The Jacobi Method

We decompose

\[ A = D - L - U \]

\[
\begin{align*}
Ax &= b \\
\Rightarrow (D - L - U)x &= b \\
\Rightarrow Dx &= (L + U)x + b \\
\Rightarrow x &= \frac{D^{-1}(L + U)x + D^{-1}b}{c} \\
&= (Tx + c)
\end{align*}
\]

**Iteration:** \[ x^{(k+1)} = D^{-1}(L + U)x^{(k)} + D^{-1}b \] or \[ Dx^{(k+1)} = (L + U)x^{(k)} + b \]

- **Solution:** Easy, since we need to solve a linear system of equations with diagonal coefficient matrix

\[
\begin{bmatrix}
  d_1 & d_2 & \cdots & d_n \\
  & d_1 & & \cdots \ \\
  & & \ddots & \vdots \\
  & & & d_1
\end{bmatrix}
\begin{bmatrix}
  x_1^{(k+1)} \\
  x_2^{(k+1)} \\
  \vdots \\
  x_n^{(k+1)}
\end{bmatrix}
= 
\begin{bmatrix}
  c_1 \\
  c_2 \\
  \vdots \\
  c_n
\end{bmatrix}
\Rightarrow x_i^{(k+1)} = \frac{c_i}{d_i}
\]

- **Convergence:** The Jacobi method if guaranteed to converge when \( A \) is diagonally dominant by rows.

- **Complexity:** Each iteration has a cost associated with:

1. Solving \( Dx^{(k+1)} = c \) which requires \( n \) divisions.
2. Computing \( x = (L + U)x^{(k)} + b \) which requires as many additions and multiplications as non-zero entries in \( A \) (worst case \( O(n^2) \), but could be \( O(n) \) for sparse matrices).
• **Stopping criteria:** \( \| b - Ax^{(k)} \| < \varepsilon \) or \( \| x^{(k+1)} - x^{(k)} \| < \varepsilon \).

There are three forms of this algorithm we will see, for different purposes:

1. **Matrix form (for proofs)**
   \[ Dx^{(k+1)} = (L + U)x^{(k)} + b. \]

2. **Algorithm form (without in-place update).** Each row of \( Dx^{(k+1)} = (L + U)x^{(k)} + b \) can be written as:
   \[ a_{ii}x_i^{(k+1)} = b_i - \sum_{j \neq i} a_{ij}x_j^{(k)} \]
   \[ \Rightarrow x_i^{(k+1)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j \neq i} a_{ij}x_j^{(k)} \right) \]

   ```
   1: x^{(0)} ← initial guess
   2: for k = 1 \ldots < \text{max iterations} > do
   3:     for i = 1 \ldots n do
   4:         x_i^{(k+1)} ← \frac{1}{a_{ii}} \left( b_i - \sum_{j \neq i} a_{ij}x_j^{(k)} \right)
   5:     end for
   6:     check for convergence
   7: end for
   ```

3. **In-place algorithm (replaces x with a better estimate)**

   ```
   1: x ← initial guess
   2: for k = 1 \ldots < \text{max iterations} > do
   3:     for i = 1 \ldots n do
   4:         x_i^{\text{new}} ← \frac{1}{a_{ii}} \left( b_i - \sum_{j \neq i} a_{ij}x_j^{(k)} \right)
   5:     end for
   6:     x ← x^{\text{new}}
   7:     check for convergence
   8: end for
   ```

**The Gauss-Seidel Method**

We again employ the decomposition \( A = D - L - U \)

\[ Ax = b \]
\[ \Rightarrow (D - L - U)x = b \]
\[ \Rightarrow (D - L)x = Ux + b \]
At this point, we place $x^{(k+1)}$ on the left hand side and $x^{(k)}$ on the right hand side

$$\quad (D - L)x^{(k+1)} = Ux^{(k)} + b \tag{1}$$

The benefit of the Gauss-Seidel method (1) over Jacobi is the improved convergence, which is guaranteed not only for diagonally dominant matrices, but also for symmetric and positive definite matrices.

