Linear Solvers

1. Introduction [1]

There are many problems in power system analysis that require the solution of the equation Ax=b and therefore simply requires a linear solver to obtain the answer. Some of these problems include power flow, state estimation, and time-domain simulation when performed using implicit integration.

Researchers have put much effort into writing computational libraries for performing solution of linear equations. The people who write these libraries know much more about solving linear equations than you do. When developing code where you will need to solve linear equations, there are three rules:

➔**Never** invert the matrix;
➔**Always** use the libraries;
➔**Never** write your own method.

There are a number of standard, portable solver libraries available, including:

- BLAS (Basic linear algebra subprograms): Many linear algebra packages including LAPACK, ScaLAPACK and PETSc, are built on top of BLAS. Most supercomputer vendors have versions of BLAS that are highly tuned for their platforms.

- ATLAS (Automatically Tuned Linear Algebra Software package) is a version of BLAS that, upon installation, tests and times a variety of approaches to each routine and selects the version that runs fastest. ATLAS is substantially faster than the generic version of BLAS.

- LAPACK (Linear Algebra PACKage) solves dense or special case sparse systems of equations depending on matrix properties such as:
    - Precision: single, double
    - Data type: real, complex
    - Shape: diagonal, bidiagonal, tridiagonal, banded, triangular, trapezoidal, Hesenberg, general dense
    - Properties: orthogonal, positive definite, Hermetian (complex), symmetric, general.

  LAPACK is built on top of BLAS, which means it can benefit from ATLAS. LAPACK is a library that you can download for free from www.netlib.org.
- ScaLAPACK is the distributed parallel (MPI) version of LAPACK. It contains only a subset of the LAPACK routines. ScaLAPACK is also available from www.netlib.org.

- PETSc (Portable, Extensible Toolkit for Scientific Computation) is a solver library for sparse matrices that uses distributed parallelism (MPI). It is designed for general sparse matrices with no special properties, but it also works well for sparse matrices with simple properties like banding & symmetry. It has a simpler Application Programming Interface than ScaLAPACK.

When choosing a solver, pick a version that’s tuned for the platform you’re running on, and use the information that you have about your system to select the one that will be most efficient. You will have to do some research and discuss with people to gain a level of knowledge to enable you to most effectively make this selection. The following four can be interfaced with practically any language:

1. UMFPACK
2. KLU
3. MUMPS (serial version)
4. SuperLU

In these notes, we will introduce some very basic ideas regarding sparse solvers. The problem of interest is to solve for the $n\times 1$ vector $\underline{x}$ in

$$\underline{A}\underline{x} = \underline{b} \tag{1}$$

when n is very large, but without inverting the $n\times n$ matrix $\underline{A}$.

An effective method to do this is LU decomposition, or “factorization.” It is actually a very widely known and used method in many different disciplines.

If you have taken a linear algebra course (from Math), this should be familiar to you. If you have not taken such a course, you should.

2. Forward-backward substitution

Consider that you are able to obtain $\underline{A}$ as the product of two special matrices, i.e.,

$$\underline{\boldsymbol{A}} = \underline{\boldsymbol{L}}\,\underline{\boldsymbol{U}} \tag{2}$$

that satisfy the following:

- Both $\underline{L}$ and $\underline{U}$ are square matrices of the same dimension as $\underline{A}$.
- $\underline{U}$ is an upper-triangular matrix, meaning that all elements below the diagonal are 0.
- $\underline{L}$ is a lower-triangular matrix, meaning that all elements above the diagonal are 0.
- All diagonal elements of $\underline{U}$ are 1.

So, for a 3×3 case, we would have:

$$\underline{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$= \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{bmatrix} = \underline{L}\underline{U} \quad (3)$$

For the moment, let's not worry about how to obtain $\underline{L}$ and $\underline{U}$. Rather, let's think about what we can do if we get them.

