Computational methods and numerical algorithms: Lecture 3

Wangtao Lu
Zhejiang University

September 27, 2020

Contents

Wangtao Lu
Zhejiang University

2.5 Iterative Methods

Why we need iterative methods? Since the computational complexity of Gaussian elimination $\mathcal{O}(n^3)$ is too large when n is large.
As Newton's method for finding roots, we need an initial guess and refine the guess at each step, and except it converges to the true solution.

Jacobi Method

Jacobi Method: Solves the i-th equation for the i-th unknown in each iteration at each iterative step.

Example 1. Apply the Jacobi Method to the system,

$3u + v = 5, u + 2v = 5.$

Sol. We use the initial guess $u_0 = 0, v_0 = 0$. Then, from the following iterative formula:

$$u_{i+1} = (5 - v_i)/3, \quad v_{i+1} = (5 - u_i)/2,$$

we get

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} 5/3 \\ 5/2 \end{bmatrix}, \quad \begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} 5/6 \\ 5/3 \end{bmatrix},$$

$$\vdots,$$

$$\begin{bmatrix} u_{10} \\ v_{10} \end{bmatrix} \approx \begin{bmatrix} 0.999871399176955 \\ 1.999742798353910 \end{bmatrix} \ldots \begin{bmatrix} u_{20} \\ v_{20} \end{bmatrix} \approx \begin{bmatrix} 0.999999983461828 \\ 1.999999966923657 \end{bmatrix}$$

True solution, $u = 1, v = 2$.

A Failure Example

Example 2. Apply the Jacobi Method to the system
$u + 2v = 5, 3u + v = 5.$

Sol. We use the initial guess $u_0 = 0, v_0 = 0$. Then, from the following iterative formula:

$$u_{i+1} = (5 - 2v_i), \quad v_{i+1} = (5 - 3u_i),$$

we get

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \quad \begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} -5 \\ -10 \end{bmatrix},$$

$$\vdots,$$

$$\begin{bmatrix} u_{10} \\ v_{10} \end{bmatrix} = \begin{bmatrix} -7775 \\ -15550 \end{bmatrix} \cdots \begin{bmatrix} u_{20} \\ v_{20} \end{bmatrix} = \begin{bmatrix} -60466175 \\ -120932350 \end{bmatrix}$$

True solution, $u = 1, v = 2$.

Wangtao Lu
Zhejiang University

Jacobi Iteration Formula

For any given $n \times n$ matrix A and a given vector $b \in \mathbb{R}^{n\times 1}$, we can split A into

$$A = L + U + D.$$

Thus, the Jacobi Method gives the following iterative formula

$$Dx_{k+1} = b - (L+U)x_k \to x_{k+1} = D^{-1}(b - (L+U)x_k).$$

If $\{x_k\}$ converges to x, x is a fixed point of

$$x = D^{-1}(b - (L+U)x).$$

A sufficient condition for the convergence

The $n \times n$ matrix $A = (a_{ij})$ is **strictly diagonally dominant** if, for each $1 \leq i \leq n$,

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|.$$

Theorem. If A is a strictly diagonally dominant matrix of size $n \times n$, then: (1) $\det(A) \neq 0$; (2) for every b and any initial guess, the Jacobi Method applied to $Ax = b$ converges to the (unique) solution.
Proof. Recall the Jacobi's iterative formula

$$x_{k+1} = D^{-1}(b - (L + U)x_k),$$

which gives

$$(x_{k+1} - x_k) = -R(x_k - x_{k-1}), \quad R = D^{-1}(L + U).$$

But $||R||_\infty < 1$ since A is strictly diagonally dominant. Consequently, we get

$$||x_{k+1} - x_k||_\infty \leq ||R||_\infty ||x_k - x_{k-1}||_\infty \leq ||R||_\infty^{k-1} ||x_1 - x_0||.$$

Thus, for all $p \in \mathbb{N}_*$,

$$\begin{aligned} ||x_{k+p} - x_k||_\infty &\leq \sum_{i=1}^{p-1} ||x_{k+i} - x_{k+i-1}|| \\ &\leq \sum_{i=1}^{p-1} ||R||_\infty^{k+i-2} ||x_1 - x_0|| \\ &= \left[||x_1 - x_0||_\infty \frac{1 - ||R||_\infty^{p-1}}{1 - ||R||_\infty} \right] ||R||_\infty^{k-1} \to 0, \quad k \to \infty. \end{aligned}$$

This indicates that $\{x_k\}$ is a Cauchy sequence so that it converges uniquely to some vector $x \in \mathbb{R}^n$, which satisfies

$$x = D^{-1}b - D^{-1}(L + U)x.$$

This is equivalent to $Ax = b$ has a unique solution for every b and any initial guess, which in fact indicates that $det(A) \neq 0$.

Gauss-Seidel Method

In contrast to Jacobi Method, Gauss-Seidel Method uses the most recent updated values of the unknowns at each step, while Jacobi Method always uses the previous guess of unknowns at each step.

Example 3. Apply the Gauss-Seidel Method to the system, $3u + v = 5$, $u + 2v = 5$.

