_3 &= y_1 & x_1 &= \left(y_1 - u_{12}x_2 - u_{13}x_3\right)/u_{11} \end{aligned}$$

In generic form, the solutions for y_i and subsequently x_i are

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

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

For the inverse, use the combination of the above equations to iterate through each column of the identity matrix $\mathbf{b} = \mathbb{1}$.

Cholesky decomposition

Properties and symmetries of **A** are often used to simplify the process of solving linear equations.

An useful scheme for factorization of Hermitian, positive definite matrix is Cholesky decomposition. Example : covariance matrix, PDE finite element matrix

Matrix is factorised into a product of lower triangular matrix **L** and $\mathbf{L}^T$. For a 3×3 system

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

When **A** is real and positive definite,

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

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

Cholesky is about twice as efficient as the LU decomposition for solving system of linear equations.

Iterative method : Gauss-Jacobi

Gauss-Jordon and LU decomposition become unmanageable with increasing matrix size and increasing number of 0 entries. Pivoting becomes a costly affair.

Hence, need for iterative methods – Jacobi, Gauss-Seidel etc.

Jacobi works when **A** has no zeros in the main diagonal. It decomposes, but not factorises, **A** into a diagonal **D** and **L**, **U**

$$\mathbf{A} = \mathbf{D} + (\mathbf{L} + \mathbf{U}) = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix} + \begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{pmatrix}$$

Then the solution can be approached as

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \Rightarrow \Big[\mathbf{D} + (\mathbf{L} + \mathbf{U})\Big] \cdot \mathbf{x} = \mathbf{b} \Rightarrow \mathbf{x} = \mathbf{D}^{-1}\Big[\mathbf{b} - (\mathbf{L} + \mathbf{U}) \cdot \mathbf{x}\Big]$$

Above expression evaluated iteratively for $\mathbf{x}$ until $||\mathbf{x}^{k+1} - \mathbf{x}^{k}|| < \epsilon$,

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

Gauss-Jacobi method is fairly fast but computation of $x_i^{(k+1)}$ element requires all other elements, we cannot overwrite $x_i^{(k)}$ with $x_i^{(k+1)}$.

Sufficient condition for the method to converge is that the matrix **A** is strictly diagonally dominant,

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

Although Jacobi method is relatively slower to converge, 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.

As an example solve the following,

$$\begin{array}{rcrcl} 4x - y + z & = & 7 & & \\ -2x + y + 5z & = & 15 & \Rightarrow & \mathbf{A} = \begin{pmatrix} 4 & -1 & 1 \\ -2 & 1 & 5 \\ 4 & -8 & 1 \end{pmatrix} \\ 4x - 8y + z & = & -21 & & \end{array}$$

Above **A** is not diagonally dominant but swapping eqn. 2 with eqn. 3 *i.e.* row 2 with row 3 of **A** will make it so.

After swapping, **A** becomes

$$\mathbf{A} = \begin{pmatrix} 4 & -1 & 1 \\ 4 & -8 & 1 \\ -2 & 1 & 5 \end{pmatrix}$$

According to the above Jacobi formula $x_i^{(k+1)} = \cdots$ etc.

$$\begin{array}{lcl} x = \frac{7+y-z}{4} & & x^{(k+1)} = \frac{7+y^{(k)}-z^{(k)}}{4} \\ y = \frac{21+4x+z}{8} & \Rightarrow & y^{(k+1)} = \frac{21-4x^{(k)}-z^{(k)}}{8} \\ z = \frac{15+2x-y}{5} & & z^{(k+1)} = \frac{15+2x^{(k)}-y^{(k)}}{5} \end{array}$$

Start with initial guess $(x^{(0)}, y^{(0)}, z^{(0)}) = (0, 0, 0)$.

(k)	$x^{(k)}$	$y^{(k)}$	$z^{(k)}$
0	0	0	0
1	1.75	2.625	3.0
2	1.656	3.875	3.175
⋮	⋮	⋮	⋮
6	1.9995	3.996	2.9987

Iterative method : Gauss-Seidel

Gauss-Seidel method is defined by the relation

$$\mathbf{A} = \mathbf{L}_\star + \mathbf{U} \rightarrow \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} + \begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

The solution is obtained through the iterative process

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

Iterative process continues until the changes $||\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}|| < \epsilon$

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

Gauss-Seidel requires only one storage vector for $\mathbf{x}^{(k)}$ as against two in Jacobi, since computation of $\mathbf{x}^{(k+1)}$ uses elements already calculated $\mathbf{x}^{(k+1)}$ and those $\mathbf{x}^{(k)}$ that are already used in the previous iteration.

