STAT 309: MATHEMATICAL COMPUTATIONS I
FALL 2023
LECTURE 15

1. WHY ITERATIVE METHODS

- if we have a linear system $A\mathbf{x} = \mathbf{b}$ where A is very, very large but is either sparse or structured (e.g., banded, Toeplitz, banded plus low-rank, semiseparable, Hierarchical, etc), the easiest way to exploit this is to use *iterative methods*
- these are methods that construct a sequence of vectors $\mathbf{x}^{(k)}$ so that $\lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x} = A^{-1}\mathbf{b}$
- we shall focus on solving linear systems but there are also iterative methods for least squares problems, eigenvalue problems, singular value problems, etc — in fact for the last two, there are only iterative methods
- one big advantage of iterative methods is that we can control how accurate we want our solution, for example, if we want our solution to be ε-accurate (whether relative or absolute), then in principle we can stop as soon as

$$\|\mathbf{x}^{(k)} - \mathbf{x}\| < \varepsilon \quad \text{or} \quad \frac{\|\mathbf{x}^{(k)} - \mathbf{x}\|}{\|\mathbf{x}\|} < \varepsilon \tag{1.1}$$

- if, say, $n = 10,000$ but it takes only $k = 5$ iterations to reach our desired level of accuracy, then we have saved a lot of computations — direct methods like LU, QR, Cholesky, etc, do not allow this
- in practice of course we do not know $\mathbf{x} = A^{-1}\mathbf{b}$ and it might appear that we can't use forward errors like those in (1.1) to control accuracy but we will see later that we don't need to know $\mathbf{x}$ to gurantee (1.1)
- usually iterative methods converge in the limit to the solution but there are iterative methods that actually converge in finitely many steps
- for example, many *Krylov subspace methods* converge in d steps where d = number of distinct nonzero eigenvalues of A:
  - conjugate gradient (CG) method for symmetric positive definite A
  - minimal residual (MINRES) method for symmetric A
  - general minimial resitual (GMRES) method for general A
- there are three classes of iterative methods for $A\mathbf{x} = \mathbf{b}$
  - *splitting methods*: decompose A into the sum of two matrices

    $$A = M - N$$

    where M is easy to invert and then do

    $$M\mathbf{x}^{(k)} = N\mathbf{x}^{(k-1)} + \mathbf{b}$$

    these are also known as *one-step stationary methods*
  - *semi-iterative methods*: generate

    $$\mathbf{y}^{(k)} = B\mathbf{y}^{(k-1)} + \mathbf{c}$$

Date: November 8, 2023, version 1.0. Comments, bug reports: `lekheng@uchicago.edu`.

for suitable B and $\mathbf{c}$ and then form

$$\mathbf{x}^{(k)} = \sum_{j=0}^{k} \alpha_{jk} \mathbf{y}^{(j)}$$

- *Krylov subspace methods*: based on the idea that if the minimum polynomial is $m_A(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_d x^d$, then

  $$A^{-1} = -\frac{c_1}{c_0} I - \frac{c_2}{c_0} A - \cdots - \frac{c_d}{c_0} A^{d-1},$$

  noting that $d =$ number of distinct eigenvalues by Homework 0, Problem 6(a); thus

  $$A^{-1}\mathbf{b} \in \operatorname{span}\{\mathbf{b}, A\mathbf{b}, A^2, \mathbf{b}, \ldots, A^{d-1}\mathbf{b}\}$$

  and we find

  $$\mathbf{x}^{(k)} \in \operatorname{span}\{\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, \ldots, A^k\mathbf{b}\}$$

  in a way that approximates the solution, i.e., $\mathbf{x}^{(k)} \approx A^{-1}\mathbf{b}$

- splitting methods and semi-iterative methods are often called *stationary methods* to distinguish them from Krylov subspace methods (although this is not so clear cut — for example, conjugate gradient method, the oldest Krylov subspace method, may also be viewed as a semi-iterative method)
- we will only have time to discuss splitting methods