Sol. We still use the initial guess $u_0 = 0$, $v_0 = 0$. In the first step we get

$$u_1 = (5 - v_0)/3 = 5/3.$$

In Gauss-Seidel method, we get

$$v_1 = (5 - u_1)/2 = 5/3.$$

Then,

$$u_2 = (5 - v_1)/2 = 10/9,$$
$$v_2 = (5 - u_2)/2 = 35/18,$$

and

$$u_3 = (5 - v_2)/2 = 55/54,$$
$$v_3 = (5 - u_3)/2 = 215/108.$$

In Jacobi method, we get

$$v_1 = (5 - u_0)/2 = 5/2.$$

Then,

$$u_2 = (5 - v_1)/2 = 5/6,$$
$$v_2 = (5 - u_1)/2 = 5/3,$$

and

$$u_3 = (5 - v_2)/2 = 10/9,$$
$$v_3 = (5 - u_2)/2 = 25/12.$$

Wangtao Lu
Zhejiang University

Example 4. Apply Gauss-Seidel Method to solve

$$\begin{bmatrix} 3 & 1 & -1 \\ 2 & 4 & 1 \\ -1 & 2 & 5 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ 1 \end{bmatrix}.$$

Sol. True solution is $[2, -1, 1]^T$. The Gauss-Seidel iteration is

$$\begin{aligned} u_{k+1} &= \frac{4 - v_k + w_k}{3}, \\ v_{k+1} &= \frac{1 - 2u_{k+1} - w_k}{4}, \\ w_{k+1} &= \frac{1 + u_{k+1} - 2v_{k+1}}{5}. \end{aligned}$$

Using $u_0 = v_0 = w_0 = 0$, we get,

$$\begin{bmatrix} u_1 \\ v_1 \\ w_1 \end{bmatrix} = \begin{bmatrix} 4/3 \\ -5/12 \\ 19/30 \end{bmatrix}, \ldots, \begin{bmatrix} u_{10} \\ v_{10} \\ w_{10} \end{bmatrix} \approx \begin{bmatrix} 1.99957 \\ -0.99966 \\ 0.99977 \end{bmatrix}, \ldots$$

$$\begin{bmatrix} u_{20} \\ v_{20} \\ w_{20} \end{bmatrix} \approx \begin{bmatrix} 1.99999988 \\ -0.999999911 \\ 0.999999942 \end{bmatrix}.$$

Gauss-Seidel's Iterative Formula

Suppose

$$A = L + U + D,$$

then in the $k+1$-th iteration, we solve

$$Lx_{k+1} + Ux_k + Dx_{k+1} = b.$$

for x_{k+1}, and get

$$x_{k+1} = (L + D)^{-1}(b - Ux_k).$$

If $\{x_k\}$ converges to $x \in \mathbb{R}^n$. Then, x is a fixed-point of

$$x = (L + D)^{-1}(b - Ux).$$

When will Gauss-Seidel iteration converges?

Convergence Theorem

Theorem. If the $n \times n$ matrix A is strictly diagonally dominant, then: (1) A is nonsingular, i.e., $det(A) \neq 0$; (2) for every b and every starting guess, the Gauss-Seidel Method applied to $Ax = b$ converges to the solution to $Ax = b$.

Successive Over-Relaxation

For a given parameter $\omega \in \mathbb{R}$, in Successive Over-Relaxation, we define each component of the new guess x_{k+1} as a weighted average of ω times the Gauss-Seidel formula and $1-\omega$ times the current guess. The number ω is called the relaxation parameter; the case when $\omega > 1$ is referred to as over-relaxation.

Iterative formula:

$$x_0 = \text{initial guess},$$
$$x_{k+1} = (\omega L + D)^{-1}[(1-\omega)Dx_k - \omega U x_k] + \omega(D+\omega L)^{-1} b.$$

How to derive this?

Successive Over-Relaxation

Wangtao Lu
Zhejiang University

For a given parameter $\omega \in \mathbb{R}$, in Successive Over-Relaxation, we define each component of the new guess x_{k+1} as a weighted average of ω times the Gauss-Seidel formula and $1-\omega$ times the current guess. The number ω is called the relaxation parameter; the case when $\omega > 1$ is referred to as over-relaxation.
Iterative formula:

$$\begin{aligned} x_0 &= \text{initial guess}, \\ x_{k+1} &= (\omega L + D)^{-1}[(1-\omega)Dx_k - \omega U x_k] + \omega(D+\omega L)^{-1} b. \end{aligned}$$

How to derive this? the m-th component of the $k+1$-th guess x_{k+1}^m in the SOR method,

Successive Over-Relaxation

For a given parameter $\omega \in \mathbb{R}$, in Successive Over-Relaxation, we define each component of the new guess x_{k+1} as a weighted average of ω times the Gauss-Seidel formula and $1-\omega$ times the current guess. The number ω is called the relaxation parameter; the case when $\omega > 1$ is referred to as over-relaxation.
Iterative formula:

$$x_0 = \text{initial guess},$$
$$x_{k+1} = (\omega L + D)^{-1}[(1-\omega)Dx_k - \omega U x_k] + \omega(D+\omega L)^{-1}b.$$

How to derive this? the m-th component of the $k+1$-th guess x_{k+1}^m in the SOR method,

$$x_{k+1}^m = (1-\omega)x_k^m + \omega \frac{b^m - \sum_{j>m} a_{mj}x_k^j - \sum_{j<m} a_{mj}x_{k+1}^j}{a_{mm}}.$$

Example 5. Compare Jacobi, Gauss-Seidel, and SOR on the system of six equations in six unknowns:

$$\begin{bmatrix} 3 & -1 & 0 & 0 & 0 & \frac{1}{2} \\ -1 & 3 & -1 & 0 & \frac{1}{2} & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & 0 \\ 0 & \frac{1}{2} & 0 & -1 & 3 & -1 \\ \frac{1}{2} & 0 & 0 & 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} \frac{5}{2} \\ \frac{3}{2} \\ 1 \\ 1 \\ \frac{3}{2} \\ \frac{5}{2} \end{bmatrix}$$

Sol. The true solution is $[1, 1, 1, 1, 1, 1]^T$. The following table shows the result after six iterations of each of the three methods.

