Lecture Notes on Numerical Methods

Jan Mandel

November 15, 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:

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

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

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

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

$$J(x) = \frac{1}{2}x^T Ax - b^T x = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} x_i a_{ij} x_j - \sum_{i=1}^{n} b_i x_i \tag{11}$$

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

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

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

$$\nabla J(x) = Ax - b \tag{12}$$

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

$$\frac{d}{dt} J\left(x + td\right) = d^T Ax - d^T b + td^T Ad = 0 \tag{13}$$
$$\text{implies } t = -\frac{d^T\left(Ax - b\right)}{d^T d} = \frac{d^T\left(b - Ax\right)}{d^T d}$$

so one step of the descent method is now written as

$$\begin{aligned} r &= b - Ax \\ x &\leftarrow x + \frac{d^T r}{d^T Ad} d \end{aligned} \tag{14}$$

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,

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

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

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

from (12), 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.

12 Conjugate gradients as direct solver

Recall that if A is SPD and $J(x) = \frac{1}{2}x^T Ax - b^T x$ then $\nabla J(x) = Ax - b$ and solving the linear system $Ax = b$ can be cast as minimization of J

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

Given a search direction d we can look for an improvement of approximate solution x in the form $x = x + td$, $t \in \mathbb{R}$:

$$\min_t J(x + td), \text{ then } x \leftarrow x + td.$$

The steepest descent method is obtained by the choice of the direction $d = -\nabla J(x) = b - Ax$. What happens if we optimize J in direction d_1 then in the direction d_2? We have computed in (13)

$$\frac{d}{dt}J(x + td_1) = d_1^T Ax - d_1^T b + td_1^T Ad_1 = 0 \tag{15}$$

$$t_1 = \frac{d_1^T (b - Ax)}{d_1^T d}$$

Note that if $d_1^T (b - Ax) = 0$ then x was already optimal with respect to increments in the direction d_1. Since optimizing twice with respect to the same direction d_1 does not change anything, $J(x + t_1 d_1)$ is already optimal to chantes in t_1, so

$$d_1^t (b - A(x + t_1 d_1)) = 0$$

Now fix this t_1 and suppose we optimize $x + t_1 d_1$ in a direction d_2 which is A-orthogonal to d_1, that is,

$$d_2^T Ad_1 = 0.$$

We find t_2 the same way as (15) with $x + t_1 d_1$ in the place of x, so

$$d_2^T (b - A(x + t_1 d_1 + t_2 d_2)) = 0.$$

Note that because of the orthogonality condition $d_2^T Ad_1 = d_1^T Ad_2 = 0$ (A is symmetric), we also have

$$d_1^T (b - A(x + t_1 d_1 + t_2 d_2)) = d_1^T (b - A(x + t_1 d_1)) = 0.$$

In general, if we have A-orthogonal vectors $d_1, d_2, \ldots, d_n$ and we continue computing t_1, t_2,... we get

$$d_k^T (b - A(x + t_1 d_1 + t_2 d_2)) = d_k^T (b - A(x + t_1 d_1 + \cdots + t_n d_n)) = 0, \quad k = 1, \ldots, n$$

Write $x^* = t_1 d_1 + \cdots + t_n d_n$, then

$$d_k^T (b - Ax^*) = 0, \quad k = 1, \ldots, n$$

thus $b - Ax^* = 0$ - we have **exact** solution.

13 Conjugate gradients method as iterative solver

Coming soon.

Please see also the textbook chapters 3.3 section "Conjugate gradient method" and 7.2 section "Conjugate gradient".

Part III
Interpolation and approximation

14 Interpolation

Please see the textbook chapters 4.1-4.3.

15 Least squares approximation

We are given values y_i at points x_i, $i = 1, \ldots, m$ and instead of interpolating, we want a single linear formula, that is, producing a straight line that goes near the data points (x_i, y_i). So we formulate the optimization problem

$$E(a,b) = \sum_{i=1}^{m} (ax_i + b - y_i)^2 \to \min_{a,b} \tag{16}$$

We write this problem in matrix form to minimize over a, b

$$\left\| \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} a + \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} b - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \right\|^2 = \left\| \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_m & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \right\|^2 = \|Az - y\|^2 \to \min_z$$

with $\|\cdot\|$ is the 2 norm,

$$\|v\| = \left(v^T v\right)^{1/2},$$

and

$$A = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_m & 1 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}, \quad z = \begin{bmatrix} a \\ b \end{bmatrix}.$$

You can easily verify that this is exactly the same as the original problem (16).

