Math 471 (Numerical methods)

Chapter 3 (second half). System of equations

Overlap §3.5–3.8 of Bradie

§3.5 LU factorization w/o pivoting.

Motivation: $\Big(A \,|\, I\Big) \rightarrow$ *Gaussian Elimination* $\rightarrow \Big(U \,|\, L_1\Big)$ where U is upper triangular and L_1 is lower triangular. Then, the entire Gaussian elimination process amounts to multiplying the augmented matrix from the left with L_1. Thus

$$L_1 A = U \implies A = L_1^{-1} U =: LU$$

Why LU? (for less operation count but not for stability)

- When used for solving linear system $A\vec{x} = \vec{b}$, i.e.

$$LU\vec{x} = \vec{b} \Leftrightarrow \begin{cases} L\vec{y} = \vec{b} & \text{solved with forward substitution} \\ U\vec{x} = \vec{y} & \text{solved with backward substitution} \end{cases},$$

  it helps reducing the operation count from $O(n^3)$ in the original Gaussian Elimination to $O(n^2)$ in the forward/backward substitution for solving lower/upper triangular systems.

- Also, once LU factorization is done, it can be repeatedly used for solving multiple linear systems with the same coefficient matrix but different right-hand-side vectors.

- Note that LU factorization itself, in the most general case, requires $O(n^3)$ operations. The reason is exactly the same as for the op. count of Guassian Elimination.

In details,

each row operation performed on the coefficient matrix

amounts to

a left-multiplication of an elementary matrix.

Moreover, each one of these matrices (except those representing row-swapping) is lower triangular, and therefore their product is also lower triangular — we showed in class that product of lower triangular matrices is still lower triangular.

Theorem 1 *If we perform a row operation "add $m\times$ [row i] to [row j]" on A and arrive at A_1, then*

$$MA = A_1, \text{ where } M = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & m & \ddots & \\ & & & 1 \end{pmatrix}.$$

Here, the elementary matrix M is an identity matrix superposed with an entry m at the j-th row and i-th column.

Similarly,

Theorem 2 *If we perform a column operation "add $m\times$ [column i] to [column j]" on A and arrive at A_2, then*

$$AM = A_2, \text{ where } M = \begin{pmatrix} 1 & & & \\ & 1 & m & \\ & & \ddots & \\ & & & 1 \end{pmatrix}.$$

Here, the elementary matrix M is an identity matrix superposed with an entry m at the i-th row and j-th column.

In principle

- If a matrix M represents a row/column operation, it is obtained by performing the same operation on the identity matrix. One should **left-**multiply M with the target matrix if M stands for row operation and **right-**multiply M with the target matrix if M stands for column operation.

********** The part below is optional but can be helpful **********

Now, we make an observation. On the j-th step of the Gaussian elimination of zeroing out the lower part of the j-th column, multiple row operations are performed which can be represented by a product $N_m...N_2N_1$. If we perform this same sequence of row operations on I, each N_k corresponding to adding a rescaled [row j] to some row below. Therefore,

the 1's on the diagonal will be kept and each entry in the lower part of the j-column will represent the factors used in each rescaling of [row j],

$$N_m...N_2N_1 = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & m_{j+1,j} & \ddots & \\ & \vdots & & \ddots \\ & m_{n,j} & & 1 \end{pmatrix}.$$

So, we use exactly n lower triangular matrices $M_1, ..., M_n$. Each M_j represents the operations collectively performed for zeroing out the j-th column of A,

$$M_n...M_2M_1A = U \tag{1}$$

which suggests an algorithm to find L. In practice, one does a bookkeeping of the Gaussian elimination – a sequence of row operations – and stores the information in the M's. The final product $L = M_1^{-1}M_2^{-1}...M_{n-1}^{-1}$ in (1) can be easily computed thanks to the following facts

$$\begin{pmatrix} 1 & & & \\ & 1 & & \\ & * & \ddots & \\ & * & & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & -* & \ddots & \\ & -* & & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & & & \\ @ & 1 & & \\ @ & & \ddots & \\ @ & & & 1 \end{pmatrix} \begin{pmatrix} 1 & & & \\ & 1 & & \\ & * & \ddots & \\ & * & & 1 \end{pmatrix} \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & \& & 1 \end{pmatrix} = \begin{pmatrix} 1 & & & \\ @ & 1 & & \\ @ & * & \ddots & \\ @ & * & \& & 1 \end{pmatrix}$$

The proofs of these facts are skipped. But note that the last equation holds true for $M_iM_jM_k$ only when $i < j < k$.

**********end of optional part**********

• A final remark. From the above discussion, we see $L_1 = L^{-1}$ is a by-product obtained as an intermediate result from Guassian Elimination. Why not just use L^{-1} directly? The answer is yes **if** both L and L^{-1} are "density matrix" with $O(n^2)$ nonzero entries. In such case, solving $L\vec{x} = \vec{b}$

1. with forward substitution; or

2. using $\vec{x} = L^{-1}\vec{b}$

both cost $O(n^2)$ operations. In other words, there is no siginificant difference in using L and L^{-1} in terms of efficiency.

However, when A is a **sparse matrix** with m nonzero entries (m is much less than n^2), it's LU decomposition will very likely preserve such sparsity pattern, which brings the operation count for solving $L\vec{x} = \vec{b}$ down to $O(m)$. On the other hand, L^{-1} may be a dense matrix with $O(n^2)$ nonzero entries, which means calculation of $L^{-1}\vec{b}$ requires the same $O(n^2)$ operations, far more than $O(m)$.