Jacobi	Gauss-Seidel	SOR ($\omega = 1.1$)
0.9879	0.9950	0.9989
0.9846	0.9946	0.9993
0.9674	0.9969	0.9996
0.9674	0.9996	1.0004
0.9846	1.0016	1.0009
0.9879	1.0013	1.0004

Sparse Matrix

A matrix is called **sparse** if many of the matrix entries are known to be zero. Typically, a sparse matrix only contains $O(n)$ nonzero entries. A **full** matrix is the opposite, only a few entries are nonzero.
When the matrix is of extremely big size but sparse with $O(n)$ elements, Gaussian elimination loses efficiency since the LU factorization causes **fill-in** since the resulting matrices L and U are always full; in contrast, iterative methods are always more desirable/attractive.

See MATLAB for the Reference Page for the built-in function sparse!

Sparse Matrix

A matrix is called **sparse** if many of the matrix entries are known to be zero. Typically, a sparse matrix only contains $O(n)$ nonzero entries. A **full** matrix is the opposite, only a few entries are nonzero.
When the matrix is of extremely big size but sparse with $O(n)$ elements, Gaussian elimination loses efficiency since the LU factorization causes **fill-in** since the resulting matrices L and U are always full; in contrast, iterative methods are always more desirable/attractive.
A question: What is the computational complexity of the product operation of a sparse matrix A with $O(n)$ elements and a vector b?
See MATLAB for the Reference Page for the built-in function sparse!

2.6 Symmetric positive-definite matrices

The $n \times n$ matrix A is **symmetric** if $A^T = A$. The matrix A is **positive-definite** if $x^T Ax > 0$ for all vectors $x \neq 0$.

Example 1. Show that $A = \begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix}$ is symmetric positive-definite.

Sol. By completing squares,

$$x^T Ax = 2x_1^2 + 4x_1x_2 + 5x_2^2 = 2(x_1 + x_2)^2 + 3x_2^2 > 0,$$

for all $x \neq 0$. Thus, A is strictly positive-definite.

Example 2. Show that $A = \begin{bmatrix} 2 & 4 \\ 4 & 5 \end{bmatrix}$ is not positive-definite.

Sol. By completing squares,

$$x^T Ax = 2x_1^2 + 8x_1x_2 + 5x_2^2 = 2(x_1 + 2x_2)^2 - 3x_2^2.$$

Thus, if $x_2 = 1$ and $x_1 = -2x_2 = -2$, then $x^T Ax = -3 < 0$, which indicates that A is not strictly positive-definite.

Wangtao Lu
Zhejiang University

Properties of a strictly positive-definite matrix?

Wangtao Lu
Zhejiang University

Property 1. A symmetric matrix A is strictly positive-definite if and only if all of its eigenvalues are positive.
Property 2. If A is $n \times n$ symmetric and strictly positive-definite and X is an $n \times m$ matrix of full rank with $n \geq m$, then $X^T AX$ is $m \times m$ symmetric and strictly positive-definite.
A **principle submatrix** of a square matrix A is a square submatrix whose diagonal entries are diagonal entries of A; for example,

$$\begin{bmatrix} 2 & 4 \\ 4 & 5 \end{bmatrix}$$

is a principle submatrix of

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 3 & 4 \\ 7 & 4 & 5 & 6 \\ 9 & 4 & 8 & 5 \end{bmatrix}$$

Property 3. Any principle submatrix of a symmetric and strictly positive-definite matrix is symmetric and strictly positive definite.

Cholesky factorization

For a symmetric positive-definite matrix A, it has a much more special factorization than the usual LU factorization.
Theorem. If A is symmetric positive-definite $n \times n$ matrix, then there exists an upper triangular $n \times n$ matrix R such that $A = R^T R$; and this decomposition is called the **Cholesky Factorization**.
Proof. We construct R by induction on the size n. Then case when $n = 1$ is trivial since one just takes $R = \sqrt{A}$ and it is done. Consider A partitioned as

$$A = \left[\begin{array}{c|c} a & b^T \\ \hline b & C \end{array}\right].$$

where $a > 0$, $b \in \mathbb{R}^{n-1}$, and C is $(n-1) \times (n-1)$ symmetric and strictly positive-definite matrix. To zero b^T and b in A, we define

$$L = \left[\begin{array}{c|c} 1 & -u^T \\ \hline & I_{n-1} \end{array}\right], \quad u = \frac{b}{a} \in \mathbb{R}^{n-1}.$$

Then,

$$L^TAL = \left[\begin{array}{c|c} 1 & \\ \hline -u & I_{n-1} \end{array}\right] \left[\begin{array}{c|c} a & b^T \\ \hline b & C \end{array}\right] \left[\begin{array}{c|c} 1 & -u^T \\ \hline & I_{n-1} \end{array}\right] = \left[\begin{array}{c|c} a & \\ \hline & C - \frac{bb^T}{a} \end{array}\right]$$