2. SPLITTING METHODS

- we want to solve $A\mathbf{x} = \mathbf{b}$ for $A \in \mathbb{R}^{n\times n}$ nonsingular
- we pick a suitable *splitting*

  $$A = M - N$$

  where M is nonsingular and easy to invert (not explicitly but in the sense that it is easy to solve $M\mathbf{x} = \mathbf{b}$ for any $\mathbf{b}$)
- from $A\mathbf{x} = \mathbf{b}$, we get

  $$M\mathbf{x} = N\mathbf{x} + \mathbf{b} \tag{2.1}$$

- this inspires the iteration

  $$M\mathbf{x}^{(k+1)} = N\mathbf{x}^{(k)} + \mathbf{b} \tag{2.2}$$

- subtracting (2.2) from (2.1), we obtain

  $$M(\mathbf{x} - \mathbf{x}^{(k+1)}) = N(\mathbf{x} - \mathbf{x}^{(k)})$$

- if we denote the *error* in $\mathbf{x}^{(k)}$ by $\mathbf{e}^{(k)} = \mathbf{x} - \mathbf{x}^{(k)}$, then

  $$\mathbf{e}^{(k+1)} = M^{-1}N\mathbf{e}^{(k)} =: B\mathbf{e}^{(k)}$$

- thus $\mathbf{e}^{(k)} = B\mathbf{e}^{(k)} = B^{k+1}\mathbf{e}^{(0)}$
- note that

  $$\mathbf{x}^{(k)} \to \mathbf{x} \quad \text{if and only if} \quad \mathbf{e}^{(k)} \to \mathbf{0} \quad \text{if and only if} \quad \|\mathbf{e}^{(k)}\| \to 0$$

- the matrix $B = M^{-1}N$ is somtimes called the *iteration matrix*
- its spectral radius $\rho(B)$ governs convergence rate, i.e., how quickly the error goes to zero
- recall that if $\rho(B^k) < 1$ then $\mathbf{e}^{(k)} \to \mathbf{0}$ for all choices of $\mathbf{x}^{(0)}$
- we have the following theorem:

Theorem 1. *$\mathbf{e}^{(k)} \to \mathbf{0}$ as $k \to \infty$ for all $\mathbf{e}^{(0)}$ if and only if $\rho(B) < 1$.*

Proof. Note that $\mathbf{e}^{(k)} = B^{k+1}\mathbf{e}^{(0)} \to \mathbf{0}$ for all $\mathbf{e}^{(0)}$ is equivalent to $\lim_{k\to\infty} B^k = O$ (the zero matrix) since we could choose $\mathbf{e}^{(0)}$ to be each of the standard basis vectors $\mathbf{e}_1, \dots, \mathbf{e}_n$ in turn and so we get

$$B^k = B^k I = B^k[\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_n] = [B^k\mathbf{e}_1, B^k\mathbf{e}_2, \dots, B^k\mathbf{e}_n] \to [\mathbf{0}, \mathbf{0}, \dots, \mathbf{0}] = O$$

as $k \to \infty$. Now by what we discussed in an earlier lecture (about the Jordan form), for a Jordan block,

$$J_r^k = \begin{bmatrix} \lambda_r^k & \binom{k}{1}\lambda_r^{k-1} & \binom{k}{2}\lambda_r^{k-2} & \cdots & \binom{k}{n_r-1}\lambda_r^{k-(n_r-1)} \\ & \ddots & \ddots & & \vdots \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & \vdots \\ & & & & \lambda_r^k \end{bmatrix} \to O$$

as $k \to \infty$. Since B has a Jordan decomposition,

$$B = X \begin{bmatrix} J_1 & & \\ & \ddots & \\ & & J_m \end{bmatrix} X^{-1},$$

we have

$$B^k = X \begin{bmatrix} J_1^k & & \\ & \ddots & \\ & & J_m^k \end{bmatrix} X^{-1} \to X \begin{bmatrix} O & & \\ & \ddots & \\ & & O \end{bmatrix} X^{-1} = O$$

