

Math 642:550 — Summer 2009
MTTh 6:00–8:30 PM Hill 425
Prof. Bumby

Supplement 10, Numerical Methods in Linear Algebra

1. Introduction Chapter 7 of the textbook describes computer algorithms for solving equations and finding eigenvalues and eigenvectors. Numerical methods require attention to the **steps of a computation**, not just to the result that would be obtained if those steps were performed exactly. Even the most fundamental rules of algebra, like the associative laws of addition and multiplication, can fail for approximate computations.

2. Two kinds of numbers In order to facilitate hand computation, most exercises done so far used matrices with integer entries and relied on methods that were able to **give exact answers easily** with this data. These same methods can be programmed to give exact answers to larger problems involving integer matrices. If a system supporting **arbitrary precision** is used, then exact answers can be obtained even if the numbers in the problem are very large. If the numbers appearing throughout the solution are also required to be integers, **division must be avoided**. For example, in the case of the algorithm for QR factorization, it was convenient to get a preliminary form Q_0 of the first factor with columns that are orthogonal vectors with **integer** entries, along with a second factor R_0 with $A = Q_0 R_0$. If the lengths of the vectors in Q_0 are arranged in a diagonal matrix D , then $A = Q_0(D^{-1}D)R_0 = (Q_0 D^{-1})(DR_0)$. Here $Q = Q_0 D^{-1}$ has **orthonormal columns**, so $R = DR_0$ is the second factor in the QR factorization. In addition, a special method for finding the characteristic polynomial was described that relied on matrix **multiplication** for most calculations to facilitate computation with integers.

Although it is common for books to give formulas for desired quantities, it is usually a bad idea to use those formulas. These formulas were only **possible to derive** because these equations are very easy to solve. By organizing your solution in terms of these equations, you also make it less likely that you will assert an answer that obviously fails to have the desired property. There are often many ways to solve an equation, but a formula for the solution suggests only one approach to simplifying its expression, and this way may not be the best approach to the problem. The **quadratic formula** gives a good example. To get a simple algebraic expression, one writes a denominator of $2a$. However, if the coefficient of x^2 is 1 and the coefficient of x is an even integer, the **completing the square** method works entirely with integers, and the answer that it gives requires no further simplification. Another example: in linear algebra, the solution to $Ax = b$ is often written $x = A^{-1}b$, but much effort is spent on solving such equations **without** finding A^{-1} .

Most applications involve numbers that are **obtained as measurements**. The appropriate tools for computing with such numbers are **floating point** representations on a computer (or calculator). Such representations are **essentially** approximations. Arithmetic with these quantities introduces **round off** and **truncation** errors. These **errors** are **not mistakes**. They are the result of limitations in the computer representation of real numbers that are required to get answers in finite time. The subject of **Numerical Analysis** deals with finding methods of computation that **control error** while giving efficient procedures for approximating the desired quantities to **reasonable** accuracy. Such methods are often **iterative**, generating a sequence that, if calculated exactly, converges to the desired quantity. The process must stop when **it can give no improvement with the available data**.

A **floating point number** consists of a **sign**, an **exponent**, and a **mantissa** packed into a fixed number of bits according to an established standard. Here, the mantissa is the collection of digits relative to the

scale set by the exponent. Calculators use base 10, and would write something like 5.237×10^{21} . The internal representation in a computer is more likely to be in **binary** so that the exponent represents a power of 2. What is important is that numbers can be scaled over a very large range, and show a fixed number of **significant digits** that is the same, independent of the size of the number or whether the number is positive or negative. (The number zero has a special representation distinct from all others.) This means that the appropriate measure of accuracy of a computation is **relative error** obtained by dividing the difference between correct and computed values by something on the scale of the correct value. The exact definition is not important — one only works with **bounds** on the error to determine how much of the mantissa can be trusted — so any convenient measure of the size of the number can be used as denominator. Relative error behaves reasonably when numbers are multiplied or divided, but addition or subtraction can be troublesome. If numbers are of much different sizes, the smaller one will be written to the scale of the larger causing initial digits to be padded with 0 by shifting, while the later digits get shifted off the end and lost. The situation is worse if the sum is smaller than the numbers being combined since there are not enough digits to give much accuracy after the number is written to its proper scale. This leads to an apparent breakdown of the rules of algebra in which equivalent expressions can give very different answers. In numerical work, the **algorithm** used in computation is more important than any formula that may be used to describe the result. A formula may have a role in proving that the algorithm is correct, but it **should not** guide the computation.

3. Efficient computation The small problems used as exercises allow many ways of presenting the solution with no preferred method. We have given **suggestions** that aim to use a **structured computation** in which the location in which a quantity appears indicates its significance. The biggest danger in hand or semi-automatic computation (where you use a calculator for parts of the calculation, but record the steps by hand) is that an incorrect value (often only the sign of one number) will get written **somewhere** in one step, and this will affect all subsequent steps. The best protection against mistakes is to check the answer with the original data. Such checks should be a **routine** part of all computations. **Everyone** makes mistakes, often in unlikely places, but **it is foolish to commit to them in the face of contrary evidence**. It is also useful to have some theoretical properties of the answer that can be easily verified. For example, we **know** that the dominant eigenvalue of a matrix with positive entries has an eigenvector with components positive. Similarly, the eigenvalues of a real symmetric matrix are all real. Any computation that leads to results contradicting these theorems should be **visibly disturbing**. You need to be able to find mistakes and correct them before showing your work to anyone. Only correct answers are to be accepted. It is a sad fact, that your command of results learned in this course will be much more reliable than your ability to do arithmetic with fractions or signed numbers that you were supposed to have mastered long ago.

