Numerical Methods for Linear Systems

Jan Mandel

October 17, 2023

Contents

Part I
Basic Iterative Methods

1 Jacobi method

Solving system of n linear equations for n unknowns.

Idea: one iteration is computes unknown i from equation i for all i at the same time - using **old** values of x

Example: The system of linear equations

$$\begin{aligned} 4x_1 - 3x_2 &= -1 \\ 2x_1 + 5x_2 &= 19 \end{aligned}$$

The Jacobi iterative method is: starting from given $x_1^{(0)}$, $x_2^{(0)}$, compute for $k = 0, 1, \ldots$

$$\begin{aligned} x_1^{(k+1)} &= \frac{1}{4}(-1 + 3x_2^{(k)}) \\ x_2^{(k+1)} &= \frac{1}{5}\left(19 - 2x_1^{(k)}\right) \end{aligned}$$

For a system of n equations:

$$\sum_{j=1}^{n} a_{1j}x_j = b_i,\ i = 1, \ldots, n$$

$$a_{ii}x_i + \sum_{j=1}^{i-1} a_{1j}x_j + \sum_{j=i+1}^{n} a_{1j}x_j = b_i,\ i = 1, \ldots, n$$

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

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

1.1 Matrix form

Write the equations above as

$$\underbrace{\begin{bmatrix} 4 & -3 \\ 2 & 5 \end{bmatrix}}_{A}\underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} -1 \\ 19 \end{bmatrix}}_{b}$$

so

$$Ax = b$$

To write the Jacobi iterative method in matrix form, write

$$A = D + L + U$$

where D is diagonal matrix, L is strictly lower triangular, and U is strictly upper triangular. Then $Ax = b$ becomes

$$\begin{aligned}(D + L + U)\, x &= b \\ Dx + Lx + Ux &= b\end{aligned}$$

and to derive a fixed point form (hence iterations), compute x from the term Dx:

$$\begin{aligned}Dx &= b - Lx + Ux \\ x &= D^{-1}\,(b - (L + U)x) \\ x^{(k+1)} &= D^{-1}\left(b - (L + U)x^{(k)}\right) \qquad (1)\end{aligned}$$

In the example here,

$$A = \begin{bmatrix} 4 & -3 \\ 2 & 5 \end{bmatrix}, \quad D = \begin{bmatrix} 4 & 0 \\ 0 & 5 \end{bmatrix}, \quad L = \begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix}, \quad U = \begin{bmatrix} 0 & -3 \\ 0 & 0 \end{bmatrix}$$

so the iterations (1) become

$$\begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \end{bmatrix} = \begin{bmatrix} 4 & 0 \\ 0 & 5 \end{bmatrix} \left(\begin{bmatrix} -1 \\ 19 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \end{bmatrix} \right)$$

which is again

$$\begin{aligned}x_1^{(k+1)} &= \frac{1}{4}(-1 + 3x_2^{(k)}) \\ x_2^{(k+1)} &= \frac{1}{5}\left(19 - 2x_1^{(k)}\right)\end{aligned}$$

2 Convergence

Rewrite equation (1)

$$x^{(k+1)} = D^{-1}\left(b - (L + U)x^{(k)}\right)$$

using $A = D + L + U$ as (add and substract $Dx^{(k)}$ iside the bracket)

$$x^{(k+1)} = D^{-1}\left(b - (D + L + U)x^{(k)} + Dx^{(k)}\right)$$

which is, using $D^{-1}Dx^{(k)} = x^{(k)}$, the same as

$$x^{(k+1)} = x^{(k)} + D^{-1}\left(b - Ax^{(k)}\right)$$

Now we realize that instead of D we could have used any other invertible matrix M (the same size as A) and get the general iterative method

$$x^{(k+1)} = x^{(k)} + M^{-1}\left(b - Ax^{(k)}\right) \qquad (2)$$

When x^* is the exact solution, i.e., $Ax^* = b$, we get

$$x^* = x^* + M^{-1}(b - Ax^*) \tag{3}$$

since $Ax^* - b = 0$, thus (3) is simply $x^* = x^*$. Now subtract (2) and (3) to get by simple algebra

$$x^{(k+1)} - x^* = x^{(k)} + M^{-1}\left(b - Ax^{(k)}\right) - x^* - M^{-1}(b - Ax^*)$$
$$x^{(k+1)} - x^* = \left(x^{(k)} - x^*\right) - M^{-1}A\left(x^{(k)} - x^*\right)$$

and, finally, the **error transformation equation**

$$x^{(k+1)} - x^* = \left(I - M^{-1}A\right)\left(x^{(k)} - x^*\right). \tag{4}$$