as $k \to \infty$. ☐

- convergence can still occur if $\rho(B) = 1$, but in that case we must be careful in how we choose $\mathbf{x}^{(0)}$
- recall also that for all consistent norms,
$$\rho(B) \le \|B\|$$
and
$$\|B^k\| \le \|B\|^k$$
- from $\mathbf{e}^{(k)} = B^k\mathbf{e}^{(0)}$, it follows that
$$\frac{\|\mathbf{e}^{(k)}\|}{\|\mathbf{e}^{(0)}\|} \le \|B\|^k$$
- so if we find a consistent norm with $\|B\| < 1$, then this gives a sufficient condition for convergence
- note that convergence does not depend on the choice of norms since on finite-dimensional spaces, all norms are equivalent
- if we can prove statements like $\|B^k\| \to 0$ or $\|\mathbf{e}^{(k)}\| \to 0$ for any one norm, we know that it will hold for all norms

3. CONVERGENCE RATE

- formally, for a sequence $\mathbf{x}_k$ that converges to $\mathbf{x}$, its *convergence rate* $r \in (0,1)$ is defined to be
$$r = \limsup_{k\to\infty} \frac{\|\mathbf{e}^{(k+1)}\|}{\|\mathbf{e}^{(k)}\|} = \limsup_{k\to\infty} \frac{\|\mathbf{x}^{(k+1)} - \mathbf{x}\|}{\|\mathbf{x}^{(k)} - \mathbf{x}\|}$$
or alternatively, the smallest $r \in (0,1)$ such that
$$\|\mathbf{e}^{(k+1)}\| \le r\|\mathbf{e}^{(k)}\| \qquad \text{for all } k \text{ sufficiently large}$$

- a sequence that has such a property is called *linearly convergent* and we will often say that an iterative algorithm is linearly convergent for a class of problem if it generates a linearly convergent sequence for all choices of initial points $\mathbf{x}^{(0)}$
- if
$$\limsup_{k\to\infty} \frac{\|\mathbf{e}^{(k+1)}\|}{\|\mathbf{e}^{(k)}\|} = 0,$$
we say that the sequence (resp. algorithm) is *superlinearly convergent*
- if there exists $M > 0$ such that
$$\|\mathbf{e}^{(k+1)}\| \le M\|\mathbf{e}^{(k)}\|^2 \qquad \text{for all } k \text{ sufficiently large,}$$
we say that the sequence (resp. algorithm) is *quadratically convergent*
- note that M does not need to be in $(0,1)$
- more generally the largest p for which there exists $M > 0$ such that
$$\|\mathbf{e}^{(k+1)}\| \le M\|\mathbf{e}^{(k)}\|^p \qquad \text{for all } k \text{ sufficiently large,}$$
is called the *order of convergence*

4. Jacobi method

- the simplest splitting is to take M to be the diagonal part of A and $-N$ to be the off-diagonal part — this works as long as the diagonal elements of A is nonzero (but the iterates may not converge)
- if we write $A\mathbf{x} = \mathbf{b}$ in coordinate form,
$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i = 1,\dots,n,$$
then
$$a_{ii}x_i = b_i - \sum_{i\ne j} a_{ij}x_j,$$
or
$$x_i = \frac{1}{a_{ii}}\Bigl[b_i - \sum_{j\ne i} a_{ij}x_j\Bigr] \tag{4.1}$$
- in other words,
$$M = \begin{bmatrix} a_{11} & & & \\ & \ddots & & \\ & & \ddots & \\ & & & a_{nn} \end{bmatrix}, \quad N = -\begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{n1} & \cdots & a_{n,n-1} & 0 \end{bmatrix}$$
- our iteration is therefore
$$x_i^{(k+1)} = \frac{1}{a_{ii}}\Bigl[b_i - \sum_{j\ne i} a_{ij}x_j^{(k)}\Bigr],$$
known as the *Jacobi method*
- if we write $A = L + D + U$ where
$$L = \begin{bmatrix} 0 & & & \\ a_{21} & \ddots & & \\ \vdots & & \ddots & \\ a_{n1} & \cdots & a_{n,n-1} & 0 \end{bmatrix}, \quad D = \begin{bmatrix} a_{11} & & & \\ & \ddots & & \\ & & \ddots & \\ & & & a_{nn} \end{bmatrix}, \quad U = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ & \ddots & & \vdots \\ & & \ddots & a_{n-1,n} \\ & & & 0 \end{bmatrix}$$