In terms of complexity, each iteration of (1) amounts to solving a lower triangular system via forward substitution, i.e., incurs a cost $O(k)$, where $k$ is the number of non-zero entries in $A$. Once again, form (1) is useful for proofs, while the pseudo code version is given as:

- Without in-place update

```plaintext
1: $x^{(0)} \leftarrow$ initial guess
2: for $k = 1 \ldots < \max\text{ iterations} >$ do
3:     for $i = 1 \ldots n$ do
4:         $x^{(k+1)}_i \leftarrow \frac{1}{a_{ii}} \left( b_i - \sum_{j<i} a_{ij}x^{(k+1)}_j - \sum_{j>i} a_{ij}x^{(k)}_j \right)$
5:     end for
6:     check for convergence
7: end for
```

- In-place update

```plaintext
1: $x \leftarrow$ initial guess
2: for $k = 1 \ldots < \max\text{ iterations} >$ do
3:     for $i = 1 \ldots n$ do
4:         $x^{\text{new}}_i \leftarrow \frac{1}{a_{ii}} \left( b_i - \sum_{j<i} a_{ij}x^{\text{new}}_j - \sum_{j>i} a_{ij}x_j \right)$
5:     end for
6:     $x \leftarrow x^{\text{new}}$
7:     check for convergence
8: end for
```

Compare the above in-place update with that for Jacobi
1: \( x \leftarrow \) initial guess
2: \( \text{for } k = 1 \ldots < \text{max iterations} \) \( \text{do} \)
3: \( \text{for } i = 1 \ldots n \) \( \text{do} \)
4: \[
x^{\text{new}}_i \leftarrow \frac{1}{a_{ii}} \left( b_i - \sum_{j \neq i} a_{ij} x^{(k)}_j \right)
\]
5: \( \text{end for} \)
6: \( x \leftarrow x^{\text{new}} \)
7: check for convergence
8: \( \text{end for} \)

The real difference in performance is that Gauss-Seidel is generally serial in nature (although parallel variants exist), while Jacobi is highly parallel.

Overdetermined systems

So far, we considered linear systems \( Ax = b \) with the same number of equations and unknowns (i.e., \( A \in \mathbb{R}^{n \times n} \)). In the case where \( A \in \mathbb{R}^{m \times n} \), with \( m > n \) (more equations than unknowns) the existence of a true solution is not guaranteed, in this case we look for the “best possible” substitute for a solution. Before analyzing what that means, let’s look at how such problems arise.

As an example, in an experiment, we measure the pressure of a gas in a closed container, as a function of the temperature. From physics,

\[
pV = nR \frac{5}{9} (T + 459.67)
\]

\[\Rightarrow p = \alpha T + \beta, \quad \alpha = \frac{5nR}{9V}, \beta = \frac{5nR \cdot 459.67}{9V}\]

What are \( \alpha \) and \( \beta \)? Experimentally, the measurements should ideally lie on a straight line \( y = c_1 x + c_0 \), but do not, due to measurement error: if we have \( n \) measurement pairs \((x_1, y_1), \ldots, (x_n, y_n)\) we would have wanted:

\[
\begin{align*}
y_1 &= c_1 x_1 + c_0 \\
y_2 &= c_1 x_2 + c_0 \\
\vdots \\
y_n &= c_1 x_n + c_0
\end{align*}
\]

\[
\begin{bmatrix}
x_1 & 1 \\
x_2 & 1 \\
\vdots & \vdots \\
x_n & 1
\end{bmatrix}
\begin{bmatrix}
c_1 \\
c_0
\end{bmatrix}
= \begin{bmatrix}
y_1 \\
y_2 \\
\vdots \\
y_n
\end{bmatrix}
\]

Here, \( A_{n \times 2} x_{2 \times 1} = b_{n \times 1} \) is a rectangular system. We cannot hope to find a true solution to this system. Instead, lets try to find an “approximate” solution, such that \( Ax \approx b \). Let’s look at the residual of this “interpolation”. The residual of the approximation of each data point is:

\[
r_i = y_i - f(x_i) = y_i - c_1 x_i - c_0
\]
If we write the vector of all residuals:

\[
\begin{bmatrix}
  r_1 \\
  r_2 \\
  \vdots \\
  r_n
\end{bmatrix} =
\begin{bmatrix}
  y_1 - c_1 x_1 - c_0 \\
  y_2 - c_1 x_2 - c_0 \\
  \vdots \\
  y_n - c_1 x_n - c_0
\end{bmatrix} =
\begin{bmatrix}
  y_1 \\
  y_2 \\
  \vdots \\
  y_n
\end{bmatrix} -
\begin{bmatrix}
  x_1 \\ 1 \\
  x_2 \\ 1 \\
  \vdots \\
  x_n \\ 1
\end{bmatrix}
\begin{bmatrix}
  c_1 \\
  0
\end{bmatrix} = b - Ax
\]

Although we can’t find an \( x \) such that \( Ax = b \) (thus, \( r = 0 \)), we can at least try to make \( r \) small.