In Section 3.6 "Direct Factorization", we will see an alternative approach that obtains L, U without using Guassian Elimination. There, the case of sparse matrices will be discussed in details.

§3.5 (cont'd) LU factorization with pivoting.

LU decomposition, being equivalent to the Gaussian elimination, has the same problem of having zero (or very small) pivot entries. Pivoting is important in some cases. So row interchange needs to be added to the LU algorithm.

Using the same matrix language, row interchange is characterized with permutation matrix:

$$A \to \text{swapping row i and row j} \to A_1 \qquad \text{amounts to} \qquad PA = A_1$$

where permutation matrix

$$P = \begin{pmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & 0 & & 1 & & \\ & & & \ddots & & & \\ & & 1 & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}$$

is a matrix resulting from swapping the i-th and j-th row of the identity matrix.

Similarly, with the same P given above,

$$A \to \text{swapping column i and column j} \to A_2 \qquad \text{amounts to} \qquad AP = A_2$$

Thus, Gaussian elimination with pivoting can be described with addition of permutation matrices.

$$A = (M_{n-1}P_{n-1}...M_2P_2M_1P_1)^{-1}U \tag{2}$$

Problem! Row interchange destroys the lower triangular pattern in M so using (2) directly will not yield a lower triangular matrix L.

Remedy. The following properties of permutation matrix are useful.

$$P^2 = I,$$

$$P_iM_j = \hat{M}_jP_i \quad \text{for} \quad i > j.$$

(The proofs are discussed in class.)

The key point here is to change the order of P's and M's in the LHS of (2) so that it becomes

$$A = \left[(\hat{M}_{n-1}...\hat{M}_2\hat{M}_1)(P_{n-1}...P_2P_1)\right]^{-1} U$$

and therefore

$$PA = LU.$$

This version of the LU decomposition, as efficient as the original LU, is more stable because of pivoting.

Note. Now, solving $A\vec{x} = \vec{b}$ amounts to solving $PA\vec{x} = P\vec{b}$, i.e. $LU\vec{x} = P\vec{b}$. The additional calculation of $P\vec{b}$ is just permutation of entries in $\vec{b}$, which costs $O(n)$ operations.

§3.6 Direct Factorization

There is a way to find LU factorization if we completely forget about Guassian Elimination. It gets heuristic of the most elementary idea: why not solve $A = LU$ by treating it as n^2 equations?

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & ... & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & ... & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & ... & a_{3,n} \\ ... & ... & ... & ... & ... \\ a_{n,1} & a_{n,2} & a_{n,3} & ... & a_{n,n} \end{pmatrix} = \begin{pmatrix} l_{1,1} & 0 & 0 & ... & 0 \\ l_{2,1} & l_{2,2} & 0 & ... & 0 \\ l_{3,1} & l_{3,2} & l_{3,3} & ... & 0 \\ ... & ... & ... & ... & ... \\ l_{n,1} & l_{n,2} & l_{n,3} & ... & l_{n,n} \end{pmatrix} \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} & ... & u_{1,n} \\ 0 & u_{2,2} & u_{2,3} & ... & u_{2,n} \\ 0 & 0 & u_{3,3} & ... & u_{3,n} \\ ... & ... & ... & ... & ... \\ 0 & 0 & 0 & ... & u_{n,n} \end{pmatrix}$$

Each entry $a_{i,j}$ gives an equation in terms of the unknown l's and u's

$$a_{i,j} = \text{[row i] of } L * \text{[column j] of } U$$

that is

$$a_{i,j} = l_{i,1}u_{1,j} + l_{i,2}u_{2,j} + ... + l_{i,n}u_{n,j} = \sum_{k=1}^{n} l_{i,k}u_{k,j}. \tag{3}$$

Notice the running index k.

However, solving these n^2 equations in a most straightforward way, i.e. with Guassian Elimination, will cost $O(n^2)^3$ operations which far too expensive. We have to take advantage of the lower and upper triangular structure of L and U, and come up with a smarter algorithm.

First, let's fix the diagonal entries of U as 1. One can also fix the diagonal entries of L as 1. The former called the Crout method and the latter the Doolittle method. But here, we adopt

$$u_{i,i} = 1. \tag{4}$$

Then, we scan thought A row-by-row, writing down $a_{i,j}$ in terms of [row i] of L and [row j] of U while taking into account the zeros of L and U. It turns out,

Scanning of [row i] of A yields values of the same row in L and U.

This is obvious for [row 1].

- Equation with $a_{1,1}$,

  $$a_{1,1} = l_{1,1}u_{1,1}$$

  gives $l_{1,1}$ due to (4).

- Equation with $a_{1,j}$ when $j > 1$

  $$a_{1,j} = l_{1,1}u_{1,j}$$

  only has 1 term on the right hand side due to the lower triangular structure of L. Since $l_{1,1}$ is solved from above, one easily solves for $u_{i,j}$.