the the Jacobi mathod can be written in matrix form as

$$D\mathbf{x}^{(k+1)} = -(L+U)\mathbf{x}^{(k)} + \mathbf{b} \tag{4.2}$$

- the iteration matrix is

$$M^{-1}N = I - D^{-1}A = -\begin{bmatrix} 0 & \frac{a_{12}}{a_{11}} & \cdots & \frac{a_{1n}}{a_{11}} \\ \frac{a_{21}}{a_{22}} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \frac{a_{n1}}{a_{nn}} & \cdots & \frac{a_{n,n-1}}{a_{nn}} & 0 \end{bmatrix} =: B_{\mathrm{J}}$$

- so if

$$\|B_{\mathrm{J}}\|_\infty = \max_{1\le i\le n} \sum_{j\ne i} \left|\frac{a_{ij}}{a_{ii}}\right| < 1,$$

i.e., if A is *strictly diagonally dominant*, then the iteration converges
- therefore, a sufficient condition for convergence of the Jacobi method is $\|B_{\mathrm{J}}\|_\infty < 1$ where

$$b_{ij} = \begin{cases} -\dfrac{a_{ij}}{a_{ii}} & i \ne j, \\ 0 & i = j \end{cases}$$

- for example, suppose

$$A = \begin{bmatrix} 4 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4 \end{bmatrix},$$

then $\|B_{\mathrm{J}}\|_\infty = \frac{1}{2}$ and so the Jacobi method converges rapidly
- on the other hand, if

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

which arises from discretizing the one-dimensional Laplacian, then $\|B_{\mathrm{J}}\|_\infty = 1$
- a more subtle analysis can be used to show convergence in this case, but convergence is slow

5. Gauss–Seidel method

- in the Jacobi method, we compute $x_i^{(k+1)}$ using the elements of $\mathbf{x}^{(k)}$, even though $x_1^{(k+1)}, \dots, x_{i-1}^{(k+1)}$ are already known

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

- a general adage in numerical computations is: *use the latest information available*
- the *Gauss–Seidel* method is designed to take advantage of the latest information available about $\mathbf{x}$:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right] \tag{5.1}$$

- if we write $A = L + D + U$ where

$$L = \begin{bmatrix} 0 & & & \\ a_{21} & \ddots & & \\ \vdots & & \ddots & \\ a_{n1} & \cdots & a_{n,n-1} & 0 \end{bmatrix}, \quad D = \begin{bmatrix} a_{11} & & & \\ & \ddots & & \\ & & \ddots & \\ & & & a_{nn} \end{bmatrix}, \quad U = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ & \ddots & & \vdots \\ & & \ddots & a_{n-1,n} \\ & & & 0 \end{bmatrix}$$

  then the Gauss–Seidel iteration can be written in matrix form as

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

  or

$$(D + L)\mathbf{x}^{(k+1)} = -U\mathbf{x}^{(k)} + \mathbf{b} \tag{5.2}$$

  which yields

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

- thus the iteration matrix for the Gauss–Seidel method is

$$B_{\mathsf{GS}} = -(D + L)^{-1}U$$

  as opposed to the iteration matrix for the Jacobi method

$$B_{\mathsf{J}} = -D^{-1}(L + U)$$

- in some cases (cf. last line of the section on optimal SOR parameter in the next lecture)