Notice that $C - \frac{bb^T}{a}$ is an $(n-1) \times (n-1)$ symmetric positive-definite matrix so that there exists an $(n-1) \times (n-1)$ matrix R_{n-1} such that $C - \frac{bb^T}{a} = R_{n-1}^T R_{n-1}$. Thus,

$$L^TAL = \left[\begin{array}{c|c} a & \\ \hline & R_{n-1}^T R_{n-1} \end{array}\right] = \left[\begin{array}{c|c} \sqrt{a} & \\ \hline & R_{n-1} \end{array}\right]^T \left[\begin{array}{c|c} \sqrt{a} & \\ \hline & R_{n-1} \end{array}\right].$$

Then, directly taking

$$R = \left[\begin{array}{c|c} \sqrt{a} & \\ \hline & R_{n-1} \end{array}\right] L^{-1}, \quad L^{-1} = \left[\begin{array}{c|c} 1 & -u^T \\ \hline & I_{n-1} \end{array}\right]^{-1} =$$

one complets the proof.

Then,

$$L^T A L = \left[\begin{array}{c|c} 1 & \\ \hline -u & I_{n-1} \end{array}\right] \left[\begin{array}{c|c} a & b^T \\ \hline b & C \end{array}\right] \left[\begin{array}{c|c} 1 & -u^T \\ \hline & I_{n-1} \end{array}\right] = \left[\begin{array}{c|c} a & \\ \hline & C - \frac{bb^T}{a} \end{array}\right]$$

Notice that $C - \frac{bb^T}{a}$ is an $(n-1) \times (n-1)$ symmetric positive-definite matrix so that there exists an $(n-1) \times (n-1)$ matrix R_{n-1} such that $C - \frac{bb^T}{a} = R_{n-1}^T R_{n-1}$. Thus,

$$L^T A L = \left[\begin{array}{c|c} a & \\ \hline & R_{n-1}^T R_{n-1} \end{array}\right] = \left[\begin{array}{c|c} \sqrt{a} & \\ \hline & R_{n-1} \end{array}\right]^T \left[\begin{array}{c|c} \sqrt{a} & \\ \hline & R_{n-1} \end{array}\right].$$

Then, directly taking

$$R = \left[\begin{array}{c|c} \sqrt{a} & \\ \hline & R_{n-1} \end{array}\right] L^{-1}, \quad L^{-1} = \left[\begin{array}{c|c} 1 & -u^T \\ \hline & I_{n-1} \end{array}\right]^{-1} = \left[\begin{array}{c|c} 1 & u^T \\ \hline & I_{n-1} \end{array}\right],$$

one complets the proof.

Cholesky factorization

Wangtao Lu
Zhejiang University

From the above proof, we see that, for a symmetric positive-definite matrix

$$A = \left[\begin{array}{c|c} a & b^T \\ \hline b & C \end{array}\right].$$

, its Cholesky factorization is $A = R^T R$ with

$$R = \left[\begin{array}{c|c} \sqrt{a} & b^T/\sqrt{a} \\ \hline & R_{n-1} \end{array}\right],$$

where R_{n-1} satisfies $C - bb^T/a = R_{n-1}^T R_{n-1}$.

Pseudocode of Cholesky factorization

for $k = 1, 2, \ldots, n$
  if $A_{kk} < 0$, **stop**, **end**
  $R_{kk} = \sqrt{A_{kk}}$
  $u^T = \frac{1}{R_{kk}} A_{k,k+1:n} \qquad R_{k,k+1:n} = u^T$
  $A_{k+1:n,k+1:n} = A_{k+1:n,k+1:n} - uu^T$
end

Example 3. Find the Cholesky factorization of

$$A = \left[\begin{array}{c|cc} 4 & -2 & 2 \\ \hline -2 & 2 & -4 \\ 2 & -4 & 11 \end{array}\right]$$

Sol. First, $u = [-2/\sqrt{4}, 2/\sqrt{4}]^T = [-1, 1]^T$. Thus,

$$R = \left[\begin{array}{c|cc} \sqrt{4} & -1 & 1 \\ \hline & & \\ & & \end{array}\right], A_{2:3,2:3} = \left[\begin{array}{c|c} 2-(-1)^2 & -4-(-1)\times 1 \\ -4-(-1)\times 1 & 11-1\times 1 \end{array}\right]$$

Second, $u = -3$ Thus,

$$R = \left[\begin{array}{c|c|c} 2 & -1 & 1 \\ \hline & 1 & -3 \\ \hline & & \end{array}\right], A_{3,3} = 10 - (-3)^T(-3) = 1.$$

Pseudocode of Cholesky factorization

for $k = 1, 2, \ldots, n$
  if $A_{kk} < 0$, **stop**, **end**
  $R_{kk} = \sqrt{A_{kk}}$
  $u^T = \frac{1}{R_{kk}} A_{k,k+1:n} \qquad R_{k,k+1:n} = u^T$
  $A_{k+1:n,k+1:n} = A_{k+1:n,k+1:n} - uu^T$
end

Example 3. Find the Cholesky factorization of

$$A = \left[\begin{array}{c|cc} 4 & -2 & 2 \\ \hline -2 & 2 & -4 \\ 2 & -4 & 11 \end{array}\right]$$

Sol. First, $u = [-2/\sqrt{4}, 2/\sqrt{4}]^T = [-1, 1]^T$. Thus,

$$R = \left[\begin{array}{c|cc} \sqrt{4} & -1 & 1 \\ \hline & & \\ & & \end{array}\right], A_{2:3,2:3} = \left[\begin{array}{c|c} 1 & -3 \\ -3 & 10 \end{array}\right].$$