In a general step involving [row i], we can still follow the above two-part procedure, the first part finding $l_{i,j}$ with $j \leq i$ and the second part finding $u_{i,j}$ with $j > i$. This is best described in a for loop

%Scanning of [row i]. Here, we should have already obtained values of
%[row 1] ... [row j-1] in L and U
For j=1:n %Each iteration uses $a_{i,j}$ to find either $l_{i,j}$ or $u_{i,j}$
%Here, we should have already obtained values of $l_{i,1}$ ... $l_{i,j-1}$
%and $u_{i,1}$ ... $u_{i,j-1}$ some of which are simply zero

- *If $i \geq j$, the sum in (3) stops at $l_{i,j}u_{j,j}$*

$$a_{i,j} = l_{i,1}u_{1,j} + l_{i,2}u_{2,j} + ... + l_{i,j}u_{j,j}$$

but every term in this equation, except $l_{i,j}$ has already been obtained
(see the above comments about the availability of the l's and u's, also in below)

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & ... & a_{1,n} \\ ... & ... & ... & ... & ... \\ a_{i,1} & ... & a_{i,j} & ... & a_{3,n} \\ ... & ... & ... & ... & ... \\ a_{n,1} & a_{n,2} & a_{n,3} & ... & a_{n,n} \end{pmatrix} = \begin{pmatrix} l_{1,1} & ... & ... & ... & 0 \\ ... & ... & ... & ... & 0 \\ l_{i,1} & ... & l_{i,j} & ... & 0 \\ ... & ... & ... & ... & ... \\ l_{n,1} & l_{n,2} & l_{n,3} & ... & l_{n,n} \end{pmatrix} \begin{pmatrix} ... & ... & ... & ... & u_{1,n} \\ ... & u_{j,j} = 1 & ... & ... & ... \\ 0 & ... & u_{i,i} = 1 & ... & u_{i,n} \\ ... & ... & ... & ... & ... \\ 0 & 0 & 0 & ... & u_{n,n} \end{pmatrix}$$

where green color indicates entry value is available
and red indicates the entry we can solve. Therefore

$$l_{i,j} = \frac{1}{u_{j,j}} \left[a_{i,j} - (l_{i,1}u_{1,j} + ... + l_{i,j-1}u_{j-1,j}) \right]. \tag{5}$$

- *If $i < j$, the sum in (3) stops at $l_{i,i}u_{i,j}$*

$$a_{i,j} = l_{i,1}u_{1,j} + l_{i,2}u_{2,j} + ... + l_{i,i}u_{i,j}$$

but every term in this equation, except $u_{i,j}$ has already been obtained

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & ... & a_{1,n} \\ ... & ... & ... & ... & ... \\ a_{i,1} & ... & a_{i,j} & ... & a_{3,n} \\ ... & ... & ... & ... & ... \\ a_{n,1} & a_{n,2} & a_{n,3} & ... & a_{n,n} \end{pmatrix} = \begin{pmatrix} l_{1,1} & ... & ... & ... & 0 \\ ... & ... & ... & ... & 0 \\ l_{i,1} & ... & l_{i,i} & ... & 0 \\ ... & ... & ... & ... & ... \\ l_{n,1} & l_{n,2} & l_{n,3} & ... & l_{n,n} \end{pmatrix} \begin{pmatrix} ... & ... & ... & ... & u_{1,n} \\ ... & ... & ... & ... & ... \\ 0 & ... & u_{i,j} & ... & u_{3,n} \\ ... & ... & ... & ... & ... \\ 0 & 0 & 0 & ... & u_{n,n} \end{pmatrix}$$

Therefore,

$$u_{i,j} = \frac{1}{l_{i,i}} \left[a_{i,j} - (l_{i,1}u_{1,j} + ... + l_{i,i-1}u_{i-1,j}) \right] \tag{6}$$

end of for j=1:n

So, the final algorithm for the Crout method is just to combine (5) and (6) into another loop

For i=1:n

insert (5) *and* (6) *here*

end of for i=1:n

What about operation count in the above algorithm. Notice that we need to solve for n^2 entries of $l_{i,j}$ or $u_{i,j}$, depending on $i \geq j$ or $i < j$. Each solving uses either (5) or (6), which obviously costs $O(n)$ operations at most. So, in total, the Crout method needs $O(n^2) * O(n) = O(n^3)$ operations, same as the Gaussian Elimination !!

The main advantage of direct factorization, however, rises when A is **sparse**, especially when the nonzero entries are close to the diagonal. It is very likely that the sparse pattern of A will be propagated into L and U so that many of equation (5) and (6) are simply "$0 = 0$" which requires no operation. We illustrate this phenomenon using a classical example

• Example. Tridiagonal matrix

$$A = \begin{pmatrix} a_1 & b_1 & & & & & \\ c_2 & a_2 & b_2 & & & & \\ & c_3 & a_3 & b_3 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & c_{n-1} & a_{n-1} & b_{n-1} \\ & & & & & c_n & a_n \end{pmatrix} \tag{7}$$

It has LU decomposition (a la the Doolittle method) with L, U sharing the same sparsity pattern – indeed, both are bi-diagonal

$$L = \begin{pmatrix} 1 & & & \\ l_2 & 1 & & \\ & \ddots & \ddots & \\ & & l_n & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} u_1 & b_1 & & \\ & u_2 & b_2 & \\ & & \ddots & \ddots \\ & & & u_n \end{pmatrix} \tag{8}$$

Note that the values of $b_1, b_2...$ from A are all preserved in U, which can be checked easily by expressing the b_{i-1} in A in terms of entries from L and U.