What we know at this point is that $\underline{A} = \underline{L}\underline{U}$, and $\underline{A}\underline{x} = \underline{b}$. Therefore,

$$\underline{L}\underline{U}\underline{x} = \underline{b} \qquad (4)$$

Define:

$$\underline{w} = \underline{U}\underline{x} \qquad (5)$$

Substitution of (5) into (4) results in

$$\underline{L}\underline{w} = \underline{b} \qquad (6)$$

The situation is the following. We want to find $\underline{x}$. It appears that (5) and (6) are not very helpful, because solving them for $\underline{x}$ and $\underline{w}$, respectively, will require an inverse. But let's take a closer look at eqs. (5) and (6), in terms of the fully expressed matrix relations and see if we can get x and w without matrix inversion. If so, then our procedure will be to use (6) to find w and then (5) to find x.

$$\underline{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \underline{U}\,\underline{x} \tag{7}$$

$$\underline{L}\underline{w} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \underline{b} \tag{8}$$

Observe that

- Equation (8) can be solved for w without inverting L and
- Equation (7) could then be solved for x without inverting U.

Let's start with (8). We see that we can begin with the row 1 equation and proceed as follows:

$$l_{11}w_1 = b_1 \Rightarrow w_1 = b_1 / l_{11} \tag{9}$$

$$l_{21}w_1 + l_{22}w_2 = b_2 \Rightarrow w_2 = \frac{b_2 - l_{21}w_1}{l_{22}} \tag{10}$$

$$l_{31}w_1 + l_{32}w_2 + l_{33}w_3 = b_3 \Rightarrow w_3 = \frac{b_3 - l_{31}w_1 - l_{32}w_2}{l_{33}} \tag{11}$$

This procedure is called **forward substitution**. Consideration of the pattern of calculation introduced by eqs. (9)-(11) suggests a generalized formula for forward substitution, useful for computer programming, as follows:

$$w_k = \frac{b_k - \sum_{j=1}^{k-1} l_{kj}w_j}{l_{kk}} \tag{12}$$

Now that we have w, we can use (7) to find x. We see that we can begin with the row 3 equation and proceed as follows:

$$x_3 = w_3 \tag{13}$$

$$x_2 + u_{23}x_3 = w_2 \Rightarrow x_2 = w_2 - u_{23}x_3 \tag{14}$$

$$x_1 + u_{12}x_2 + u_{13}x_3 = w_1 \Rightarrow x_1 = w_1 - u_{12}x_2 - u_{13}x_3 \tag{15}$$

This procedure is called **backward substitution**. Consideration of the pattern of calculation introduced by eqs. (13)-(15) suggests a generalized formula for forward substitution, useful for computer programming, as follows:

$$x_k = w_k - \sum_{j=k+1}^{n} u_{kj} x_j \qquad (16)$$

where *n* is the dimension of the matrix.

3. Factorization using Crout algorithm

So we see that if we have L̲ and U̲, we can solve A̲ x̲=b̲ for x̲. So natural question at this point is: How to find L̲ and U̲?

The method of finding L̲ and U̲ from A̲ is called the LU factorization of A̲, otherwise known as the LU-decomposition of A̲.

To motivate it, let's first look back at eq. (3).

$$\underline{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$= \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{bmatrix} = \underline{L}\underline{U} \qquad (3)$$

Based on eq. (3), we see that the following sequence of calculations may be performed.

- Row 1 of A̲ and L̲, across columns of U̲.

$$a_{11} = l_{11}$$

$$a_{12} = l_{11}u_{12} \Rightarrow u_{12} = a_{12} / l_{11}$$

$$a_{13} = l_{11}u_{13} \Rightarrow u_{13} = a_{13} / l_{11}$$

- Row 2 of <u>A</u> and <u>L</u>, across columns of <u>U</u>.

$$a_{21} = l_{21}$$

$$a_{22} = l_{21}u_{12} + l_{22} \Rightarrow l_{22} = a_{22} - l_{21}u_{12}$$

$$a_{23} = l_{21}u_{13} + l_{22}u_{23} \Rightarrow u_{23} = \frac{a_{23} - l_{21}u_{13}}{l_{22}}$$