Convergence is only guaranteed if the matrix is either *strictly diagonally dominant* or *symmetric and positive definite*.

Let us try to understand Gauss-Seidel method through an explicit calculation in a 4×4 matrix

$$\text{RHS} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} - \begin{pmatrix} 0 & a_{12} & a_{13} & a_{14} \\ 0 & 0 & a_{23} & a_{24} \\ 0 & 0 & 0 & a_{34} \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \\ x_4^{(k)} \end{pmatrix}$$

$$= \begin{pmatrix} b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - a_{14}x_4^{(k)} \\ b_2 - a_{23}x_3^{(k)} - a_{24}x_4^{(k)} \\ b_3 - a_{34}x_4^{(k)} \\ b_4 \end{pmatrix} = b_i - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}$$

$$\text{LHS} = \begin{pmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \begin{pmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \\ x_4^{(k+1)} \end{pmatrix}$$

$$= \begin{pmatrix} a_{11}x_1^{(k+1)} \\ a_{21}x_1^{(k+1)} + a_{22}x_2^{(k+1)} \\ a_{31}x_1^{(k+1)} + a_{32}x_2^{(k+1)} + a_{33}x_3^{(k+1)} \\ a_{41}x_1^{(k+1)} + a_{42}x_2^{(k+1)} + a_{43}x_3^{(k+1)} a_{44}x_4^{(k+1)} \end{pmatrix} = a_{ii}x_i^{(k+1)} + \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)}$$

Equating LHS with RHS

$$
\begin{aligned}
a_{11}x_1^{(k+1)} &= b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - a_{14}x_4^{(k)} \\
a_{21}x_1^{(k+1)} a_{22}x_2^{(k+1)} &= b_2 - a_{23}x_3^{(k)} - a_{24}x_4^{(k)} \\
a_{31}x_1^{(k+1)} + a_{32}x_2^{(k+1)} + a_{33}x_3^{(k+1)} &= b_3 - a_{34}x_4^{(k)} \\
a_{41}x_1^{(k+1)} + a_{42}x_2^{(k+1)} + a_{43}x_3^{(k+1)} a_{44}x_4^{(k+1)} &= b_4
\end{aligned}
$$

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

Thus we see, $x_i^{(k)}$ is used only once and can be replaced with $x_i^{(k+1)}$.
More explicitly

$$
\begin{aligned}
x_1^{(k+1)} &= \frac{1}{a_{11}}\left(b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - a_{14}x_4^{(k)}\right) \\
x_2^{(k+1)} &= \frac{1}{a_{22}}\left(b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)} - a_{24}x_4^{(k)}\right) \\
x_3^{(k+1)} &= \frac{1}{a_{33}}\left(b_3 - a_{31}x_1^{(k+1)} - a_{32}x_2^{(k+1)} - a_{34}x_4^{(k)}\right) \\
x_4^{(k+1)} &= \frac{1}{a_{44}}\left(b_4 - a_{41}x_1^{(k+1)} - a_{42}x_2^{(k+1)} - a_{43}x_3^{(k+1)}\right) \\
\textit{i.e. } x_i^{(k+1)} &= \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right)
\end{aligned}
$$

Sparse matrices

Jacobi and Gauss-Seidel emphasize on non-zero diagonal matrix but not talk about the off-diagonal elements. If matrices are large and sparse *i.e.* plenty of zero in off-diagonals, then iterative methods are best choice.

Consider, for instance, a simplified Ising model Hamiltonian,

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

where (m, n) pairs are nearest neighbors, J is spin-spin coupling, h represents 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

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

It has more than 50% zero entries.

Almost similar situation arises with fermion on discrete Euclidean space-time lattice. Consider just the free Euclidean Dirac action,

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

where partial derivative is replaced with finite, symmetric difference. This immediately suggests sparse form of fermion matrix. The fermion propagator in QFT is the inverse of this fermion matrix.

