Numerical Solution of Systems of Linear Equations and Matrix Factorization

In this topic, we will solve systems of lineat equations of the form:

$$A\vec{x} = \vec{b}$$

for a real, non-singular matrix A and real vectors $\vec{x}$ and $\vec{b}$.

Crammer's Rule

The simplest method for computing the solution to small systems of linear equations is *Crammer's Rule*:

> **Crammer's Rule** gives the solution of the system as:
>
> $$x_j = \frac{\Delta_j}{|A|}$$
>
> where Δ_j is the determinant of the matrix A but whose j-column has been substituted by the vector $\vec{b}$.

However, *Crammer's Rule* is very slow, so not very useful for systems of more than a few equations.

Direct and Indirect Methods

There exist other alternative methods to *Crammer's Rule* that are **much** more efficient. They are usually divided in two families: **direct** and **iterative**. While direct methods will give you the solution in a finite number of operations, iterative will require, theoretically, an infinite number of iterations to get it.

Condition Number of a Matrix

> The **Condition Number** of a Matrix is given by the formula:
>
> $$K_p(A) = ||A||_p ||A^{-1}||_p$$
>
> where p defines the norm that is used.

The condition number has several properties:

- $K_p(A) \geq 1$

$$K_p(A) = ||A||_p ||A^{-1}||_p \geq ||AA^{-1}||_p = 1$$

- $K_p(A) = K_p(A^{-1})$

- $K_p(A) = K_p(\alpha A),\ \forall \alpha \in \mathbb{C}$
- $K_2(A) = 1$ if A is orthogonal

$$||A||_2 = \sqrt{\rho(A^T A)} = \sqrt{\rho(I)} = 1$$

where $\rho(A)$ is the spectral radius of the matrix A, defined as the maximum of the absolute value of the eigenvalues of A.

- From a practical point of view, the condition number of a matrix for $p = 2$ can be comuted as the ratio:

$$K_2(A) = \frac{\sigma_{\max}}{\sigma_{\min}}$$

where $\sigma_{\max}$ and $\sigma_{\min}$ are, respectively, the largest and the smallest singular values of A. Furthermore, if A is symmetric and positive definite, the condition number can be computed as:

$$K_2(A) = \frac{\lambda_{\max}}{\lambda_{\min}}$$

where $\lambda_{\max}$ and $\lambda_{\min}$ are, respectively, the largest and the smallest eigenvalues of A.

The larger the condition number, the more difficult it is to solve the system of linear equations $A\vec{x} = \vec{b}$.

Direct Methods

Gaussian Elimination

If A is a square matrix, the **Gaussian Elimination Method** (*GEM*) aims at reducing the system of linear equations to an equivalent one (that is, one having the same solution) of the form $Ux = \tilde{b}$, where U is an upper triangular matrix and $\tilde{b}$ is an updated right hand side vector. This new system can be solved by the backward substitution method.

The process of reducing the system to $Ux = \tilde{b}$ is simple, and it can be done with linear operations between columns and rows, or even by permutations of rows (called *pivoting*).

$$\left[\begin{array}{ccc|c} \boxed{1} & 2 & 3 & 1 \\ 3 & 6 & 3 & -1 \\ 2 & 1 & 1 & 2 \end{array}\right] \xrightarrow[r_3 - 2r_1]{r_2 - 3r_1} \left[\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 0 & 0 & -6 & -4 \\ 0 & -3 & -5 & 0 \end{array}\right] \xrightarrow{r_2 \leftrightarrow r_3}$$

$$\rightarrow \left[\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 0 & -3 & -5 & 0 \\ 0 & 0 & -6 & -4 \end{array}\right]$$

And using back substitution:

$$x_3 = \frac{-4}{-6} = \frac{2}{3}$$

$$x_2 = \frac{5x_3}{-3} = -\frac{10}{9}$$

$$x_1 = \frac{1 - 3x_3 - 2x_2}{1} = \frac{11}{9}$$

Gaussian Elimination with Partial Pivoting

By looking at this example above it is clear that, as for every iteration of the back substitution we are dividing by a pivot, we need these to be as large as possible (in absolute value) in order to minimize truncation errors.