- Row 3 of <u>A</u> and <u>L</u>, across columns of <u>U</u>:

$$a_{31} = l_{31}$$

$$a_{32} = l_{31}u_{12} + l_{32} \Rightarrow l_{32} = a_{32} - l_{31}u_{12}$$

$$a_{33} = l_{31}u_{13} + l_{32}u_{23} + l_{33} \Rightarrow l_{33} = a_{33} - l_{31}u_{13} - l_{32}u_{23}$$

The above is convincing evidence that we will be able to perform the desired factorization. Although we have done so for only a 3×3 case, the procedure would also work for a matrix of any dimension.
If you study closely the pattern of calculation, you can convince yourself that the following generalized formula can be used in computer programming [2].

$$l_{ij} = a_{ij} - \sum_{s=1}^{j-1} l_{is}u_{sj} \qquad (17)$$

$$\boldsymbol{u_{ij} = \frac{a_{ij} - \sum_{s=1}^{i-1} l_{is} u_{sj}}{l_{ii}}} \qquad (18)$$

Use of eqs. (17,18) comprise what is known as the Crout algorithm.

4. Method of Dolittle

There are some variations on this, for example factorization where the L matrix diagonal elements are 1's and the U matrix diagonal matrix elements are nonunity, as indicated in eq. (3a).

$$\underline{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} = \underline{L}\underline{U} \quad (3a)$$

We can show how to obtain L and U in a fashion similar to that of Section 3.0.

- Row 1 of A and L, across columns of U.

$$a_{11} = u_{11}$$

$$a_{12} = u_{12}$$

$$a_{13} = u_{13}$$

- Row 2 of <u>A</u> and <u>L</u>, across columns of <u>U</u>.

$$a_{21} = l_{21}u_{11} \Rightarrow l_{21} = \frac{a_{21}}{u_{11}}$$

$$a_{22} = l_{21}u_{12} + u_{22} \Rightarrow u_{22} = a_{22} - l_{21}u_{12}$$

$$a_{23} = l_{21}u_{13} + u_{23} \Rightarrow u_{23} = a_{23} - l_{21}u_{13}$$

- Row 3 of <u>A</u> and <u>L</u>, across columns of <u>U</u>:

$$a_{31} = l_{31}u_{11} \Rightarrow l_{31} = \frac{a_{31}}{u_{11}}$$

$$a_{32} = l_{31}u_{12} + l_{32}u_{22} \Rightarrow l_{32} = \frac{a_{32} - l_{31}u_{12}}{u_{22}}$$

$$a_{33} = l_{31}u_{13} + l_{32}u_{23} + u_{33} \Rightarrow u_{33} = a_{33} - l_{31}u_{13} - l_{32}u_{23}$$

5. Factorization with Gaussian elimination

<u>Trick</u>: Augment A matrix before you begin the below algorithm by adding the vector b as the n+1 column. Then, when you finish the algorithm, you will have the vector w in the n+1 column.

The algorithm is as follows:

1. Perform Gaussian elimination on <u>A</u>. Let i=1. In each repetition below, row i is the *pivot row* and a_{ii} is the *pivot*.
   a. $L_{ji}=a_{ji}$ for j=i,…,n.

b. Divide row i by a_{ii}.

c. If [i=n, go to 2] else [go to d].

d. Eliminate all a_{ji}, j=i+1,…,n. This means to make all elements directly beneath the pivot equal to 0 by adding an appropriate multiple of the pivot row to each row beneath the pivot.

e. i=i+1, go to a.