This fermion matrix is not only very sparse, but also very large. Fermion fields $\psi_k^\alpha(x)$ have $k = 4$ Dirac indices, $\alpha = 3$ color indices and they are complex, hence 2 more components. On a modest 8^4, the total degrees of freedom is $8^4 \times 4 \times 3 \times 2 = 98,304$.

Hence, the problem is to invert a $10^5 \times 10^5$, sparse matrix! DIY the maths for memory requirement.

Two most widely used methods are Conjugate Gradient (and its various flavors) and Minimal Residue. Somewhat less popular is Gauss-Seidel.

Among various flavors of conjugate gradient, the Stabilised bi-Conjugate Gradient is an extremely useful algorithm as it helps inverting any sparse matrix not necessarily hermitian or positive definite.

Conjugate Gradient

Consider **A** to be symmetric and positive definite matrix, *i.e.* for every nonzero vector **x**,

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

A Hermitian matrix is a positive definite matrix. The following statement will be taken for granted without proof,

The solution vector $\mathbf{x}^\star$ *of the equation* $\mathbf{A}\mathbf{x} = \mathbf{b}$ *where* **A** *is positive definite and symmetric, is the minimal value of the quadratic form*

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

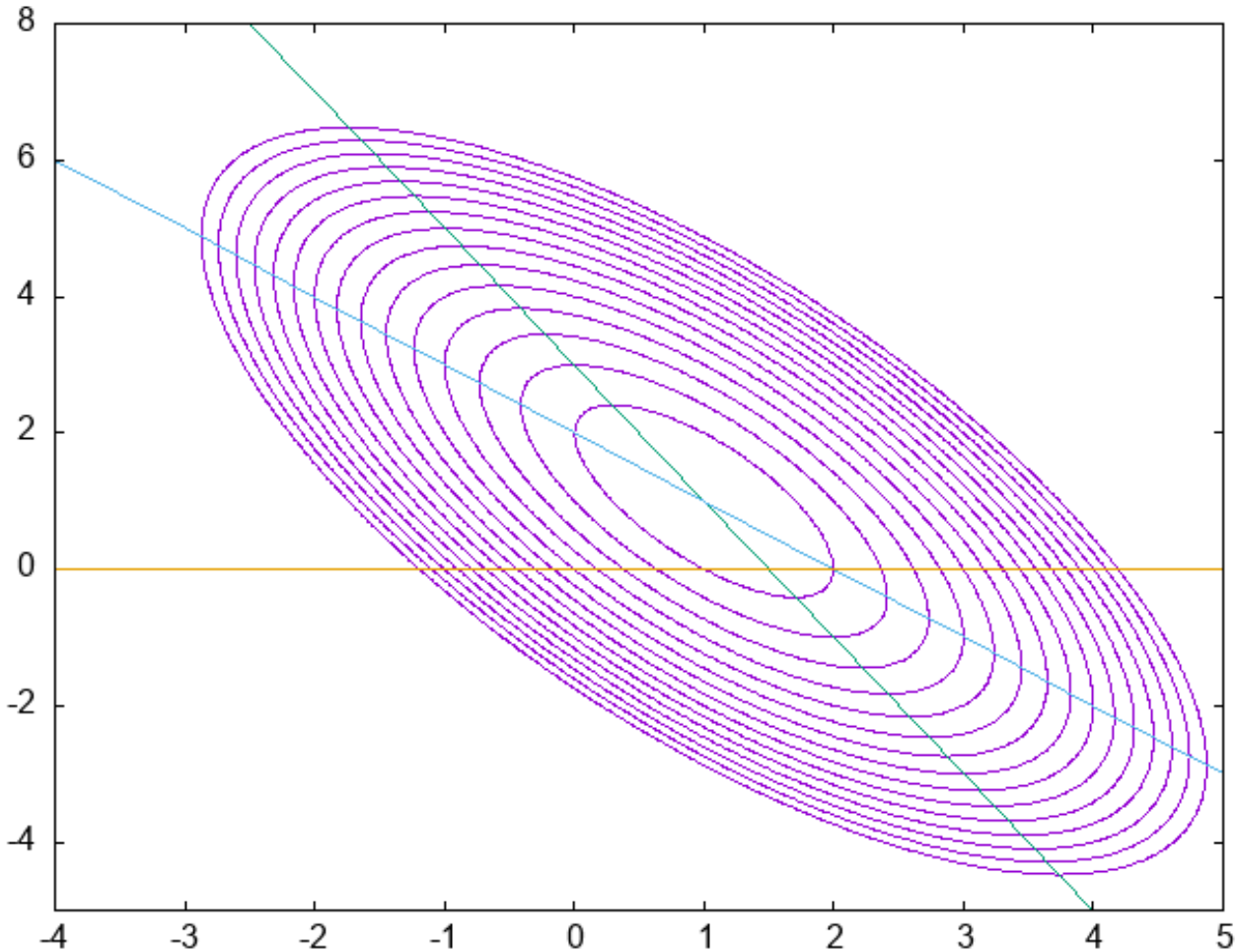
As illustration consider linear equations in two variables