The design of algorithm for this problem follows exactly the same procedure described above but should skip all the trivial equations $0 = 0$. The reader is recommended to perform the algorithm on a 5-by-5 tri-diagonal matrix by hand for a thorough understanding.

For i=1:n

We scan [row i] of A. There are at most 3 nonzero entries.
Use a typical row with 3 entries, c_i, a_i, b_i. Note, at this point the values of $l_1 ... l_{i-1}$ and $u_i ... u_{i-1}$ should already be available.

c_i is at the $(i-1, i)$ position of A, so

$$c_i = l_i u_{i-1} \implies l_i = u_{i-1}/c_i$$

a_i is at the (i, i) position of A, so

$$a_i = l_i b_{i-1} + 1 * u_i \implies u_i = a_i - l_i b_{i-1}$$

where l_i is solved above and b_{i-1} appears in the $(i-1, i)$ position of U.

end of for i=1:n

Operation counts.

- Factorizatrion, $O(n)$ (why??);
- Solving $A\vec{x} = \vec{b}$, $O(n)$ (why??).

So much faster than $O(n^3)$.

§ 3.8 Iterative methods for solving linear systems.

First, review of vector and matrix norms.

- vector norms: e.g. $\|\vec{x}\|_\infty = \max\{|x_1|, ..., |x_n|\}$, $\|\vec{x}\|_2 = (|x_1|^2 + |x_2|^2 + ... + |x_n|^2)^{1/2}$, $\|\vec{x}\|_1 = |x_1| + |x_2| + ... + |x_n|$.
- matrix norms deduced from vector norms

$$\|A\| = \max_{\vec{x} \neq 0} \frac{||A\vec{x}||}{||\vec{x}||}.$$

Note each specific vector norm is associated with a matrix norm, e.g. $\|A\|_\infty = \max_{\vec{x} \neq 0} \frac{||A\vec{x}||_\infty}{||\vec{x}||_\infty}$ and we have learned that

$$\|A\|_\infty = \max_i \sum_j |a_{ij}|, \quad \text{max row sum}$$

- have proved that $||A\vec{x}|| \leq ||A|| \cdot ||\vec{x}||$, $||AB|| \leq ||A|| \cdot ||B||$

Now, iterative methods for solving $A\vec{x} = \vec{b}$,

$$\vec{x}_{k+1} = B\vec{x}_k + \vec{c} \tag{9}$$

where B is the iteration matrix.

ASSUME the above iteratin converges $\vec{x} = \lim_{k\to\infty} x_k$. By taking the limit on (9),

$$\vec{x} = B\vec{x} + \vec{c} \;\Rightarrow\; (I - B)\vec{x} = \vec{c}.$$

Thus, we require the above equation to be equivalent to $A\vec{x} = \vec{b}$. That is, theoretically,

$$(I - B)^{-1}\vec{c} = A^{-1}\vec{b}.$$

• Example. Jacobi method. To solve $A\vec{x} = \vec{b}$,

Splitting. $A = D + L + U$ where D contains only the diagonal entries of A, L the *stricly* lower triangular part and U the *strictly* upper triangular part. Other entries of D, L, U are filled with zeros.

(Note: the matrices L, U here are completely irrelavent to the LU decomposition)

Then the iteration scheme is derived as

$$A\vec{x} = \vec{b} \iff (D + L + U)\vec{x} = \vec{b} \iff D\vec{x} = -(L + U)\vec{x} + \vec{b}$$

$$\overset{thus}{\Longrightarrow} \qquad \vec{x}_{k+1} = -D^{-1}(L + U)\vec{x}_k + D^{-1}\vec{b}$$

In practice, the scheme is usually written as

$$D\vec{x}_{k+1} = -(L + U)\vec{x}_k + \vec{b} \tag{10}$$

since we only need to solve for $\vec{x}_{k+1}$ and don't have to find D^{-1}.

• Example. Apply the Jacobi method to the following system

$$A\vec{x} = \vec{b} \text{ where } A = \begin{pmatrix} 3 & -1 & 0 \\ 1 & -3 & 1 \\ 3 & -2 & 6 \end{pmatrix} \text{ and } \vec{b} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}.$$

Splitting

$$A = D + L + U = \begin{pmatrix} 3 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & 6 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 3 & -2 & 0 \end{pmatrix} + \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

Using (10), we have

$$D\vec{x}_{k+1} = -(L+U)\vec{x}_k + \vec{b}$$

and in terms of components $\vec{x}_k = \begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \end{pmatrix}$,

$$\begin{cases} 3x_1^{(k+1)} = -x_2^{(k)} + 1 \\ -3x_2^{(k+1)} = x_1^{(k)} + x_3^{(k)} + 2 \\ 6x_3^{(k+1)} = 3x_1^{(k)} - 2x_2^{(k)} + 3 \end{cases}$$

Operation count. In each step of Jacobi iteration, the operation count is $O(n^2)$. Why? Multiplication $(L+U)\vec{x}_k$ costs $O(n^2)$ operations – or more precisely, $O(\hat{n})$ with $\hat{n}$ being the number of nonzero entries of A. Then, solving for $\vec{x}_{k+1}$ in $D\vec{x}_{k+1} = ...$ only takes $O(n)$ operations because D is a **diagonal matrix**! So the total operation count is $O(n^2 \times m)$ or $O(\hat{n} \times m)$ where m is the number of iterations performed.