Suppose that $\|\cdot\|$ denotes a vector norm as well as a compatible matrix norm, so that we have the standard propety $\|Tx\| \le \|T\|\,\|x\|$. Then, from (4), we get

$$\left\|x^{(k+1)} - x^*\right\| \le \left\|I - M^{-1}A\right\| \left\|x^{(k)} - x^*\right\|$$

and, by induction over k and using the property $\|TU\| \le \|T\|\,\|U\|$ of a matrix norm,

$$\left\|x^{(k)} - x^*\right\| \le \left\|\left(I - M^{-1}A\right)^k\right\| \left\|x^{(0)} - x^*\right\| \le \left\|\left(I - M^{-1}A\right)\right\|^k \left\|x^{(0)} - x^*\right\|. \tag{5}$$

Take-home conclusions

1. If $\left\|I - M^{-1}A\right\| < 1$ then the iterations converge
2. If $\left|I - M^{-1}A\right|$ is small, the iterations converge fast
3. If $M \approx A$ (i.e., M is close to A), then $\left\|I - M^{-1}A\right\| = \left\|M^{-1}(M - A)\right\| \le \left\|M^{-1}\right\|\,\|M - A\|$
4. In the extreme case when $M = A$, we have $I - M^{-1}A = 0$ and the iterations converge in one step

3 Iterative solvers in practice

3.1 Implementation of an iterative method

The matrix M is called **preconditioner.**

Write (2) in the form

$$x^{(k+1)} = x^{(k)} + M^{-1}r^{(k)}, \quad r^{(k)} = b - Ax^{(k)}$$

Then one iteration can be written as 3 steps.

1. Compute the **residual** $r^{(k)} = b - Ax^{(k)}$
2. Solve the **preconditioning system** $M\delta^{(k)} = r^{(k)}$ for the **increment** $\delta^{(k)}$
3. **Apply** the increment: $x^{(k+1)} = x^{(k)} + \delta^{(k)}$

In application software, multiplication by A and solving the preconditioning system are usually implemented as functions. The matrices A or M are never stored explicitly. That would be just too expensive.

3.2 Design of preconditioners

Solving the preconditioning system $M\delta^{(k)} = r^{(k)}$ should be cheap.

The preconditioning should be close to the actual solution: $M^{-1}A \approx I$

Preconditioner is often constructed from a simplified or approximate version of the same problem.

For example, if A has large diagonal entries compared to the rest of the entries, D can be a good preconditioner. This is the Jacobi method. See "diagonally dominant" in the textbook. And solving a diagonal system is cheap.

4 Infinity norm

For $x \in \mathbb{R}^n$, define the vector norm, called the infinity norm, by

$$\|x\|_\infty = \max_{i=1,\dots,n} |x_i| .$$

What is the induced norm? Consider matrix $A \in \mathbb{R}^{n\times n}$, let $y = Ax$, and estimate:

$$|y_i| = \left|\sum_{j=1}^n a_{ij}x_j\right| \le \sum_{j=1}^n |a_{ij}|\,|x_j| \le \sum_{j=1}^n |a_{ij}| \max_{i=j,\dots,n} |x_j| = \sum_{j=1}^n |a_{ij}|\,\|x\|_\infty$$

$$\max_{i=1,\dots,n} |y_i| \le \max_{i=1,\dots,n} \sum_{j=1}^n |a_{ij}|\,\|x\|_\infty$$

Thus, we have

$$\|Ax\|_\infty \le \|A\|_\infty \|x\|_\infty \tag{6}$$

if we define

$$\|A\|_\infty = \max_{i=1,\dots,n} \sum_{j=1}^n |a_{ij}|$$

Exercise: show that the inequality (6) is sharp, that is, the inequality becomes equality for some $x \neq 0$, and we have in fact

$$\|A\|_\infty = \max_{x\neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty}$$

5 Convergence of Jacobi method for strictly diagonally dominant matrices

From (5), we have for the Jacobi method,

$$\left\|x^{(k)} - x^*\right\| \le \left\|\left(I - D^{-1}A\right)\right\|^k \left\|x^{(0)} - x^*\right\|$$

where D is the diagonal of A. Matrix $A = [a_{ij}]$ is called strictly diagonally dominant if for all i,

$$\sum_{\substack{j=1\\ j\neq i}}^n |a_{ij}| < |a_{ii}| . \tag{7}$$

So, suppose that $A \in \mathbb{R}^{n\times n}$ is strictly diagonally dominant, and compute $I - D^{-1}A$

$$I - D^{-1}A = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} - \begin{bmatrix} 1/a_{11} & & & \\ & 1/a_{22} & & \\ & & \ddots & \\ & & & 1/a_{nn} \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \\ a_{na} & a_{n2} & \cdots & 1/a_{nn} \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -a_{12}/a_{11} & \cdots & -a_{1n}/a_{11} \\ -a_{21}/a_{22} & 0 & \cdots & -a_{2n}/a_{22} \\ \vdots & \vdots & \ddots & \vdots \\ -a_{n1}/a_{nn} & -a_{n2}/a_{nn} & \cdots & 0 \end{bmatrix}$$