Second, $u = -3$ Thus,

$$R = \left[\begin{array}{c|c|c} 2 & -1 & 1 \\ \hline & 1 & -3 \\ \hline & & \end{array}\right], A_{3,3} = 10 - (-3)^T(-3) = 1.$$

Pseudocode of Cholesky factorization

for $k = 1, 2, \ldots, n$
  if $A_{kk} < 0$, **stop**, **end**
  $R_{kk} = \sqrt{A_{kk}}$
  $u^T = \frac{1}{R_{kk}} A_{k,k+1:n} \qquad R_{k,k+1:n} = u^T$
  $A_{k+1:n,k+1:n} = A_{k+1:n,k+1:n} - uu^T$
end

Example 3. Find the Cholesky factorization of

$$A = \left[\begin{array}{c|cc} 4 & -2 & 2 \\ \hline -2 & 2 & -4 \\ 2 & -4 & 11 \end{array}\right]$$

Sol. First, $u = [-2/\sqrt{4}, 2/\sqrt{4}]^T = [-1, 1]^T$. Thus,

$$R = \left[\begin{array}{c|cc} \sqrt{4} & -1 & 1 \\ \hline & & \\ & & \end{array}\right], A_{2:3,2:3} = \left[\begin{array}{c|c} 1 & -3 \\ -3 & 10 \end{array}\right].$$

Second, $u = -3$ Thus,

$$R = \left[\begin{array}{c|c|c} 2 & -1 & 1 \\ \hline & 1 & -3 \\ \hline & & 1 \end{array}\right], A_{3,3} = 10 - (-3)^T(-3) = 1.$$

Conjugated Gradient Method

Let A be a symmetric positive-definite $n \times n$ matrix. For two n- vectors v and w, define the **A-inner product**

$$(v, w)_A = v^T A w.$$

The vectors v and w are **A-conjugate** if $(v, w)_A = 0$. Differentiate from the Euclidean inner product $a^T b$.

Property 1. Suppose the nonzero vectors $\{x_k\}_{k=1}^n$ in $\mathbb{R}^n$ satisfy

$$(x_k, x_j) = 0, \forall k \neq j,$$

then $\{x_k\}_{k=1}^n$ is a basis of $\mathbb{R}^n$, where $(\cdot, \cdot)$ is an inner-product in $\mathbb{R}^n$.

Property 2. For vectors $\{x_k\}_{k=1}^n$ in $\mathbb{R}^n$, suppose

$$(r, x_k) = 0, \forall k = 1, \ldots, n_0 \leq n.$$

then

$$(r, \sum_{k=1}^{n_0} c_k x_k) = 0, \forall c_k \in \mathbb{R}, k = 1, \ldots, n_0.$$

.

Property 3. Suppose $\{x_k\}_{k=1}^n$ is a basis of the space $\mathbb{R}^n$. Then, for any vector $r \in \mathbb{R}^n$, if

$$(r, x_k) = 0, \quad \forall k = 1, \ldots, n,$$

then $r = 0_n$, where $(\cdot, \cdot)$ is an inner-product in $\mathbb{R}^n$.

Motivation of CG Method

Conjugated Gradient method is an iterative method for solving $Ax = b$ when A is (especially large and sparse,) **symmetric and strictly positive-definite**.

Suppose CG method gives a sequence of guess solutions $\{x_k\}_{k=1}^{\infty}$ to $Ax = b$. Define

$$r_k = b - Ax_k,$$

and the searching direction at step k is d_k such that there exist $\alpha_k, \beta_k \in \mathbb{R}$

$$x_{k+1} = x_k + \alpha_k d_k, \quad d_{k+1} = r_{k+1} + \beta_k d_k.$$

The Conjugated Gradient method selects α_k and β_k in the following way:

1. d_{k+1} is A-conjugate to $\{d_i\}_{i=1}^{k}$in the sense that $d_k^T A d_i = 0$.
2. r_{k+1} is orthogonal to $\{r_i\}_{i=1}^{k}$ in the sense that $r_{k+1}^T r_i = 0$,

.

Motivation of CG Method

Conjugated Gradient method is an iterative method for solving $Ax = b$ when A is (especially large and sparse,) **symmetric and strictly positive-definite**.

Suppose CG method gives a sequence of guess solutions $\{x_k\}_{k=1}^{\infty}$ to $Ax = b$. Define

$$r_k = b - Ax_k,$$

and the searching direction at step k is d_k such that there exist $\alpha_k, \beta_k \in \mathbb{R}$

$$x_{k+1} = x_k + \alpha_k d_k, \quad d_{k+1} = r_{k+1} + \beta_k d_k.$$

The Conjugated Gradient method selects α_k and β_k in the following way:

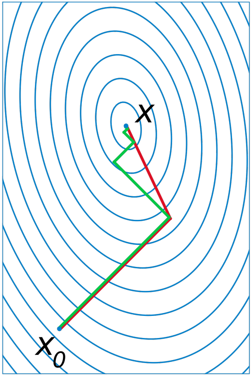
1. d_{k+1} is A-conjugate to $\{d_i\}_{i=1}^{k}$in the sense that $d_k^T A d_i = 0$.
2. r_{k+1} is orthogonal to $\{r_i\}_{i=1}^{k}$ in the sense that $r_{k+1}^T r_i = 0$, and so orthogonal to d_i.