The form $\|Az - y\|^2 \to \min_z$ allows an elegant matrix-based solution. From the definition of the 2-norm,

$$\begin{aligned} \|Az - y\|^2 &= (Az - y)^T (Az - y) \\ &= z^T A^T A z - y^T A z - (Az)^T y + y^T y \end{aligned}$$

From the symmetry of the inner product $u^T v = v^T u$, it follows that

$$(Az)^T y = y^T Az = \left(A^T y\right)^T z$$

and we have

$$\begin{aligned}\|Az - y\|^2 &= z^T A^T Az - 2\left(A^T y\right)^T z + y^T y \\ &= 2\left(\frac{1}{2} z^T \left(A^T A\right) z - \left(A^T y\right)^T z\right) + y^T y \to \min_z .\end{aligned}$$

Since y is constant, this is the same as

$$J(z) = \frac{1}{2} z^T \left(A^T A\right) z - \left(A^T y\right)^T z \to \min_z$$

We have minimized such functions before - substituting into (12), it follows that the solution satisfies

$$\nabla_z J(z) = \left(A^T A\right) z - A^T y = 0$$

(the gradient ∇_z has subscript z because we differentiate with respect to z). So our least squares problem (16) becomes the **normal equation**[1]

$$A^T Az = A^T y. \tag{17}$$

Note that the normal equation is obtained by multiplying the original equation $Az = y$, which cannot be satisfied in general, by A^T. The **normal matrix** $A^T A$ is always positive semidefinite, since for any vector v,

$$v^T A^T Av = (Av)^T Av = \|Av\|^2 \geq 0.$$

If A is of full column rank, then $A^T A$ is nonsingular thus positive definite and, consequently, the solution z of the normal equation (17) exists and is unique.

You can now compute $A^T y$ and use a linear solver such as Cholesky decomposition to solve the normal equation (17). However, this is generally not a good idea, as we will discuss in the next section.

Why One Should Not Use the Normal Equation

The normal equation is a classic approach to solving least squares problems, particularly in linear regression. However, it is not recommended in many situations due to numerical stability issues and inefficiency.

Given a system of linear equations $A\mathbf{x} = \mathbf{b}$, where A is an $m \times n$ matrix and $\mathbf{b}$ is an m-dimensional vector, the least squares problem seeks to find $\mathbf{x}$ that minimizes $||\mathbf{b} - A\mathbf{x}||^2$. The normal equation is derived from this problem and is given by:

$$A^T A\mathbf{x} = A^T \mathbf{b}$$

[1] Verify as an exercise that for the approximation problem (16), these are the same equations as in the book.

Issues with the Normal Equation

1. **Numerical Stability**: The main issue arises when A^TA is nearly singular or poorly conditioned, which means that the smallest change in A or $\mathbf{b}$ can lead to large changes in the solution $\mathbf{x}$. This is common in practical applications where multicollinearity is present. In such cases, the normal equation can yield inaccurate results.

2. **Computational Inefficiency**: Computing A^TA involves a matrix multiplication, which has a computational complexity of $O(n^2m)$ for an $m \times n$ matrix A. This is followed by solving the system, which typically involves $O(n^3)$ operations. This becomes computationally expensive for large matrices.

3. **Risk of Overfitting**: In cases where the number of features (columns of A) is large, using the normal equation can result in overfitting, especially if the dataset isn't large enough to support the number of features.

Alternatives to the Normal Equation

Given these issues, alternative methods like QR decomposition or Singular Value Decomposition (SVD) are preferred. These methods are more numerically stable and can handle rank-deficient or ill-conditioned matrices more robustly.

16 QR Decomposition

QR decomposition is a method used in linear algebra to decompose a matrix into two components: Q and R. Given a matrix A, which is an $m \times n$ matrix, the QR decomposition expresses A as the product of two matrices, Q and R, where:

1. Q **(Orthogonal Matrix)**:

   - Q is an $m \times m$ orthogonal matrix. This means that the columns of Q are orthogonal to each other and each column has a unit norm.
   - Orthogonality implies that $Q^TQ = QQ^T = I$, where I is the identity matrix and Q^T is the transpose of Q.
   - In the context of QR decomposition, especially when $m > n$, we often use only the first n columns of Q, which form an $m \times n$ matrix. These columns form an orthonormal basis for the column space of A.

2. R **(Upper Triangular Matrix)**:

   - R is an $m \times n$ upper triangular matrix, which means all the entries below the main diagonal are zero.
   - If $m = n$, R is a square upper triangular matrix.
   - If $m > n$, R can be thought of as being composed of an $n \times n$ upper triangular matrix on top and a zero matrix of size $(m - n) \times n$ at the bottom.