2. The matrix U is what remains.

Example: Use LU decomposition to solve for x in the below system of equations.

$$\begin{aligned}
3x_1 + 3x_2 + 6x_3 + 9x_4 &= 1 \\
x_1 + 3x_2 + 4x_3 + x_4 &= 3 \\
x_1 + 2x_2 + 5x_3 + 6x_4 &= 2 \\
x_1 \qquad\quad - x_3 + 3x_4 &= 1
\end{aligned}$$

In matrix form, the above is:

$$\begin{bmatrix} 3 & 3 & 6 & 9 \\ 1 & 3 & 4 & 1 \\ 1 & 2 & 5 & 6 \\ 1 & 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 2 \\ 1 \end{bmatrix}$$

Form the augmented matrix.

$$\begin{bmatrix} 3 & 3 & 6 & 9 & 1 \\ 1 & 3 & 4 & 1 & 3 \\ 1 & 2 & 5 & 6 & 2 \\ 1 & 0 & -1 & 3 & 1 \end{bmatrix}$$

Now perform the algorithm.

$$\begin{bmatrix} 3 & 3 & 6 & 9 & 1 \\ 1 & 3 & 4 & 1 & 3 \\ 1 & 2 & 5 & 6 & 2 \\ 1 & 0 & -1 & 3 & 1 \end{bmatrix}; \mathrm{L}= \begin{bmatrix} 3 & 0 & 0 & 0 \\ 1 & \_ & 0 & 0 \\ 1 & \_ & \_ & 0 \\ 1 & \_ & \_ & \_ \end{bmatrix}$$

Divide first row by 3 and then add multiples of it to remaining rows so that first element in remaining rows gets zeroed.

$$\begin{bmatrix} 1 & 1 & 2 & 3 & 1/3 \\ 0 & 2 & 2 & -2 & 8/3 \\ 0 & 1 & 3 & 3 & 5/3 \\ 0 & -1 & -3 & 0 & 2/3 \end{bmatrix}; \mathrm{L}= \begin{bmatrix} 3 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 1 & \_ & 0 \\ 1 & -1 & \_ & \_ \end{bmatrix}$$

Divide second row by 2; then add multiples of it to remaining rows so that second element in remaining rows gets zeroed.

$$\begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{1/3} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{-1} & \mathbf{4/3} \\ \mathbf{0} & \mathbf{0} & \mathbf{2} & \mathbf{4} & \mathbf{1/3} \\ \mathbf{0} & \mathbf{0} & \mathbf{-2} & \mathbf{-1} & \mathbf{6/3} \end{bmatrix}; \quad \mathrm{L}= \begin{bmatrix} \mathbf{3} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{2} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{1} & \mathbf{2} & \mathbf{0} \\ \mathbf{1} & \mathbf{-1} & \mathbf{-2} & \_ \end{bmatrix}$$

Divide third row by 2 and then add multiples of it to remaining rows so that third element in remaining rows gets zeroed.

$$\begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{1/3} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{-1} & \mathbf{4/3} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{1/6} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{3} & \mathbf{7/3} \end{bmatrix}; \quad \mathrm{L}= \begin{bmatrix} \mathbf{3} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{2} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{1} & \mathbf{2} & \mathbf{0} \\ \mathbf{1} & \mathbf{-1} & \mathbf{-2} & \mathbf{3} \end{bmatrix}$$

Divide third row by 3.

$$\begin{bmatrix} 1 & 1 & 2 & 3 & 1/3 \\ 0 & 1 & 1 & -1 & 4/3 \\ 0 & 0 & 1 & 2 & 1/6 \\ 0 & 0 & 0 & 1 & 7/9 \end{bmatrix} = \left[\underline{U} \quad | \quad \underline{w}\right] \Rightarrow \underline{\boldsymbol{w}} = \underline{\boldsymbol{U}}\,\underline{\boldsymbol{x}}$$

Use backwards substitution to get $\underline{x}$. This method eliminates forward substitution step.

---

[1] 2007 presentation slides from Paul Gray, University of Northern Iowa, Henry Neeman, University of Oklahoma, Charlie Peck, Earlham College.
[2] Vlach and Singhal, "Computer Methods for Circuit Analysis and Design," 2nd edition, 1994.