Thus

$$\left\|I - D^{-1}A\right\| = \max_{i=1,\ldots,n} \sum_{\substack{j=1 \\ j\neq i}}^{n} \left|\frac{a_{ij}}{a_{ii}}\right| = \max_{i=1,\ldots,n} \frac{\sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}|}{|a_{ii}|} < 1$$

because A is strictly diagonally dominant (7).

6 Gauss Seidel method

Solving system of n linear equations for n unknowns.

Idea: one iteration is computes unknown i from equation i one at a time, using **new** values of x as soon as they are available

Example: The system of linear equations

$$\begin{aligned} 4x_1 - 3x_2 &= -1 \\ 2x_1 + 5x_2 &= 19 \end{aligned}$$

The Gauss-Seidel iterative method is: starting from given $x_1^{(0)}$, $x_2^{(0)}$, compute for $k = 0, 1, \ldots$

$$\begin{aligned} x_1^{(k+1)} &= \frac{1}{4}(-1 + 3x_2^{(k)}) \\ x_2^{(k+1)} &= \frac{1}{5}\left(19 - 2x_1^{(k+1)}\right) \end{aligned}$$

For a system of n equations:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i,\ i = 1, \ldots, n$$

$$a_{ii}x_i + \sum_{j=1}^{i-1} a_{ij}x_j + \sum_{j=i+1}^{n} a_{ij}x_j = b_i,\ i = 1, \ldots, n$$

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

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

Write Gauss-Seidel method in a correction form - put $x_i^{(k)}$ inside the bracket and duplicater outside:

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

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

Coding is simple: for $k = 1, 2, \ldots$

$$x_i \leftarrow x_i + \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{n} a_{ij} x_j \right), \ i = 1, \ldots, n \tag{8}$$

- Simply use the latest values of x. This also saves memory for storing x, very useful on early computers.
- Also a single uninterrupted sum from 1 to n may save a bit of time - only one setup of the sum rather than two.

7 SOR

Now, instead of the correction, **make ω times the correction** in every step:

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

Equivalent writing - move $x_i^{(k)}$ back inside

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

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

Coding is simple: for $k = 1, 2, \ldots$

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

A significant improvement for $\omega > 1$. SOR can be proved to converge for $0 < \omega < 2$ (under assumptions).

SOR was a **major** improvement for matrices that come from discretizing differential equations, such as

$$\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}$$

SOR made solving such equations numerically practical!

The math why is a bit complicated. Not here. We'll see a special case later.

For more on such iterative methods, see the easy exposition [Var62]. For an exhaustive study of SOR and extensions, see [You03].

Part II
Symmetric Positive Definite Matrices

Matrix $A \in \mathbb{R}^{n \times n}$ is Symmetric Positive Definite (SPD) if $A = A^T$ and

$$\text{for all } v \in \mathbb{R}^n,\ v \neq 0 : v^T A v > 0.$$

Equivalent: all principal minors are positive. (Principal minor is the determinant of a submatrix obtained by selecting the same sets of rows and columns.) (For theory only)

8 Cholesky Decomposition

If A is SPD, there exist an upper triangular matrix R such that $A = R^T R$, called Cholesky decomposition. Then the system of linear equations

$$Ax = b$$

is equivalent to

$$R^T R x = b$$

and writing $y = Rx$, the system $Ax = b$ can be solved by two triangular solves:

$$\begin{aligned} R^T y &= b \quad \text{(forward substitution)} \\ Rx &= y \quad \text{(back susbtitution)} \end{aligned}$$

See textbook for a $2x2$ example how to find R.

9 Outer product Cholesky decomposition method

Write the given matrix A and the sougth matrix R in a block form

$$\begin{bmatrix} 1\times 1 & 1\times n-1 \\ n-1\times 1 & n-1\times n-1 \end{bmatrix}$$

$$A=\begin{bmatrix} a_{11} & b^T \\ b & C \end{bmatrix}, \quad R=\begin{bmatrix} \alpha & \beta^T \\ 0 & \Gamma \end{bmatrix}$$

where Γ is upper triangular.