$$\rho(B_{\mathsf{GS}}) = \rho(B_{\mathsf{J}})^2$$

  so the Gauss–Seidel method converges twice as fast
- on the other hand, note that Gauss–Seidel is very sequential, i.e., it does not lend itself to parallelism
- note that the matrix forms for Jacobi and Gauss–Seidel (4.2) and (5.2) are only convenient representations useful in mathematical analysis of the methods, one should *never* implement these algorithms in such forms, instead use (4.1) and (5.1)
- we saw earlier that a sufficient condition for convergence of the Jacobi method is $\|B_{\mathsf{J}}\|_\infty < 1$ where

$$b_{ij} = \begin{cases} -\dfrac{a_{ij}}{a_{ii}} & i \neq j, \\ 0 & i = j \end{cases}$$

- since

$$\|B_{\mathsf{J}}\|_\infty = \max_i \sum_{j \neq i} \left| \frac{a_{ij}}{a_{ii}} \right| < 1,$$

  this is equivalent to saying that A is strictly diagonally dominant
- we will see that this is also enough to guarantee the convergence of Gauss–Seidel, i.e., if A is strictly diagonally dominant, then Gauss–Seidel is convergent
- define

$$r_i = \sum_{j \neq i} \left| \frac{a_{ij}}{a_{ii}} \right|, \quad r = \max_i r_i$$

Theorem 2. *If $r < 1$, then $\rho(B_{\mathsf{GS}}) < 1$, i.e., the Gauss–Seidel iteration converges if A is strictly diagonally dominant.*

Proof. The proof proceeds using induction on the elements of $\mathbf{e}^{(k)}$. We have

$$(D+L)\mathbf{e}^{(k+1)} = -U\mathbf{e}^{(k)},$$

which can be written as

$$\sum_{j=1}^{i} a_{ij}e_j^{(k+1)} = -\sum_{j=i+1}^{n} a_{ij}e_j^{(k)}, \quad i = 1,\dots,n.$$

Thus

$$e_i^{(k+1)} = -\sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}}e_j^{(k)} - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}}e_j^{(k+1)}, \quad i = 1,\dots,n.$$

For $i = 1$, we have

$$|e_1^{(k+1)}| \le \sum_{j=2}^{n} \left|\frac{a_{ij}}{a_{11}}\right| |e_j^{(k)}| \le r_1\|\mathbf{e}^{(k)}\|_\infty \le r\|\mathbf{e}^{(k)}\|_\infty.$$

Assume that for $p = 1,\dots,i-1$,

$$|e_p^{(k+1)}| \le \|\mathbf{e}^{(k)}\|_\infty r_p \le r\|\mathbf{e}^{(k)}\|_\infty.$$

Then,

$$\begin{aligned}
|e_i^{(k+1)}| &\le \sum_{j=1}^{i-1} \left|\frac{a_{ij}}{a_{ii}}\right| |e_j^{(k+1)}| + \sum_{j=i+1}^{n} \left|\frac{a_{ij}}{a_{ii}}\right| |e_j^{(k)}| \\
&\le r\|\mathbf{e}^{(k)}\|_\infty \sum_{j=1}^{i-1} \left|\frac{a_{ij}}{a_{ii}}\right| + \|\mathbf{e}^{(k)}\|_\infty \sum_{j=i+1}^{n} \left|\frac{a_{ij}}{a_{ii}}\right| \\
&\le \|\mathbf{e}^{(k)}\|_\infty \sum_{j\ne i} \left|\frac{a_{ij}}{a_{ii}}\right| \\
&= r_i\|\mathbf{e}^{(k)}\|_\infty \\
&\le r\|\mathbf{e}^{(k)}\|_\infty.
\end{aligned}$$