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

The explicit expression for the quadratic form is

$$f(x_1, x_2) = \frac{1}{2}\left(2x_1^2 + 2x_1x_2 + x_2^2\right) - 3x_1 - 2x_2$$

It traces a paraboloid and the vertex is the minimum of the paraboloid at $(1, 1)$ which is also solution of the equations above. This method gives potential solutions through iterative steps by steepest descent process.



Steepest descent

The quadratic form decreases most rapidly in the direction of steepest descent. This direction is given by $-\nabla f(\mathbf{x})$,

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

since $\mathbf{A}$ is symmetric and the following properties are used,

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

The $\mathbf{r}$ is called residual vector

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

that measure how far the estimated value of $\mathbf{A}\mathbf{x}_n$, and hence the solution, is away from $\mathbf{b}$. The $\mathbf{r}$ points to the direction of steepest descent and each consecutive steps are some scalar multiple of $\mathbf{r}$. If the starting guesses for solution and residue are $\mathbf{x}_0, \mathbf{r}_0$, then the first step is

$$\mathbf{x}_1 = \mathbf{x}_0 + \alpha_0\mathbf{r}_0$$

where α_0 is the scaling factor that determines how we descend in the steepest direction.

For successive steps, the solution goes as

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{r}_i$$

α is choosen to minimize $f(\mathbf{x})$. For the first step taken $-\nabla f(\mathbf{x}_i) = -\mathbf{r}_i$ and hence,

$$\frac{d}{d\alpha} f(\mathbf{x}_1) = 0 \quad \rightarrow \quad [\nabla f(\mathbf{x}_1)]^T \frac{d}{d\alpha} \mathbf{x}_1 = 0$$

$$\Rightarrow \quad \mathbf{r}_1^T \mathbf{r}_0 = 0 \quad \rightarrow \quad (\mathbf{b} - \mathbf{A}\mathbf{x}_1)^T \mathbf{r}_0 = 0$$

$$\Rightarrow \quad \Big(\mathbf{b} - \mathbf{A}(\mathbf{x}_0 + \alpha \mathbf{r}_0)\Big)^T \mathbf{r}_0 = 0$$

$$\Rightarrow \quad (\mathbf{b} - \mathbf{A}\mathbf{x}_0)^T \mathbf{r}_0 = \alpha (\mathbf{A}\mathbf{r}_0)^T \mathbf{r}_0 \quad \rightarrow \quad \mathbf{r}_0^T \mathbf{r}_0 = \alpha \mathbf{r}_0^T (\mathbf{A}\mathbf{r}_0)$$

$$\Rightarrow \quad \alpha = \frac{\mathbf{r}_0^T \mathbf{r}_0}{\mathbf{r}_0^T (\mathbf{A}\mathbf{r}_0)} = \frac{\langle \mathbf{r}_0, \mathbf{r}_0 \rangle}{\langle \mathbf{r}_0, \mathbf{A}\mathbf{r}_0 \rangle}$$

$$\text{for } i^{\text{th}} \text{ step } \alpha_i = \frac{\langle \mathbf{r}_i, \mathbf{r}_i \rangle}{\langle \mathbf{A}\mathbf{r}_i, \mathbf{r}_i \rangle}$$

The above steps involve two matrix-vector products, namely $\mathbf{A}\mathbf{x}_n$ and $\mathbf{A}\mathbf{r}_n$. Of these, $\mathbf{A}\mathbf{x}_n$ can be eliminated by the following process,