If $a_{11}<0$, the matrix A is not SPD and the the algorithm fails. Otherwise, write the equations to find the blocks of R:

$$R^TR=\begin{bmatrix} \alpha & 0 \\ \beta & \Gamma^T \end{bmatrix}\begin{bmatrix} \alpha & \beta^T \\ 0 & \Gamma \end{bmatrix}=\begin{bmatrix} \alpha^2 & \alpha\beta \\ \beta\alpha & \Gamma^T\Gamma+\beta\beta^T \end{bmatrix}=\begin{bmatrix} a_{11} & b^T \\ b & C \end{bmatrix}$$

$$\alpha^2=a_{11}\Rightarrow\alpha=\sqrt{a_{11}}$$
$$\beta\alpha=b\Rightarrow\beta=b/\alpha$$
$$\Gamma^T\Gamma+\beta\beta^T=C\Rightarrow\Gamma^T\Gamma=C-\beta\beta^T$$

So, to continue we need to find Cholesky decomposition of the $n-1\times n-1$ matrix $C-\beta\beta^T$, which will require Cholesky decomposition of an $n-2\times n-2$ matrix, etc.

Since $\beta\beta^T$ is called "outer product", this method is called outer product Cholesky algorithm.

To show that this algorithm works for any SPD matrix A, we only need to show that the matrix $C-\beta\beta^T$ size $n-1\times n-1$ is also SPD. Then we can find its Cholesky decomposition, etc. In the code this is done by operating on $A\left(k:n,k:n\right)$. Initially, $k=1$, then $C-\beta\beta^T$ is stored in $A\left(k:n,k:n\right)$ with $k=2$, etc., until for $k=n$ the matrix $A\left(k:n,k:n\right)$ is just a scalar. See the python code in the textbook.

Notes:

- Cholesky decomposition is a workhorse of scientific computing. It works for any SPD matrix, and a large class practical problems lead to SPD matrices.
- In Section 10 below, we show that Cholesky decomposition of SPD always matrix exists (i.e., we never have to take square root of negative number or divide by zero). Then all digonal entries of R are positive, $r_{kk}>0$) On the other hand, if Cholesky decomposition succeeds with all diagonal entries of R positive (including the last one), we know that A is SPD, since for any $v\neq 0$,
$$v^TAv=v^TR^TRv=\|Rv\|^2>0$$
because R is regular matrix as a diagonal matrix with all nonzeros on the diagonal.
- So **Cholesky decomposition can be used to decide if a symmetric matrix is positive definite or not.**
- In practice, Cholesky decomposition is very stable and reliable method.

- There is also an inner product Cholesky algorithm, which consists of computing the entries of R from $R^T R = A$ one by one in a particular order (to use only the entries of R already computed).
- The decision between outer product and inner product versions depends on your computer architecture and matrix size - the inner product version writes R into memory only once, while the outer product version does more writing.
- Cholesky decomposition is often written with a lower triangular matrix L as $A = LL^T$.
- There are versions for decomposition of symmetric matrices as $A = R^T DR$ where D is a diagonal matrix. Then D can have zero or negative entries on the diagonal, which allows A to be symmetric but not positive definite. These methods are often called LDLT decomposition. For SPD matrices, however, the numerical stability of Cholesky decomposition, especially in the presence of rounding errors, is generally superior to LDLT decomposition.

10 Cholesky decomposition of SPD matrix exists

(Graduate only material)

We will also show by construction that Cholesky decomposion exists for any SPD matrix.

So, consider a vector v structured to match the block form of A:

$$v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

where $v_1 \in \mathbb{R}$ and $v_2 \in \mathbb{R}^{n-1}$, $v_2 \neq 0$. We need to show that

$$v_2^T (C - \beta\beta^T) v_2 = v_2^T C v_2 - v_2^T \beta\beta^T v_2 > 0. \tag{9}$$

Compute

$$v^T A v = \begin{bmatrix} v_1 & v_2^T \end{bmatrix} \begin{bmatrix} a_{11} & b^T \\ b & C \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = a_{11} v_1^2 + 2 v_1 b^T v_2 + v_2^T C v_2 > 0. \tag{10}$$