The *Gaussian Elimination method with Partial Pivoting* selects the largest pivot of a column as the pivot for the next round of reducing operations.

$$\left[\begin{array}{ccc|c} 1 & 4 & 3 & 1 \\ \boxed{3} & 6 & 3 & -1 \\ 2 & 1 & 1 & 2 \end{array}\right] \xrightarrow[r_3 - \frac{2}{3}r_2]{r_1 - \frac{1}{3}r_2} \left[\begin{array}{ccc|c} 0 & 2 & 2 & 4/3 \\ 3 & 6 & 3 & -1 \\ 0 & \boxed{-3} & -1 & 8/3 \end{array}\right] \xrightarrow{r_1 + \frac{2}{3}r_3}$$

$$\rightarrow \left[\begin{array}{ccc|c} 0 & 0 & 4/3 & 28/9 \\ 3 & 6 & 3 & -1 \\ 0 & -3 & -1 & 8/3 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 3 & 6 & 3 & -1 \\ 0 & -3 & -1 & 8/3 \\ 0 & 0 & 4/3 & 28/9 \end{array}\right]$$

And using back substitution:

$$x_3 = \frac{28/9}{4/3} = \frac{7}{3}$$

$$x_2 = \frac{8/3 + x_3}{-3} = -\frac{5}{3}$$

$$x_1 = \frac{-1 - 3x_3 - 6x_2}{3} = \frac{2}{3}$$

Gaussian Elimination with Scaled Pivoting

Scaling is another technique that improves the *GEM*. It does the same as *Gaussian Elimination with Partial Pivoting*, but instead of choosing the pivots based on absolute size, it chooses them based on relative size.

Pivots are chosen such that their ratio to the largest entry in (the right hand side of) their row is maximum. At the end, in order to do backward substitution, we need to have the matrix in echelon form, so we need to reorganize the rows. Instead of moving the elements of the matrix around in the memory, as this could be very resource-heavy on the computer, we keep track of which row the pivots for each step were in inside an array l. Then, in the end we use the indices given by l for the rest of the operations. In practice it is as if we had done all the permutations, but using much less resources.

The full process would be (for an $n \times n$ matrix A):

1. Initialize a permutation vector with the natural order of the matrix rows $l = [1, 2, 3, \ldots, n]$.
2. Store in a vector $s = [s_1, s_2, \ldots, s_n]$ the maximum coefficient of each row of A:

$$s_i = \max_{1 \leq j \leq n} |a_{ij}|$$

   for $i = 1, 2, \ldots, n$.

3. Start the forward elimination process. For each column j:
   - Compute the ratio of each element of the column with the maximum element of the respective row stored in s. Choose the pivot for which that ratio is largest. If we define a vector with all the ratios for column j:

$$r = \left(\left| \frac{a_{l_i,j}}{s_{l_i}} \right| \right)_{i=1}^{n}$$

     we would choose the pivot on row l_k and column j if the largest ratio in the array r is r_k. Then, we would swap the values of l_j and l_k, as l_k should be the j-th row of the echelon form.

     Once we have executed the forward elimination procedure on this column, we go to the next and repeat the same process.

Notice that the scale factors are not changed after each pivot step. Intuitively, one might think that after each step in the Gaussian algorithm, the coefficients of the new new, modified system should be used to recompute the scale factors instead of using the original scale vector. This *could* be done, but it is generally believed that the extra computations involved in this procedure are not worthwhile in the majority of linear systems.