Therefore

$$\|\mathbf{e}^{(k+1)}\|_\infty \le r\|\mathbf{e}^{(k)}\|_\infty \le r^{k+1}\|\mathbf{e}^{(0)}\|_\infty,$$

from which it follows that

$$\lim_{k\to\infty} \|\mathbf{e}^{(k)}\|_\infty = 0$$

since $r < 1$. □

- while both the Jacobi method and the Gauss–Seidel method both converge if A is diagonally dominant, convergence can be slow in some cases
- for example, for

$$A = \begin{bmatrix} 2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{n\times n}$$

we have

$$-D^{-1}(L+U) = \begin{bmatrix} 0 & 1/2 & & \\ 1/2 & \ddots & \ddots & \\ & \ddots & \ddots & 1/2 \\ & & 1/2 & 0 \end{bmatrix}$$

and therefore

$$\rho(B_{\mathsf{J}}) = \cos\frac{\pi}{n+1} = \cos\pi h \approx 1 - \frac{\pi^2 h^2}{2} + \cdots$$

which is approximately 1 for small $h = 1/(n+1)$

- suppose $B_{\mathsf{J}} = B_{\mathsf{J}}^{\mathsf{T}}$, then

$$\frac{\|\mathbf{e}^{(k)}\|_2}{\|\mathbf{e}^{(0)}\|_2} \le \|B_{\mathsf{J}}\|_2^k = \rho(B_{\mathsf{J}})^k$$

- if we want $\|\mathbf{e}^{(k)}\|_2/\|\mathbf{e}^{(0)}\|_2 \le \varepsilon$, then setting $\rho^k = \varepsilon$, we get that

$$k = \frac{-\log\varepsilon}{-\log\rho}$$

is the number of iterations necessary for convergence
- so $\rho = \rho(A)$ controls the rate of convergence

6. SOR METHOD

- reminder: in coordinatewise form, Jacobi method is

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right] \tag{6.1}$$

and Gauss–Seidel method is

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right] \tag{6.2}$$

- reminder: in matrix form, Jacobi method is

$$D\mathbf{x}^{(k+1)} = \mathbf{b} - (L+U)\mathbf{x}^{(k)} \tag{6.3}$$

and Gauss–Seidel is

$$(D+L)\mathbf{x}^{(k+1)} = \mathbf{b} - U\mathbf{x}^{(k)} \tag{6.4}$$

- another general adage in numerical computations is: *don't discard previous information, try to use it too*
- applying this to Gauss–Seidel, we could try to use both $x_j^{(k+1)}$ and $x_j^{(k)}$ for $j = 1, \dots, i-1$ to obtain $x_i^{(k+1)}$ — this yields the method of *successive over relaxation* (SOR)
- this is given by the iteration

$$x_i^{(k+1)} = \frac{\omega}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right] + (1-\omega)x_i^{(k)} \tag{6.5}$$

- the quantity ω is called the *relaxation parameter*
- if $\omega = 1$, then the SOR method reduces to the Gauss–Seidel method, i.e.,

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

- the name 'over relaxation' comes from choosing $\omega > 1$
- in matrix form, the iteration can be written as

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

which can be rearranged to obtain

$$(D+\omega L)\mathbf{x}^{(k+1)} = \omega\mathbf{b} + [(1-\omega)D - \omega U]\mathbf{x}^{(k)} \tag{6.6}$$

or

$$\mathbf{x}^{(k+1)} = \left(\frac{1}{\omega}D + L\right)^{-1}\left[\left(\frac{1}{\omega} - 1\right)D - U\right]\mathbf{x}^{(k)} + \left(\frac{1}{\omega}D + L\right)^{-1}\mathbf{b} \tag{6.7}$$