CG method (red) vs. Steepest Descent Method (green)

Consider the following minimization problem:

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T A x - x^T b,$$

where A is an $n \times n$, symmetric, positive-definite matrix.



Iterative formula in CG method

Initial guess:

$$x_0, \quad d_0 = r_0 = b - Ax_0.$$

Iterative formula,

$$x_{k+1} = x_k + \alpha_k d_k, \quad d_{k+1} = r_{k+1} + \beta_k d_k.$$

We need to find both α_k and β_k to proceed with the iteration. Since

$$r_{k+1} = b - Ax_{k+1} = r_k - \alpha_k A d_k.$$

Thus,

$$0 = d_k^T r_{k+1} = d_k^T r_k - \alpha_k d_k^T A d_k \to \alpha_k = \frac{d_k^T r_k}{d_k^T A d_k} = \frac{r_k^T r_k}{d_k^T A d_k},$$

and

$$r_{k+1}^T r_{k+1} = 0 - \alpha_k r_{k+1}^T A d_k = -\frac{r_k^T r_k}{d_k^T A d_k} r_{k+1}^T A d_k.$$

On the other hand,

$$0 = d_k^T A d_{k+1} = d_k^T A r_{k+1} + \beta_k d_k^T A d_k \to \beta_k = -\frac{d_k^T A r_{k+1}}{d_k^T A d_k} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}.$$

Operation Count

A Pseudocode of CG method:

x_0 = initial guess

$d_0 = r_0 = b - Ax_0$

for $k = 0, 1, 2, \ldots, n-1$

  if $r_k = 0$, **stop**, **end**

  $\alpha_k = \frac{d_k^T r_k}{d_k^T A d_k} = \frac{r_k^T r_k}{d_k^T A d_k}$

  $x_{k+1} = x_k + \alpha_k d_k$

  $r_{k+1} = b - Ax_{k+1} = r_k - \alpha_k A d_k$

  $\beta_k = -\frac{d_k^T A r_{k+1}}{d_k^T A d_k} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$

  $d_{k+1} = r_{k+1} + \beta_k d_k$

end

Operation Count (Computational Complexity):

	CG	LU
full A		$\frac{n^3}{3}$
sparse A with $O(n)$ entries		

Wangtao Lu
Zhejiang University

Operation Count

A Pseudocode of CG method:

x_0 = initial guess
$d_0 = r_0 = b - Ax_0$
for $k = 0, 1, 2, \ldots, n-1$
  if $r_k = 0$, **stop**, **end**
  $\alpha_k = \frac{d_k^T r_k}{d_k^T A d_k} = \frac{r_k^T r_k}{d_k^T A d_k}$
  $x_{k+1} = x_k + \alpha_k d_k$
  $r_{k+1} = b - Ax_{k+1} = r_k - \alpha_k A d_k$
  $\beta_k = -\frac{d_k^T A r_{k+1}}{d_k^T A d_k} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
  $d_{k+1} = r_{k+1} + \beta_k d_k$
end

Operation Count (Computational Complexity):

	CG	LU
full A	$n^3 + \mathcal{O}(n^2)$	$\frac{n^3}{3}$
sparse A with $O(n)$ entries		

Operation Count

A Pseudocode of CG method:

x_0 = initial guess
$d_0 = r_0 = b - Ax_0$
for $k = 0, 1, 2, \ldots, n-1$
  if $r_k = 0$, **stop**, **end**
  $\alpha_k = \frac{d_k^T r_k}{d_k^T A d_k} = \frac{r_k^T r_k}{d_k^T A d_k}$
  $x_{k+1} = x_k + \alpha_k d_k$
  $r_{k+1} = b - Ax_{k+1} = r_k - \alpha_k A d_k$
  $\beta_k = -\frac{d_k^T A r_{k+1}}{d_k^T A d_k} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
  $d_{k+1} = r_{k+1} + \beta_k d_k$
end

Operation Count (Computational Complexity):

	CG	LU
full A	$n^3 + \mathcal{O}(n^2)$	$\frac{n^3}{3}$
sparse A with $O(n)$ entries	$\mathcal{O}(n^2)$	$\frac{n^3}{3}$

Convergence of CG method

Theorem Suppose A is an $n \times n$ symmetric, positive-definite matrix and b is an $n \times 1$ matrix. Then the Conjugated Gradient method finds the solution of $Ax = b$ in at most n steps for any initial guess vector $x_0 \in \mathbb{R}^n$.

Proof. Suppose the CG method produces a sequence of guess solutions $\{x_k\}_{k=0}^K$ such that $\{r_k\}_{k=0}^K$ are all nonzero and we let K to be the maximum integer that satisfies this property.
We claim $K < n$, since $\{r_k\}_{k=0}^K$ are orthogonal to each other and form a basis set of $\mathbb{R}^n$.
This indicates that $r_{K+1} = 0$ and the CG method finds the solution x_{K+1} to $Ax = b$ in $K + 1 \leq n$ steps.