Comparison with Gaussian Elimination and LU factorization. These two both require $O(n^3)$ operations. So, for the Jacobi method, it can save computation time if $m << n$. The trade-off, however, is loss of accuracy that gets better as m gets larger. (Think about an iterative method learned before – the Newton's method for solving $f(x) = 0$)

Note. In some situations, speedy algorithms are more crucial than perfect accuracy!

• Error analysis for the Jacobi method.

Let's mimic the techinque used for the Newton's method, that is, subtract the iterative equation

$$\vec{x}_{k+1} = B\vec{x}_k + \vec{c}$$

from the exact one

$$\vec{x} = B\vec{x} + \vec{c}$$

to get

$$\vec{x} - \vec{x}_{k+1} = B(\vec{x} - \vec{x}_k)$$

$$\text{in terms of error} \implies \vec{e}_{k+1} = B\vec{e}_k,$$

$$\text{after } k \text{ iterations} \implies \vec{e}_k = B^k\vec{e}_0$$

$$\text{take vector norms} \implies \|\vec{e}\| = \|B^k\vec{e}_0\|$$

$$\text{recall the properties of norms} \implies \|\vec{e}\| \leq \|B\|^k\|\vec{e}_0\|.$$

Conclusion. The scheme converges if $\|B\| < 1$ in certain norm! And the rate of convergence is $O(\|B\|^n)$.

• Example. Use the previous example. The scheme is $D\vec{x}_{k+1} = -(L+U)\vec{x}_k + \vec{b}$ i.e. $\vec{x}_{k+1} = -D^{-1}(L+U)\vec{x}_k + D^{-1}\vec{b}$ so

$$B = -D^{-1}(L+U) = \begin{pmatrix} 0 & -1/3 & 0 \\ -1/3 & 0 & -1/3 \\ 3/6 & -2/6 & 0 \end{pmatrix}$$

Now, pick a suitable norm $\|\cdot\|_\infty$ and simply compute (see the review above)

$$\|B\|_\infty = \frac{5}{6} < 1 \quad \Rightarrow$$

the scheme used in the example guarantees convergence and the convergence rate is

$$\|\vec{e}_k\|_\infty \leq \left(\frac{5}{6}\right)^k \|\vec{e}_0\|_\infty$$

• Example. Show that the Jacobi method must converge if A is *diagonally dominant*, that is,

$$|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}| \quad \text{for all } i = 1, 2, 3, ...n$$

Hint: consider the infinity-norm.

§ 3.8 (cont'd) Guass-Seidel Method

Have learned the Jacobi method

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

Here, L, D, U come from the splitting $A = L + D + U$. We define the Jacobi iteration matrix

$$B_{Jac} \stackrel{def}{=} -D^{-1}(L+U)$$

and we know that $||B_{Jac}||_\infty < 1$ for diagonally dominant matrix A.

• Another iterative method for solving $A\vec{x} = \vec{b}$: Gauss-Seidel method

First, splitting $A = L + D + U$ as in Jacobi method. Then, rewrite $A\vec{x} = \vec{b}$ in equivalent forms

$$\begin{aligned} A\vec{x} = \vec{b} &\iff (L+D+U)\vec{x} = \vec{b} \\ &\iff (L+D)\vec{x} = -U\vec{x} + \vec{b} \\ &\iff \vec{x} = -(L+D)^{-1}U\vec{x} + (L+U)^{-1}\vec{b} \end{aligned} \tag{11}$$

By the last equation of (11) the iteration scheme has the same formality as before

$$\vec{x}_{k+1} = B_{GS}\vec{x}_k + \vec{c} = -(L+D)^{-1}U\vec{x}_k + (L+U)^{-1}\vec{b}$$

but with a different iteration matrix

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

In the case of A being sparse, however, $(L+D)^{-1}$ is mostly like to be dense with $O(n^2)$ nonzero entries, increasing the operation count. Instead, we use the *second* equation of (11) and solve for $\vec{x}_{k+1}$ in

$$(L+D)\vec{x}_{k+1} = -U\vec{x}_k + \vec{b}$$

which requires $O(\hat{n})$ operations with $\hat{n}$ the number of nonzero entries in $L+D$.

• Example.

$$A\vec{x} = \vec{b} \text{ where } A = \begin{pmatrix} 3 & -1 & 0 \\ 1 & -3 & 1 \\ 3 & -2 & 6 \end{pmatrix} \text{ and } \vec{b} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}.$$

Splitting

$$A = D + L + U = \begin{pmatrix} 3 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & 6 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 3 & -2 & 0 \end{pmatrix} + \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

So the Gauss-Seidel iteration scheme is

$$\begin{pmatrix} 3 & 0 & 0 \\ 1 & -3 & 0 \\ 3 & -2 & 6 \end{pmatrix} \vec{x}_{k+1} = \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \vec{x}_k + \vec{b}$$

and in each step $\vec{x}_{k+1}$ is solved using forward substitution. The operation count is $O(n^2)$.