Now choose v_1 to make (10) into (9). In the Choleski decomposition algorithm, we defined

$$\beta = \frac{b}{\alpha} = \frac{b}{\sqrt{a_{11}}}.$$

so

$$v_2^T \beta\beta^T v_2 = \frac{v_2^T b b^T v_2}{a_{11}}$$

and (9) becomes

$$v_2^T C v_2 - \frac{v_2^T b b^T v_2}{a_{11}} > 0.$$

So, choose in (10)

$$v_1 = -\frac{v_2^T b}{a_{11}}$$

and we get

$$a_{11}\frac{v_2^T bbv_2}{a_{11}^2} - 2\frac{v_2^T bb^T v_2}{a_{11}} + v_2^T Cv_2 > 0$$

$$v_2^T Cv_2 - \frac{v_2^T bb^T v_2}{a_{11}} > 0.$$

Therefore:

$$v_2^T (C - \beta\beta^T) v_2 > 0.$$

11 Descent methods

Suppose A is SPD and define

$$J(x) = \frac{1}{2}x^T Ax - b^T x = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} x_i a_{ij} x_j - \sum_{i=1}^{n} b_i x_i$$

$$\frac{\partial}{\partial x_k} b_i x_i = \begin{cases} b_k \text{ if } k = i \\ 0 \text{ if } k \neq i \end{cases}$$

$$\frac{\partial}{\partial x_k} x_i a_{ij} x_j = \begin{cases} 2a_{kk} \text{ if } k = i = j \\ a_{kj} x_j \text{ if } k = i \neq j \\ x_i a_{ik} \text{ if } k = j \neq i \\ 0 \text{ if } k \neq i,\ k \neq j \end{cases}$$

Using symmetry, $a_{ij} = a_{ji}$:

$$\frac{\partial}{\partial x_k} J(x) = \frac{\partial}{\partial x_k}\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} x_i a_{ij} x_j - \frac{\partial}{\partial x_k}\sum_{i=1}^{n} b_i x_i = \sum_{j=1}^{n} a_{kj} x_j - b_k$$

So

$$\nabla J(x) = Ax - b \tag{11}$$

and

$$Ax = b \Leftrightarrow \nabla J(x) = 0$$

It can be shown that if A is SPD then J has unique minimum at the solution of $Ax = b$. (Exercise)

In each step of a descent method, given approximate solution x and direction d, we search for the minimum of $J(x + td)$, $t \in \mathbb{R}$ and replace x by $x + td$

$$J(x + td) \to \min_t$$

$$x \leftarrow x + td,$$

(That's why d is also called search direction.) Compute the step size t from the zero derivative condition at minimum:

$$\begin{aligned} J(x + td) &= \frac{1}{2}(x + td)^T A(x + td) - b^T(x + td) \\ &= \frac{1}{2}x^T Ax + \frac{1}{2}tx^T Ad + \frac{1}{2}td^T Ax + \frac{1}{2}t^2 d^T Ad - b^T x - tb^T d \\ &= \frac{1}{2}x^T Ax + td^T Ax + \frac{1}{2}t^2 d^T Ad - b^T x - tb^T d \end{aligned}$$

because $x^T Ad = \left(x^T Ad\right)^T = d^T A^T x = d^T Ax$ from $A^T = A$. Thus

$$\frac{d}{dt} J\left(x + td\right) = d^T Ax + td^T Ad - d^T b = 0$$
$$t = -\frac{d^T\left(Ax - b\right)}{d^T d} = \frac{d^T\left(b - Ax\right)}{d^T d}$$

so one step of the descent method is now written as

$$\begin{aligned} r &= b - Ax \\ x &\leftarrow x + \frac{d^T r}{d^T Ad} d \end{aligned} \tag{12}$$

11.1 Gauss-Seidel as coordinate descent

Suppose that A is SPD and choose as the descent direction one of the standard unit vectors, $d = e_i$. Then,

$$d^T r = r_i\left(b - Ax\right) = b_i - \sum_{j=1}^{n} a_{ij} x_j$$
$$d^T Ad = e_i^T Ae_i = a_{11}$$

and one step of the descent method becomes

$$x \leftarrow x + \frac{b_i - \sum_{j=1}^{n} a_{ij} x_j}{a_{ii}} e_i$$

That is,

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

which is exactly one step of the Gauss-Seidel method (8).

11.2 Steepest descent

The direction in which a function decreases fastest is its negative gradient. Thus, choosing the descent direction

$$d = -\nabla J\left(x\right) = b - Ax$$

using from (11), and noting that r above is the residual, we get the steepest descent method

$$\begin{aligned} d &= r = b - Ax \\ x &\leftarrow x + \frac{d^T r}{d^T Ad} d \end{aligned}$$

We keep the separate notation for the descent direction d and the residual r, because the more advanced conjugate gradients method in the next section consists of a one-line change in the definition of d.

References

[Var62] Richard S. Varga. *Matrix iterative analysis.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1962.

[You03] David M. Young. *Iterative solution of large linear systems.* Dover Publications, Inc., Mineola, NY, 2003. Unabridged republication of the 1971 edition [Academic Press, New York-London, MR 305568].