Example 4. Solve

$$\begin{bmatrix} 2 & 2 \\ 2 & 5 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 6 \\ 3 \end{bmatrix},$$

by the CG method in exact arithmetic and in IEEE double-precision arithmetic. Sol. Let $x_0 = (0,0)^T$, $r_0 = d_0 = (6,3)^T$,

In exact arithmetic, we have,

$$\alpha_0 = r_0^T r_0/(d_0^T A d_0)^T = \frac{5}{21}$$

$$x_1 = x_0 + \alpha_0 d_0 = (\frac{10}{7}, \frac{5}{7})^T$$

$$r_1 = r_0 - \alpha_0 A d_0 = (\frac{12}{7}, \frac{-24}{7})^T$$

$$\beta_0 = r_1^T r_1/(r_0^T r_0) = \frac{16}{49}$$

$$d_1 = r_1 + \beta_0 d_0 = (\frac{180}{49}, \frac{-120}{49})$$

$$\alpha_1 = r_1^T r_1/(d_1^T A d_1) = \frac{7}{10}$$

$$x_2 = x_1 + \alpha_1 d_1 = (4, -1)^T$$

$$r_2 = r_1 - \alpha_1 A d_1 = (0,0)^T, \text{STOP}.$$

In IEEE double-precision arithmetic, we get from MATLAB the following guess solutions

$$x_0 = (0,0)^T, \quad r_0 = (6,3)^T,$$

$$x_1 = (1.428571428571428, \; 0.714285714285714),$$

$$r_1 = (1.714285714285714, \; -3.428571428571428),$$

$$x_2 = (3.999999999999999, \; -1.000000000000000)$$

$$r_2 = (1.77e-14, 0.089e-14),$$

$$x_3 = (3.999999999999999, \; -1.000000000000000),$$

$$r_3 = (0.051e-14, -0.102e-14).$$

Preconditioned Conjugated Gradient Method

Why **Preconditioning**? When A is **ill-conditioned**, CG method could accumulate round-off errors quickly.
What is a **preconditioner**? Instead of solving

$$Ax = b,$$

we solve

$$M^{-1}Ax = M^{-1}b.$$

Here the invertible matrix M should satisfy the following properties:

1. M should be as close to A as possible;
2. M should be easy to invert.

2^*. M should be strictly positive-definite, symmetric so that the CG method applies here.

Suppose $A = L + D + U$ such that $U = L^T$, we could use the following two possible preconditioners in practice.

1. **Jacobi preconditioner**: $M = D$.
2. **Symmetric successive over-relaxation preconditioner**: $M = (D + wL)D^{-1}(D + wU)$ for $w \in [0, 2]$. The special case $w = 1$ is called the **Gauss-Seidel preconditioner**.

Pseudocode of PCG method

When preconditioned, the new coefficient matrix $M^{-1}A$ in general is no longer symmetric, positive-definite in the usual Euclidean inner product. But it is positive-definite in the M-inner product in the sense that

$$(M^{-1}Au, u)_M \geq 0.$$

CG Pseudocode:
x_0 = initial guess
$d_0 = r_0 = b - Ax_0$
for $k = 0, 1, 2, \ldots, n-1$
  if $r_k = 0$, **stop**, **end**
  $\alpha_k = \frac{r_k^T r_k}{d_k^T A d_k} = \frac{(r_k, r_k)}{(d_k, Ad_k)}$
  $x_{k+1} = x_k + \alpha_k d_k$
  $r_{k+1} = r_k - \alpha_k A d_k$
  $\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} = \frac{(r_{k+1}, r_{k+1})}{(r_k, r_k)}$
  $d_{k+1} = r_{k+1} + \beta_k d_k$
end

PCG Pseudocode:
x_0 = initial guess
$d_0 = z_0 = M^{-1}b - M^{-1}Ax_0$
for $k = 0, 1, 2, \ldots, n-1$
  if $z_k = 0$, **stop**, **end**
  $\alpha_k = \frac{(z_k, z_k)_M}{(d_k, M^{-1}Ad_k)_M}$
  $x_{k+1} = x_k + \alpha_k d_k$
  $z_{k+1} = z_k - \alpha_k M^{-1} A d_k$
  $\beta_k = \frac{(z_{k+1}, z_{k+1})_M}{(z_k, z_k)_M}$
  $d_{k+1} = z_{k+1} + \beta_k d_k$
end

Pseudocode of PCG method

When preconditioned, the new coefficient matrix $M^{-1}A$ in general is no longer symmetric, positive-definite in the usual Euclidean inner product. But it is positive-definite in the M-inner product in the sense that

$$(M^{-1}Au, u)_M = (M^{-1}Au)^T Mu = u^T Au \geq 0.$$

CG Pseudocode:

$x_0 =$ initial guess
$d_0 = r_0 = b - Ax_0$
for $k = 0, 1, 2, \ldots, n-1$
  if $r_k = 0$, **stop**, **end**
  $\alpha_k = \frac{r_k^T r_k}{d_k^T A d_k} = \frac{(r_k, r_k)}{(d_k, Ad_k)}$
  $x_{k+1} = x_k + \alpha_k d_k$
  $r_{k+1} = r_k - \alpha_k A d_k$
  $\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} = \frac{(r_{k+1}, r_{k+1})}{(r_k, r_k)}$
  $d_{k+1} = r_{k+1} + \beta_k d_k$
end

PCG Pseudocode: $z_k = M^{-1} r_k$.