• Example. Find an iteration scheme to solve the following n-by-n system (which is NOT tri-diagonal anymore) with the Gauss-Seidel iteration. Compare the operation counts of two theoretically equivalent approaches $\vec{x}_{k+1} = -(L+D)^{-1}U\vec{x}_k + (L+U)^{-1}\vec{b}$

and $(L+D)\vec{x}_{k+1} = -U\vec{x}_k + \vec{b}$.

$$A\vec{x} = \vec{b}, \qquad A = \begin{pmatrix} -2 & 1 & & & & & & 0.3 \\ 1 & -2 & 1 & & & & & \\ & 1 & -2 & 1 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & 1 & -2 & 1 & \\ 0.5 & & & & & 1 & -2 & \end{pmatrix}$$

The scheme is easy to write. What about operation counts? In using $\vec{x}_{k+1} = -(L+D)^{-1}U\vec{x}_k + (L+U)^{-1}\vec{b}$, each iteration step costs $O(n^2)$ operations because $(L+D)^{-1}$ is a dense matrix with $O(n^2)$ nonzero entries (try this with Matlab). On the other hand, using $(L+D)\vec{x}_{k+1} = -U\vec{x}_k + \vec{b}$ only requires $O(n)$ operations in each iteration step, thanks to the sparse structure of

$$L+D = \begin{pmatrix} -2 & 0 & & & & & & \\ 1 & -2 & 0 & & & & & \\ & 1 & -2 & 0 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & 1 & -2 & 0 & \\ 0.5 & & & & & 1 & -2 & \end{pmatrix} \text{ and } U = \begin{pmatrix} 0 & 1 & & & & & & 0.3 \\ 0 & 0 & 1 & & & & & \\ & 0 & 0 & 1 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & 0 & 0 & 1 & \\ & & & & & 0 & 0 & \end{pmatrix}$$

• Example. Do an error analysis on the previous example for $n = 10$ in terms of the infinity norm $\|B_{GS}\|_\infty$. Compare the infinity norm with $\|B_{Jac}\|_\infty$.

With the help of Matlab, we can compute

```
>> n=10; D=eye(n)*(-2); L=D*0; U=D*0;
>> for i=1:n-1 L(i+1,i)=1; U(i,i+1)=1; end;
>> L(n,1)=0.5; U(1,n)=0.3;
>> B=-inv(L+D)*U;
>> rs=sum(abs(B),2); max(rs)
```

Here, in the last line, `rs=sum(abs(B),2)` computes the row sum of B in absolute value. The second parameter of the function *sum* indicates column sum or row sum.

The answer is

$$\|B_{GS}\|_\infty = 0.9975 \implies \|\vec{e}_{k+1}\|_\infty \le 0.9975\|\vec{e}_k\|_\infty \qquad \text{convergent!}$$

The infinity norm of $B_{Jac} = -D^{-1}(L+U)$ can be easily computed by hand.

$$\|B_{Jac}\|_\infty = 1 \implies \|\vec{e}_{k+1}\|_\infty \le \|\vec{e}_k\|_\infty \qquad \text{no conclusion on convergence}$$

However, numerical experiment shows that the Jacobi method still converges in this case even though $\|B_{Jac}\|_\infty = 1$.

To this end, we introduce a fundamental quantity that determines the convergence rate of iterative schemes for linear systems ...

§ 3.8 (cont'd) Spectral radius

The spectral radius of a matrix, defined as,

$$\rho(B) = \max\{\text{abs(eigenvalues of } B)\}$$

is the most fundamental quantity in deciding the convergene of the iterative method

$$\vec{x}_{k+1} = B\vec{x}_k + \vec{c}. \tag{12}$$

In fact, we have the following theorem

Theorem 3 *Let $\vec{x}$ be the exact solution and $\vec{e} = \vec{x} - \vec{x}_k$ be the error. Then,*

$$\rho(B) = \lim_{k\to\infty} \frac{\|\vec{e}_{k+1}\|_\infty}{\|\vec{e}_k\|_\infty}$$

Proof. (Math 571)

This theorem implies that, if $\rho(B) < 1$, then $\|\vec{e}_k\|$ decays almost like $(\rho(B))^k$ for large k's and therefore the method (12) converges. With different wording, we can say the iteration converges at order 1 with asymptotic constant $\lambda = \rho(B)$. (Now it seems reasonable to use λ here since it is also a common symbol for eigenvalues).

Theorem 4 *For any matrix norm induced upon a vector norm, it is always true that*

$$\rho(B) \le \|B\|.$$

Proof We let λ_1 be the eigenvalue with the largest absolute value so that $\rho(B) = |\lambda_1|$. Let $\vec{u}_1$ be the associated eigenvector so that $B\vec{u}_1 = \lambda_1\vec{u}_1$. Then, by the definition of matrix norm

$$\begin{aligned}\|B\| &= \max_{\vec{u}\neq 0} \frac{\|B\vec{u}\|}{\|\vec{u}\|} \\ &\geq \frac{\|B\vec{u}_1\|}{\|\vec{u}_1\|} \\ &= \frac{\|\lambda_1\vec{u}_1\|}{\|\vec{u}_1\|} \\ &= |\lambda_1| = \rho(B).\end{aligned}$$

DONE.

This theorem asserts that the spectral radius serves as a lower bound of all matrix norms induced from vector norms. So if one can use $\|B\| < 1$ to show convergence, then $\rho(B) < 1$ will also show convergence. On the other hand, $\|B\| > 1$ does not necessarily imply divergence whereas $\rho(B) \geq 1$ does guarantee divergence.