We have the following system:

$$\left[\begin{array}{ccc|c} 1 & 4 & 3 & 1 \\ 3 & 6 & 3 & -1 \\ 2 & 1 & 1 & 2 \end{array}\right]$$

The first thing to do is to initialize l and compute the vector s with the maximum absolute value of each row:

$$l = [1, 2, 3]$$

$$s = [4, 6, 2]$$

We can then start the forward elimination:

1. For column 1:

$$r = \left[\frac{1}{4}, \frac{3}{6}, \frac{2}{2}\right] = [0.25, 0.5, 1]$$

The largest element is r_3, so we choose the pivot in row $l_3 = 3$ and swap l_1 and l_3:

$$l = [3, 2, 1]$$

Our matrix is now:

$$\left[\begin{array}{ccc|c} 0 & 7/2 & 5/2 & 0 \\ 0 & 9/2 & 3/2 & -4 \\ \boxed{2} & 1 & 1 & 2 \end{array}\right] \xrightarrow[\text{through } l]{\text{Indirectly,}} \left[\begin{array}{ccc|c} 2 & 1 & 1 & 2 \\ 0 & 9/2 & 3/2 & -4 \\ 0 & 7/2 & 5/2 & 0 \end{array}\right]$$

2. For column 2:

$$r = \left[\text{None}, \frac{9/2}{6}, \frac{7/2}{4}\right] = [\text{None}, 0.75, 0.875]$$

Notice that now it makes no sense to take into account r_1, as the pivot has already been used! The largest element is r_3, so we choose the pivot in row $l_3 = 1$ (which is $7/2$) and swap l_2 and l_3:

$$l = [3, 1, 2]$$

Our matrix is now:

$$\left[\begin{array}{ccc|c} 0 & \boxed{7/2} & 5/2 & 0 \\ 0 & 0 & -12/7 & -4 \\ 2 & 1 & 1 & 2 \end{array}\right] \xrightarrow[\text{through } l]{\text{Indirectly,}} \left[\begin{array}{ccc|c} 2 & 1 & 1 & 2 \\ 0 & 7/2 & 5/2 & 0 \\ 0 & 0 & -12/7 & -4 \end{array}\right]$$

And using back substitution:

$$x_3 = \frac{-4}{-12/7} = \frac{7}{3}$$

$$x_2 = \frac{-5/2 \cdot x_3}{7/2} = -\frac{5}{3}$$

$$x_1 = \frac{2 - x_3 - x_2}{2} = \frac{2}{3}$$

Iterative Methods

Iterative methods calculate a sequence of approximate solutions $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$ that converges to the real solution x of the system:

$$\lim_{n \to \infty} x^{(n)} = x$$

This process can be stopped whenever we are close enough to the solution, in contrast with the Gaussian elimination method that doesn't give any kind of solution till you finish the process.

One problem of Gaussian elimination is that it may cause fill-in of the zero entries by nonzero values due to truncation errors. Thus, it is not a good idea to use it for ***sparse problems***, which are systems whose matrix has many zero coefficients. In those cases, it is better to use iterative methods, which preserve the sparse structure of systems.

> The general form of **Iterative Algorithms** is:
>
> $$Qx^{(n+1)} = (Q - A)x^{(n)} + b \qquad n = 0, 1, 2, \ldots$$
>
> where Q is a non-singular matrix and $x^{(0)}$ is chosen by the user.

We can check that, if x^* this sequence has a limit, it is the solution of our system:

$$Qx^* = (Q - A)x^* + b = Qx^* - Ax^* + b \rightarrow Ax^* = b$$

The choice of Q is influenced by the fact that the SLE at each iteration must be easy to solve, otherwise we'd better use a direct method. Also, it should be chosen in a way that the method converges no matter what is the initial condition, and the faster it converges, the better.

Richardson Iteration

In *Richardson iteration*, we choose $Q = I$.