Representation of QR Decomposition

The decomposition can be represented as:

$$A = QR$$

Properties

- **Preservation of Orthogonality**: Since Q is orthogonal, the columns of A are transformed into a set of orthogonal vectors (the columns of Q) when multiplied by Q^T.
- **Efficient for Solving Linear Systems**: QR decomposition is particularly useful in solving linear systems and least squares problems because of the orthogonal and triangular properties of Q and R, respectively.

Applications

QR decomposition is widely used in numerical methods, including solving linear systems, least squares fitting, eigenvalue estimation, and matrix inversion. The stability and efficiency of QR decomposition make it a preferred method in various numerical computations.

16.1 Solving a linear system by QR decomposition

Suppose we have a linear system $Ax = b$ with A an $m \times m$ matrix and b an m column vector.

With a QR decomposition $A = QR$, this system becomes

$$QRx = b$$

thatand multiplying by Q^T from the left, we have

$$Rx = Q^T b.$$

The matrix R is upper triangular, so one can solve for x simply by back substitution.

16.2 Comparison with Gaussian elimination

Solving $Rx = Q^T b$ for x is straightforward and numerically stable compared to the row operations in Gaussian elimination.

Since Q is orthogonal, $\left\|Q^T b\right\| = \|b\|$, and it can be shown that the condition number of R is the same as the condition number of A. Therefore the method is stable, there is no source of numerical error which was not in the matrix A already. In contrast, Gaussian elimination typically requires pivoting strategies to enhance stability, which adds to the complexity, and can significantly change the condition number, potentially amplifying errors. QR decomposition naturally handles these concerns through its orthogonalization process. QR can be interpreted as a method similar to pivoting: In Gaussian elimination, pivoting is used to improve numerical stability by selecting the largest available element as the pivot. This is a form of row permutation to avoid division by small numbers, which can cause numerical issues. In contrast, QR decomposition can be seen as a process that constructs a set of orthogonal vectors (the columns of Q) through a linear combination of the rows of A. This approach can be viewed as a more sophisticated form of pivoting. Instead of merely swapping rows, QR decomposition effectively 'blends' the rows of A to form the orthogonal matrix Q.

17 Numerical solution of matrix least squares by QR decomposition

The solution of least squares problems using QR decomposition is an efficient and numerically stable method. It's particularly useful for solving overdetermined systems, where you have more equations than unknowns. Let's consider a system of linear equations described by $A\mathbf{x} = \mathbf{b}$, where A is an $m \times n$ matrix with $m \geq n$, $\mathbf{x}$ is an n-dimensional vector of unknowns, and $\mathbf{b}$ is an m-dimensional vector.

The goal of the least squares solution is to find an $\mathbf{x}$ that minimizes the residual $\mathbf{r} = \mathbf{b} - A\mathbf{x}$. The norm of the residual $||\mathbf{r}||^2$ is minimized in the least squares sense.

1. **Decomposition**: First, decompose matrix A using QR decomposition. This means finding an orthogonal matrix Q and an upper triangular matrix R such that $A = QR$. The size of Q is $m \times m$ and R is $m \times n$.

2. **Multiply Both Sides by Q^T**: Multiply both sides of the original equation $A\mathbf{x} = \mathbf{b}$ by Q^T (the transpose of Q):
$$Q^T A\mathbf{x} = Q^T \mathbf{b}$$
Since $A = QR$, this simplifies to:
$$R\mathbf{x} = Q^T \mathbf{b}$$
So now we have a triangular system to solve! But, this system is not rectangular.

3. **Partition and Solve**: Partition Q and R as follows:
$$Q = [Q_1 \quad Q_2] \quad \text{and} \quad R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$$
where Q_1 is $m \times n$, Q_2 is $m \times (m-n)$, and R_1 is $n \times n$. Partition $Q^T\mathbf{b}$ into two components that match the partition of R:
$$Q^T \mathbf{b} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}$$
Then, the least squares problem reduces to solving the simpler triangular system:
$$R_1\mathbf{x} = \mathbf{b}_1$$

4. **Solve for $\mathbf{x}$**: Finally, solve the upper triangular system $R_1\mathbf{x} = \mathbf{b}_1$ for $\mathbf{x}$ using back substitution.

Lecture on November 16, lecture notes to be updfated after the class.
References for this section: [TB97]

References

[TB97] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. SIAM, 1997.

[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].