• Example. Given matrix $A = \begin{pmatrix} 2 & -2 \\ 2 & 3 \end{pmatrix}$.

i) Compute $\rho(B_{Jac})$, $\|B_{Jac}\|_\infty$, $\|B_{Jac}\|_2$. What convergence rate does each one of them tell us?

ii) Compute $\rho(B_{GS})$, $\|B_{GS}\|_\infty$, $\|B_{GS}\|_2$. What convergence rate does each one of them tell us?

Solution. Split

$$A = L + D + U = \begin{pmatrix} 0 & 0 \\ 2 & 0 \end{pmatrix} + \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix} + \begin{pmatrix} 0 & -2 \\ 0 & 0 \end{pmatrix}.$$

i) $B_{Jac} = -D^{-1}(L+U) = \begin{pmatrix} 0 & 1 \\ -2/3 & 0 \end{pmatrix}$

$$\det(B_{Jac} - \lambda I) = \det \begin{pmatrix} -\lambda & 1 \\ -2/3 & -\lambda \end{pmatrix} = \lambda^2 + \frac{2}{3}$$

Set the above equation $= 0$ and solve for λ

$$\lambda_{1,2} = \pm\sqrt{\frac{2}{3}}i$$

Note. Imaginary numbers appear in the analysis of a real-number problem.

So $\rho(B_{Jac}) = \left|\sqrt{\frac{2}{3}}i\right| = \sqrt{\frac{2}{3}} < 1$. Thus, the Jacobi method converges at a rate $\|\vec{e}_k\| \sim \left(\sqrt{\frac{2}{3}}\right)^k$ as $k \to \infty$.

Now, easy to compute that $\|B_{Jac}\|_\infty = 1$. This condition alone does not imply convergence or divergence (inconclusive).

Also, compute $(B_{Jac})^T B_{Jac} = \begin{pmatrix} 4/9 & 0 \\ 0 & 1 \end{pmatrix}$ and find its eigenvalues $\lambda_{1,2} = \pm\frac{2}{3}$ so that$\|B_{Jac}\|_2 = \max\sqrt{\lambda_i((B_{Jac})^T B_{Jac})} = \sqrt{\frac{2}{3}}$. So the 2-norm of B_{Jac} implies the same convergence rate as $\rho(B_{Jac})$ does.

ii) $B_{GS} = -(L+D)^{-1}U = \begin{pmatrix} 0 & 1 \\ 0 & -2/3 \end{pmatrix}$.

$$\det(B_{GS} - \lambda I) = \det\begin{pmatrix} -\lambda & 1 \\ 0 & -\frac{2}{3} - \lambda \end{pmatrix} = \lambda^2 + \frac{2}{3}\lambda$$

Set the above equation $= 0$ and solve for λ

$$\lambda_1 = -\frac{2}{3},\ \lambda_2 = 0$$

So $\rho(B_{GS}) = \frac{2}{3} < 1$. Thus, the Jacobi method converges at a rate $\|\vec{e}_k\| \sim \left(\frac{2}{3}\right)^k$ as $k \to \infty$. The G-S method converges faster than the Jacobi method in this example.

Now, easy to compute that $\|B_{GS}\|_\infty = 1$. Again, this condition alone does not imply convergence or divergence (inconclusive).

Also, compute $(B_{GS})^T B_{GS} = \begin{pmatrix} 0 & 0 \\ 0 & 13/9 \end{pmatrix}$ and find its eigenvalues $\lambda_1 = 0,\ \lambda_2 = \frac{13}{9}$ so that$\|B_{GS}\|_2 = \max\sqrt{\lambda_i((B_{GS})^T B_{GS})} = \sqrt{\frac{13}{9}} > 1$. So the 2-norm of B_{GS} gives no information on the convergence of the G-S method.

§3.8 (cont'd) The SOR (successive over-relaxation) method

Before we go to the SOR method, let's discuss more about spectral radius. As a theoretical tool, spectral radius is indeed not easy to find in practice. For a handful of special matrices, though, we do know something about the convergence of Jacobi and Gauss-Seidel methods. We state without proving the following properties.

1. The Jacobi method converges for strictly diagonally dominant matrices A (see HW 5). That is, $\rho(B_{Jac}) < 1$. The Gauss-Seidel method also converges in this case.

2. The Gauss-Seidel method converges for symmetric positive definite (definition below) matrix A. That is, $\rho(B_{Jac}) < 1$.

3. For any A with $a_{ii} > 0$ and $a_{ij} \leq 0$ $(i \neq j)$, one of the following cases has to be true

   - $0 \leq \rho(B_{GS}) \leq \rho(B_{Jac}) < 1$
   - $1 \leq \rho(B_{Jac}) \leq \rho(B_{GS})$

   In other words, the Gauss-Seidel method either converges faster than the Jacobi method or diverges faster than the Jacobi method. To show exactly convergence, one needs further argument.

Definition A square matrix A is symmetric if $A^T = A$. A symmetric matrix A is positive definite if

$$\vec{x}^T A\vec{x} > 0$$

for all nonzero $\vec{x}$. Equivalently, A is symmetric positive definite if all its eigenvalues are positive.