- the iteration matrix is

$$B_\omega = \left(\frac{1}{\omega}D + L\right)^{-1}\left[\left(\frac{1}{\omega} - 1\right)D - U\right]$$

- since $B_1 = B_{\mathsf{GS}}$, if we pick some $\omega \neq 1$ such that

$$\rho(B_\omega) < \rho(B_{\mathsf{GS}}),$$

  we would improve the convergence of Gauss–Seidel
- so SOR is at least as fast as Gauss–Seidel and often faster
- in fact, for certain types of matrices, one can pick ω so that $\rho(B_\omega)$ is minimized
- for example, if A is (i) a nonsingular matrix, (ii) its Jacobi iteration matrix B_{J} has only real eigenvalues, and (iii) A may be permuted into the form

$$A = \Pi_1 \begin{bmatrix} D_1 & B_{12} \\ B_{21} & D_2 \end{bmatrix} \Pi_2$$

  where Π_1, Π_2 are permutation matrices and D_1, D_2 are diagonal matrices, then the optimal elaxation parameter is given by

$$\omega_{\text{opt}} = \frac{2}{1 + \sqrt{1 + \rho(B_{\mathsf{J}})^2}}$$

  and

$$\rho(B_{\omega_{\text{opt}}}) = \frac{1 + \sqrt{1 - \rho(B_{\mathsf{J}})^2}}{1 + \sqrt{1 + \rho(B_{\mathsf{J}})^2}}$$

- note that if $A\mathbf{x} = \mathbf{b}$, then

$$D\mathbf{x} = \omega(\mathbf{b} - L\mathbf{x} - U\mathbf{x}) + (1 - \omega)D\mathbf{x}$$

  and so

$$\mathbf{x} = \left(\frac{1}{\omega}D + L\right)^{-1}\left[\left(\frac{1}{\omega} - 1\right)D - U\right]\mathbf{x}^* + \left(\frac{1}{\omega}D + L\right)^{-1}\mathbf{b} \tag{6.8}$$

- subtracting (6.8) from (6.7), we get

$$\mathbf{e}^{(k+1)} = B_\omega \mathbf{e}^{(k)}$$

- note that

$$\begin{aligned}
\det B_\omega &= \det\left(\frac{1}{\omega}D + L\right)^{-1} \det\left[\left(\frac{1}{\omega} - 1\right)D - U\right] \\
&= \frac{1}{\det\left(\frac{1}{\omega}D + L\right)} \det\left[\left(\frac{1}{\omega} - 1\right)D - U\right] \\
&= \frac{\omega^n}{\prod_{i=1}^n a_{ii}} \frac{(1-\omega)^n \prod_{i=1}^n a_{ii}}{\omega^n} \\
&= (1 - \omega)^n
\end{aligned}$$

- therefore $\prod_{i=1}^n \lambda_i = (1 - \omega)^n$ where $\lambda_1, \dots, \lambda_n$ are the eigenvalues of B_ω, with $|\lambda_1| \geq \cdots \geq |\lambda_n|$
- hence we get