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{r}_i$$
$$-\mathbf{A}\,\mathbf{x}_{i+1} = -\mathbf{A}\mathbf{x}_i - \alpha_i \mathbf{A}\mathbf{r}_i$$
$$-\mathbf{A}\,\mathbf{x}_{i+1} + \mathbf{b} = -\mathbf{A}\mathbf{x}_i + \mathbf{b} - \alpha_i \mathbf{A}\mathbf{r}_i$$
$$\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i \mathbf{A}\mathbf{r}_i$$

Thus we need to compute only $\mathbf{A}\mathbf{r}_i$ except the first computation of $\mathbf{r}_0$. Although this decreases computational cost, it has two problems,

1. It creases the error and new $\mathbf{x}_i$ may have to be computed at regular intervals to reduce the error.
2. It converges slowly as algorithm always follows the negative gradient and thereby the direction of steepest descent is mostly in the previously searched directions.

To travel in directions that are never taken in the past, one chooses a set of nonzero vectors $\{\mathbf{d}_0, \mathbf{d}_1, xi \ldots, \mathbf{d}_{n-1}\}$, called conjugate directions

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

This is known as the A-orthogonality condition. This amounts in re-defining the scaling coefficients as

$$\alpha_i = \frac{\langle \mathbf{d}_i, \mathbf{r}_i \rangle}{\langle \mathbf{d}_i, \mathbf{A}\mathbf{d}_i \rangle}$$

In terms of $\mathbf{d}$'s thus defined, the iterative steps are re-writen as

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{d}_i$$
$$\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i \mathbf{A}\mathbf{d}_i$$

Conjugate Gradient is a Conjugate Direction method that selects mutually orthogonal residual vectors $\mathbf{r}_i$, with $\langle \mathbf{r}_i, \mathbf{r}_j \rangle = \delta_{ij}$.

The algorithm starts with the initial guess $\mathbf{x}_0$ and corresponding conjugate direction and residual, $\mathbf{d}_0 = \mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$. The subsequent search directions are computed using the residual from the previous iteration,

$$\mathbf{d}_{i+1} = \mathbf{r}_i + \beta_i \mathbf{d}_i$$

For the A-orthogonal condition to satisfy β_i is so chosen such that

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

The α_i, β_i are given by

$$\alpha_i = \frac{\langle \mathbf{d}_i, \mathbf{A}\mathbf{r}_i \rangle}{\langle \mathbf{d}_i, \mathbf{A}\mathbf{d}_i \rangle} = \frac{\langle \mathbf{r}_{i-1}, \mathbf{r}_{i-1} \rangle}{\langle \mathbf{d}_i, \mathbf{A}\mathbf{d}_i \rangle}$$
$$\beta_i = -\frac{\langle \mathbf{d}_i, \mathbf{A}\mathbf{r}_i \rangle}{\langle \mathbf{d}_i, \mathbf{A}\mathbf{d}_i \rangle} = \frac{\langle \mathbf{r}_i, \mathbf{r}_i \rangle}{\langle \mathbf{r}_{i-1}, \mathbf{r}_{i-1} \rangle}$$

The pseudo-code for conjugate gradient algorithm is

1. Initial guess $\mathbf{x}_0$ and compute

$$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0 = \mathbf{d}_0$$

2. Iterate the following till stopping critera is met

$$\alpha_i = \frac{\langle \mathbf{r}_{i-1}, \mathbf{r}_{i-1} \rangle}{\langle \mathbf{d}_i, \mathbf{A}\mathbf{d}_i \rangle}$$
$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{d}_i$$
$$\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i \mathbf{A}\mathbf{d}_i$$

If $||\mathbf{r}_i|| < \epsilon$ stop

3. Else continue wtih

$$\beta_i = \frac{\langle \mathbf{r}_i, \mathbf{r}_i \rangle}{\langle \mathbf{r}_{i-1}, \mathbf{r}_{i-1} \rangle}$$
$$\mathbf{d}_{i+1} = \mathbf{r}_i + \beta_i \mathbf{d}_i$$
$$i = i + 1$$

The convergence is guaranteed unless round-off error plays spoilsport. To ensure speedy convergence, preconditioning, like even-odd or incomplete Cholesky is recommended.