$x_0 =$ initial guess
$d_0 = z_0 = M^{-1}b - M^{-1}Ax_0$
for $k = 0, 1, 2, \ldots, n-1$
  if $z_k = 0$, **stop**, **end**
  $\alpha_k = \frac{(z_k, z_k)_M}{(d_k, M^{-1}Ad_k)_M}$
  $x_{k+1} = x_k + \alpha_k d_k$
  $z_{k+1} = z_k - \alpha_k M^{-1} A d_k$
  $\beta_k = \frac{(z_{k+1}, z_{k+1})_M}{(z_k, z_k)_M}$
  $d_{k+1} = z_{k+1} + \beta_k d_k$
end

Pseudocode of PCG method

When preconditioned, the new coefficient matrix $M^{-1}A$ in general is no longer symmetric, positive-definite in the usual Euclidean inner product. But it is positive-definite in the M-inner product in the sense that

$$(M^{-1}Au, u)_M \geq 0.$$

PCG old Pseudocode:

$x_0 =$ initial guess
$d_0 = z_0 = M^{-1}b - M^{-1}Ax_0$
for $k = 0, 1, 2, \ldots, n-1$
  if $z_k = 0$, **stop**, **end**
  $\alpha_k = \frac{(z_k, z_k)_M}{(d_k, M^{-1}Ad_k)_M}$
  $x_{k+1} = x_k + \alpha_k d_k$
  $z_{k+1} = z_k - \alpha_k M^{-1}Ad_k$
  $\beta_k = \frac{(z_{k+1}, z_{k+1})_M}{(z_k, z_k)_M}$
  $d_{k+1} = z_{k+1} + \beta_k d_k$
end

PCG Pseudocode: $z_k = M^{-1}r_k$.

$x_0 =$ initial guess
$r_0 = b - Ax_0$
$d_0 = z_0 = M^{-1}r_0$
for $k = 0, 1, 2, \ldots, n-1$
  if $r_k = 0$, **stop**, **end**
  $\alpha_k = \frac{(r_k, z_k)}{(d_k, Ad_k)}$
  $x_{k+1} = x_k + \alpha_k d_k$
  $r_{k+1} = r_k - \alpha_k Ad_k$
  $z_{k+1} = M^{-1}r_k$
  $\beta_k = \frac{(r_{k+1}, z_{k+1})}{(r_k, z_k)}$
  $d_{k+1} = z_{k+1} + \beta_k d_k$
end

How to "invert" the preconditioner M?

For the SSOR preconditioner

$$M = (D + wL)D^{-1}(D + wU),$$

when we perform $z_k = M^{-1}r_k$ in the PCG code, we don't need to really do the inversion of M. In stead, we solve the linear system

$$Mz_k = r_k,$$

in two steps.

1. Forward solve $(I + wLD^{-1})c_k = r_k$ for c_k;
2. Backward solve $(D + wU)z_k = c_k$ for z_k.

An example

Example 1. Let A be an $n \times n$ matrix with

$$A_{ii} = \sqrt{i}, \quad i = 1, \ldots, n,$$
$$A_{i,i+10} = A_{i+10,i} = \cos i, \quad i = 1, \ldots, n-10,$$

and let $b = A\mathrm{ones}(n, 1)$. Using the PCG method to solve $Ax = b$ using the following three preconditioners: (1) $M = \mathrm{eye}(n)$; (2) Jacobi preconditioner; (3) Gauss-Seidel preconditioner.
Sol.

Numerical Results have been given in the Textbook.

2.7 Multivariate Newton's Method

Recall Newton's method for solving $f(x) = 0$ for x: $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$.
Can we extend Newton's method to a nonlinear system of equations,

$$\begin{aligned} f^1(x^1, x^2, x^3) &= 0, \\ f^2(x^1, x^2, x^3) &= 0, \\ f^3(x^1, x^2, x^3) &= 0, \end{aligned}$$

for x^1, x^2, x^3? In vector form, we solve $\vec{f}(\vec{x}) = 0$ for $\vec{x} = (x^1, x^2, x^3)$ where $\vec{f} = (f^1, f^2, f^3)$?
Suppose $\vec{x}_k$ is close to a solution vector $\vec{r}$ of $\vec{f}(\vec{x}) = 0$, i.e., $\vec{f}(\vec{r}) = 0$.
Then, by Taylor's expansion, we see that

$$0_3 = \vec{f}(\vec{r}) = \vec{f}(\vec{x}_k) + \mathrm{D}\vec{f}(\vec{x}_k)(\vec{r} - \vec{x}_k) + \mathcal{O}(||\vec{r} - \vec{x}_k||^2),$$

where

$$\mathrm{D}\vec{f}(\vec{x}) = \begin{bmatrix} \frac{\partial f^1}{\partial x^1} & \frac{\partial f^1}{\partial x^2} & \frac{\partial f^1}{\partial x^3} \\ \frac{\partial f^2}{\partial x^1} & \frac{\partial f^2}{\partial x^2} & \frac{\partial f^2}{\partial x^3} \\ \frac{\partial f^3}{\partial x^1} & \frac{\partial f^3}{\partial x^2} & \frac{\partial f^3}{\partial x^3} \end{bmatrix}.$$

A refined guess solution $\vec{x}_{k+1}$ satisfies

$$0 = \vec{f}(\vec{x}_k) + \mathrm{D}\vec{f}(\vec{x}_k)(\vec{x}_{k+1} - \vec{x}_k) \to \vec{x}_{k+1} = \vec{x}_k - [\mathrm{D}\vec{f}(\vec{x}_k)]^{-1}\vec{f}(\vec{x}_k).$$

Assignment III

Assignment III:
P116, 2.5 Computer Problems:2,6.
P130, 2.6 Computer Problems:6. (Don't use the built-in function in MATLAB!).
Due date: October 21, 2020.