• Example. Show that the Jacobi method converges for the difference matrix

$$A\vec{x} = \vec{b}, \qquad A = \begin{pmatrix} -2 & 1 & & & & & & \\ 1 & -2 & 1 & & & & & \\ & 1 & -2 & 1 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & 1 & -2 & 1 & \\ & & & & & 1 & -2 & \end{pmatrix}$$

Proof. A is diagonally dominant since $|a_{ii}| \geq \sum_{j=1}^{n} |a_{ij}|$ is true for all rows. But A is not *strictly* diagonally dominant because $|a_{ii}| > \sum_{j=1}^{n} |a_{ij}|$ is true only for $i = 1, n$. So, we can not directly use the above statements. Instead, let's prove $\rho(B_{Jac}) < 1$ by some careful argument.

Proof by contradiction. Assume $\rho(B_{Jac}) \geq 1$. Let λ be the leading eigenvalue so that $|\lambda| = \rho(B_{Jac}) \geq 1$. Let the associated eigenvector be $\vec{v} = (v_1, v_2, ..., v_n)^T$ so that

$$B_{Jac}\vec{v} = \lambda\vec{v}. \tag{13}$$

Now we know that, except the first and last row, the i-th row of B_{Jac} is $(0, ..., -\frac{1}{2}, 0, -\frac{1}{2}, ...0)$ with $-\frac{1}{2}$ in the (i-1) and (i+1) column. So the i-th row of $B_{Jac}\vec{v}$ is $-\frac{1}{2}v_{i-1} - \frac{1}{2}v_{i+1}$. Plug it into equation (13), we have

$$-\frac{1}{2}v_{i-1} - \frac{1}{2}v_{i+1} = \lambda_1 v_i \quad \text{for } i = 2, 3, ..., n-1 \tag{14}$$

Let v_k be the entry of $\vec{v}$ with the largest eigenvalue. By (14), we have

$$|\lambda v_k| = \left| -\frac{1}{2}v_{k-1} - \frac{1}{2}v_{k+1} \right| \leq \frac{1}{2}\left(|v_{k-1}| + |v_{k+1}|\right)$$

where the inequality is due to the triangle inequality. Since by assumption, $|\lambda| \geq 1$ and $|v_k| \geq |v_{k\pm1}|$, the above inequality has to be an equality

$$|\lambda v_k| = \frac{1}{2}\left(|v_{k-1}| + |v_{k+1}|\right)$$

which leaves us with only one possibility

$$|v_k| = |v_{k-1}| = |v_{k+1}|.$$

So v_{k-1} and v_{k+1} have the same absolute value as v_k and they all have the largest absolute value among all $v_1, v_2, ..., v_n$.

We can propagate the above argument to v_{k-2} and v_{k+2}, then v_{k-3} and v_{k+3} ... Eventually, all v_i's share the same absolute value. In particular $|v_1| = |v_2|$. But by inspecting the first row of equation (13), we have $2v_1 = v_2$ which yields $v_1 = 0$ and all entries of $\vec{v}$ are zero. Contradiction to the definition of eigenvectors!

DONE.

• Example. For the same matrix A as above, show that the Gauss-Seidel method converges and converges no slower than the Jacobi method.

Proof Since the above example verifies $\rho(B_{Jac}) < 1$, we use statement 3 on page 9 for $-A$ to conclude that

$$\rho(B_{GS}) \leq \rho(B_{Jac}) < 1.$$

DONE.

• The SOR method.

Motivation: can we improve the convergence rate, i.e. lower $\rho(B)$ while maintaining the same level of operation count?

Idea: the G-S method, in its exact form, is

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

Take a linear combination of the LHS and RHS with weights $1-\omega$ and ω. It should give us $\vec{x}$ again

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

Eliminate the inverse term D^{-1}

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

Assign index $k+1$ to the $D\vec{x}$ term on the LHS and the $L\vec{x}$ term on the RHS. Assign index k to the rest terms. (This way, the coeffient matrix of $\vec{x}_{k+1}$ is *lower triangular* which we have fast algorithm to solve.)

$$D\vec{x}_{k+1} = (1-\omega)D\vec{x}_k - \omega\left(L\vec{x}_{k+1} - U\vec{x}_k - \vec{b}\right)$$

The above form is the SOR method used in practice. Theoretically, the iteration can be written as

$$\vec{x}_{k+1} = B^{\omega}_{SOR}\vec{x}_k + \vec{c}$$

where

$$B^{\omega}_{SOR} = (D+\omega L)^{-1}((1-\omega)D - \omega U).$$

Note. The Gauss Seidel method amounts to $\omega = 1$.

It is easy to see that the SOR method has the same operation count as the G-S method, thanks to their lower triangular structure. Now, since ω is a free paramter, can we find some optimal value ω^* such that $\rho(B^{\omega^*}_{SOR})$ is small, especially smaller than B_{GS}? The answer is yes if A has special structures.

Theorem 5 *If A is symmetric postive definte and is (block) tri-diagonal, then $\rho(B_{GS}) = \rho^2(B_{Jac}) < 1$. And the optimal value for the SOR method is*

$$\omega^* = \frac{2}{1+\sqrt{1-\rho(B_{GS})}}$$

in which case the spectral radius

$$\rho(B^{\omega^*}_{SOR}) = \omega^* - 1.$$

Note. By simple manipulation, it is not difficult to show that $\rho(B^{\omega^*}_{SOR}) < \rho(B_{GS}) < \rho(B_{Jac})$. Thus, in this special case (positive defnite and tri-diagonal), the SOR method converges faster than GS faster than Jacobi.