As another example, consider the problem of finding the best parabola \( f(x) = c_2 x^2 + c_1 x + c_0 \) that fits measurements \( (x_1, y_1), \ldots, (x_n, y_n) \). We would like

\[
\begin{aligned}
f(x_1) &\approx y_1 \\
f(x_2) &\approx y_2 \\
\vdots \\
f(x_n) &\approx y_n
\end{aligned}
\]

\[
\begin{aligned}
f(x_1) &\approx y_1 \\
f(x_2) &\approx y_2 \\
\vdots \\
f(x_n) &\approx y_n
\end{aligned}
\]

\[
\Rightarrow \begin{bmatrix}
  x_1^2 \\
  x_2^2 \\
  \vdots \\
  x_n^2
\end{bmatrix} A =
\begin{bmatrix}
  y_1 \\
  y_2 \\
  \vdots \\
  y_n
\end{bmatrix}
\]

Once again, we would like to make \( r = b - Ax \) as small as possible.

How do we quantify \( r \) being small? \( \Rightarrow \) using a norm! We could ask that \( ||r||_1, ||r||_2 \) or \( ||r||_\infty \) be as small as possible. Any of these norms would be intuitive to consider for minimization (especially 1- and \( \infty \)-norms are very intuitive). However, we typically use the 2-norm for this purpose, because it’s the easiest to work with in this problem!

**Definition:** The **least squares solution** of the overdetermined system \( Ax \approx b \) is the vector \( x \) that minimizes \( ||r||_2 = ||b - Ax||_2 \).

Define \( Q(x) = Q(x_1, x_2, \ldots, x_n) = ||b - Ax||_2^2 \) where \( x = (x_1, \ldots, x_n) \) and \( A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m \ (m > n) \). The least squares solution is the set of values \( x_1, \ldots, x_n \) that minimize \( Q(x_1, x_2, \ldots, x_n) \)!

\[
Q(x_1, \ldots, x_n) = ||b - Ax||_2^2 = ||r||_2^2 = \sum_{i=1}^{m} r_i^2
\]

\[
r = b - Ax \Rightarrow r_i = b_i - (Ax)_i \Rightarrow r_i = b_i - \sum_{j=1}^{n} a_{ij} x_j
\]

\[
\Rightarrow Q(x_1, \ldots, x_n) = \sum_{i=1}^{m} \left( b_i - \sum_{j=1}^{n} a_{ij} x_j \right)^2
\]

If \( x_1, \ldots, x_n \) are those that minimize \( Q \), then:
\[
\frac{\partial Q}{\partial x_1} = 0, \frac{\partial Q}{\partial x_2} = 0, \ldots, \frac{\partial Q}{\partial x_n} = 0
\]
in order to guarantee a minimum.

\[
\frac{\partial Q}{\partial x_k} = \frac{\partial}{\partial x_k} \left( \sum_{i=1}^{m} \left( b_i - \sum_{j=1}^{n} a_{ij} x_j \right)^2 \right)
= \sum_{i=1}^{m} \frac{\partial}{\partial x_k} \left( b_i - \sum_{j=1}^{n} a_{ij} x_j \right)^2
= \sum_{i=1}^{m} 2 \left( b_i - \sum_{j=1}^{n} a_{ij} x_j \right) \frac{\partial}{\partial x_k} \left( b_i - \sum_{j=1}^{n} a_{ij} x_j \right)
= \sum_{i=1}^{m} -2r_i a_{ik} = -2 \sum_{i=1}^{m} [A^T]_{ki} r_i = -2 [A^T r]_k = 0
\]
\[
\Rightarrow [A^T r]_k = 0
\]
Thus,

\[
\begin{align*}
\frac{\partial Q}{\partial x_1} = 0 & \Rightarrow [A^T r]_1 = 0 \\
\frac{\partial Q}{\partial x_2} = 0 & \Rightarrow [A^T r]_2 = 0 \\
\vdots & \\
\frac{\partial Q}{\partial x_n} = 0 & \Rightarrow [A^T r]_n = 0
\end{align*}
\]
\[
\Rightarrow A^T r = 0
\]

Since \( r = b - Ax \), we have:

\[
0 = A^T r = A^T (b - Ax) = A^T b - A^T Ax \Rightarrow A^T Ax = A^T b
\]
The system above is called the normal equations system; it is a square system that has as solution the least-squares approximation of \( Ax \approx b \).