$$|\lambda_1|^n \geq (1 - \omega)^n$$

- since we must also have
$$|\lambda_1| = \rho(B_\omega) < 1$$
for convergence
- it follows that a necessary condition for convergence of SOR is
$$0 < \omega < 2$$
- if A is symmetric positive definite, then the condition $0 < \omega < 2$ is also sufficient — a result of Ostrowski implies that for such an A, $\rho(B_\omega) < 1$ iff $0 < \omega < 2$
- suppose $A \in \mathbb{R}^{n\times n}$ is a symmetric matrix, then $U = L^{\mathsf{T}}$ and if we set
$$M_\omega = \frac{\omega}{2-\omega}\left(\frac{1}{\omega}D + L\right) D^{-1} \left(\frac{1}{\omega}D + L^{\mathsf{T}}\right)$$
and define our iteration as
$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k-1)} - M_\omega^{-1}(A\mathbf{x}^{(k)} - \mathbf{b})$$
- this is called the method of *symmetric successive over relaxation* (SSOR)
- there are yet other variants of SOR such as:
  - block SOR or block SSOR when the matrix has a block structure; likewise, we may also introduce block Jacobi or block Gauss–Seidel
  - applying the SOR extrapolation to Jacobi method to get
$$x_i^{(k+1)} = \frac{\omega}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right] + (1-\omega)x_i^{(k)} \tag{6.9}$$
to preserve parallelism; this is often called JOR
  - note the difference between (6.9) and (6.5) and note that when $\omega = 1$ in (6.9), then the JOR method reduces to Jacobi method
  - one may even define a nonlinear version of SOR for iterations of the form $\mathbf{x}^{(k+1)} = f(\mathbf{x}^{(k)})$ where f is some nonlinear function $f : \mathbb{R}^n \to \mathbb{R}^n$:
$$\mathbf{x}_{\text{SOR}}^{(k+1)} = (1-\omega)\mathbf{x}_{\text{SOR}}^{(k)} + \omega f(\mathbf{x}_{\text{SOR}}^{(k)})$$
- iterative methods are intended for situations when n is so big that you can't fit an $n \times n$ matrix in memory — so direct methods based on matrix decompositions like Cholesky, LU, QR, SVD, EVD, etc are prohibitively expensive
- so the whole point of iterative methods like Jacobi, Gauss–Seidel, SOR is to store only vectors and update iterates $\mathbf{x}^{(k)}$ coordinatewise, i.e., in the form (6.1), (6.2), (6.5)
- the matrix forms (6.3), (6.4), (6.6) are only intended for mathematical discussions

anyone who implements Jacobi, Gauss–Seidel, SOR in the matrix form fails this class instantly

7. Final words: linear algebra in the wild

- looking back at what we have covered in this course, in some sense, all we are talking about is just solution of a linear system
$$A\mathbf{x} = \mathbf{b}$$
- but unlike in pure mathematics where we speak of problems in an idealized manner, we discuss $A\mathbf{x} = \mathbf{b}$ in various realistic situations and under real-world constraints:
  - what happens if there is error in $\mathbf{b}$, in A, in both?
  - what happens if A is near to a singular matrix?
  - what happens if A is singular or rectangular?
  - what happens if A is so big that we can't store it in main memory?

  - what happens if we make a mistake in a column, a row, or an entry of A?
  - what happens if we don't actually have the matrix A, only a blackbox that gives us $A\mathbf{x}$ for any input $\mathbf{x}$?
  - what happens if we have multiple $\mathbf{b}$'s?
  - what happens if we only need the solution $\mathbf{x}$ to a certain degree of accuracy?
  - what happens if we need the norm of the solution $\mathbf{x}$ to be bounded to a certain size?
- in order to deal with these scenarios, we need to know about
  - notions like norms, condition number, numerical rank, etc
  - problems like ordinary and total least squares, linearly and norm-constrained least squares, etc
  - tools like LU,QR, SVD, EVD, secular equations, Eckart–Young, Sherman–Morrison, etc
  - algorithms like GEPP, Householder QR, SOR, conjugate gradient, etc
- the problems depend a lot on the computing technology used: in this course we have assumed conventional computers with CPUs and FPUs implementing the IEEE floating point standard
- there many other situations and technologies that we did not address in this course:
  - what if we want to optimize our algorithms for a multilevel cache structure or a multicore structure?
  - what if we want to run our algorithms in a distributed manner over a cloud of servers?
  - what if we want to exploit other hardware like GPUs, FPGAs, etc?
  - what if we have a quantum computer?
  - what if we could exploit randomness?
  - what if A is given to us a column or a row at a time?
  - what if we need a sparse solution $\mathbf{x}$ with only a fixed number of nonzero entries?
  - what if we need a sparse solution $\mathbf{x}$ with as few nonzero entries as possible?