Gauss-Jacobi and Gauss-Seidel

These are the most popular methods, and they can both be obtained from writing the SLE as:

$$\sum_{j=1}^{N} a_{ij} x_i = b_i$$

or, in expanded form:

$$\begin{cases} a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & a_{23}x_3 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & \vdots & & \ddots & & \vdots & & \vdots \\ a_{n-1,1}x_1 & + & a_{n-1,2}x_2 & + & a_{n-1,3}x_3 & + & \cdots & + & a_{n-1,n}x_n & = & b_{n-1} \\ a_{n1}x_1 & + & a_{n2}x_2 & + & a_{n3}x_3 & + & \cdots & + & a_{nn}x_n & = & b_n \end{cases}$$

Gauss-Jacobi

The **Gauss-Jacobi** method solves each i-th equation of the system for the i-th unknown, assuming the elements of the diagonal are non-zero:

$$\begin{cases} x_1 & = & (b_1 & - & a_{12}x_2 & - & a_{13}x_3 & - & \cdots & - & a_{1n}x_n)/a_{11} \\ x_2 & = & (b_2 & - & a_{21}x_1 & - & a_{23}x_3 & - & \cdots & - & a_{2n}x_n)/a_{22} \\ \vdots & & \vdots & & \vdots & & \vdots & & \ddots & & \vdots \\ x_{n-1} & = & (b_{n-1} & - & a_{n-1,1}x_1 & - & a_{n-1,2}x_2 & - & \cdots & - & a_{n-1,n}x_n)/a_{n-1,n-1} \\ x_n & = & (b_n & - & a_{n,1}x_1 & - & a_{n,2}x_2 & - & \cdots & - & a_{n,n-1}x_{n-1})/a_{nn} \end{cases}$$

We start off with an approximation of x and iterate until we arrive at a satisfactory solution. In shorter form, this method is:

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

In this method, we use only points of the previous approximation of the solution (n) in order to compute the next approximation of the solution ($n+1$).

Gauss-Seidel

The **Gauss-Seidel** method is exactly the same as the *Gauss-Jacobi* method, but it uses the new approximations $x_i^{(n+1)}$ for $i = 1, 2, 3, \ldots, k$ in order to compute the approximation of $x_{k+1}^{(n+1)}$. Its mathematical expression is:

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

Successive Over-Relaxation

There is a way to accelerate *Gauss-Seidel* method using a technique called **Succesive Over-Relaxation** (SOR). This is done introducing a parameter $\omega \in [0, 2]$ that puts more weight in the updated components. The formula is then:

$$x_i^{(n+1)} = \omega \left(-\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{(n+1)} - \sum_{j=i+1}^{N} \frac{a_{ij}}{a_{ii}} x_j^{(n)} + \frac{b_i}{a_{ii}} \right) + (1-\omega) x_i^{(n)}$$

Convergence

If we rewrite the previous definition of the general iterative method, we obtain:

> The general iterative method is defined as:
>
> $$x^{(n+1)} = Q^{-1}((Q - A)x^{(n)} + b)$$
>
> where the real solution x^* to the system exists if A is non-singular.

To analyze the convergence of the method, we evaluate the expression $x^{(n+1)} - x^*$:

$$x^{(n+1)} - x^* = Q^{-1}((Q - A)x^{(n)} + b) - x^* =$$

$$= (I - Q^{-1}A)x^{(n)} + Q^{-1}Ax^* - x^* = (I - Q^{-1}A)(x^{(n)} - x^*)$$

Therefore, the error in the $(n + 1)$-th operation is:

$$e^{(n+1)} = (I - Q^{-1}A)e^{(n)}$$

We need $e^{(n+1)}$ to decrease as $n \to \infty$, therefore, we need the absolute value of all the eigenvalues of $(I - Q^{-1}A)$ to be smaller than 1.

There is another sufficient (but not necessary) condition for convergence of these methods that is easier to check, and it is the one used in practice:

> The following condition is sufficient for the *Gauss-Jacobi* and *Gauss-Seidel* methods to converge for any initial approximation $x^{(0)}$:
>
> $$|a_{ii}| > \sum_{j=1,\, j\neq i}^{N} |a_{ij}|$$
>
> which is equivalent to requiring the matrix A to be *diagonally dominant*.