Some of this applies also to machine computation. The computation will be done by a **program** that records intermediate results according to a plan. Once the program is accepted as correct, special tricks may be used to save time or space in the computation. For example, in an LU factorization program, the answer will overlay the original matrix. Cheap memory limits the usefulness of such tricks, and high level programming languages treat the **programmer's time** as the **most valuable resource**. However, these tricks are part of the legacy of numerical linear algebra and will be used if they are not otherwise wasteful. Tricks are generally not recommended for hand computation, but machines are not confused by them. The routine for printing the answer can put the parts of answer where they are expected.

The **fast Fourier transform** gives an example where efficient use of space is an **essential** part of the computation. For large n , the matrices shown in formula (16) on page 195 of the text (in the case of $n = 4$) cannot appear as explicit arrays. The n^2 elements of such an array could not be written by a program that is supposed to run in only a constant multiple $n \log n$ steps (the type of steps that are counted may lead to a higher power of $\log n$, but certainly not an extra factor of n). The calculation will have a vector of

n numbers as input and produce another such vector as output. The **linear transformations** represented by those matrices must be applied to vectors, but the program doing it cannot use the matrix. One must describe the transformation more efficiently. Since the matrices only have two nonzero elements in each row, a **sparse matrix** data structure could be used that only records the nonzero entries and their location, but the special forms of the linear transformations appearing in this computation may lead to programs that don't appear to use matrices at all.

The use of Householder matrices to compute the QR factorization is mentioned in section 7.3 of the text (pages 361-2). It concludes with the observation that Q can be stored as a product of Householder matrices instead of being computed as a single orthogonal matrix. This is efficient because **the action of** a single Householder matrix can be described in terms of a vector in the direction being reflected. Thus, n such matrices need only n^2 locations — no more than the whole product — and Q **need not be seen to be used**. A typical application involves multiplying a **single vector** by Q^T . Using the factored form takes no longer than using a computed product. Either way, there are only a constant multiple of n^2 products of numbers to be found.

4. Vector and matrix norms There are several ways to measure the **size of a vector** $v \in \mathbb{R}^n$. The most familiar is the **Euclidean length** $(v^T v)^{1/2}$. This is used in the text. In order to have a suitable foundation to work with this definition, much of chapter 6 is a prerequisite. The **Rayleigh quotient** has shown how valuable this norm can be in collecting information about matrices. Here, we shall use a norm that is easier to work with: **the maximum of the absolute values of the entries** of v . This is mentioned briefly in exercise 7.2.18, where it is given the name ℓ^∞ -norm, and it plays a role in **Gershgorin's Theorem** mentioned before exercise 7.4.4. However, it was not really **used** in any of the exercises. This norm was also implicit in our treatment of the Perron-Frobenius Theorem. There are many other vector norms: all that a function $N(v)$ needs to earn that designation is that

- (1) $N(v) \geq 0$ for all v with $N(v) = 0$ if and only if $v = 0$;
- (2) $N(\alpha v) = |\alpha| N(v)$;
- (3) $N(v_0 + v_1) \leq N(v_0) + N(v_1)$.

Here, (1) is a **positivity** condition; (2) says that the norm is **homogeneous** of degree 1; and (3) is the **triangle inequality**. Both candidates for vector norms have these properties.

If you have a **vector norm** $N(v)$, there is an associated **matrix norm** on n by n matrices M defined by

$$N(M) = \sup \{ N(Mv) : N(v) = 1 \}.$$

Because of positivity and homogeneity, this is equivalent to the **least upper bound** of $N(Mv)/N(v)$ for **all** nonzero vectors v .

There are matrix norms that are not constructed in this way from vector norms, and some of them are quite useful. For now, we show how a norm constructed from a vector norm can be computed directly from the matrix entries.

Proposition. *The matrix norm corresponding to the maximum norm is*

$$N(M) = \max_i \sum_j |m_{ij}|.$$

Proof. $N(v) \leq 1$ says that all $|v_j| \leq 1$. In this case,

$$\left| \sum_j m_{ij} v_j \right| \leq \sum_j |m_{ij} v_j| \leq \sum_j |m_{ij}| v_j,$$

so $N(M)$ is **no larger than** our proposed value. To see that it is **at least this large**, find the row i where the sum attains its maximum and choose $v_j = \pm 1$ so that $|m_{ij}| = m_{ij} v_j$. Then our proposed value is $N(Mv)/N(v)$ for this v .

5. Perron-Frobenius revisited The first bound that we obtained on the dominant eigenvalue of a positive matrix was actually the **maximum norm** of the matrix. More generally, we have.

Proposition. *If $N(M)$ is any matrix norm induced from a vector norm $N(v)$, then the absolute values of all eigenvalues of M are at most $N(M)$.*

Proof. If $Mv = \lambda v$, then

$$N(Mv) = N(\lambda v) = |\lambda| N(v),$$

but $N(Mv) \leq N(M)N(v)$.

A change of basis can change the norm of a vector or a matrix. However, while $N(S^{-1}MS)$ is **not the same** as $N(M)$, it is always a **matrix norm** of M . Indeed, if the original norm is induced by the **vector norm** $N(v)$, this is induced by $N(Sv)$, which is easily seen to be a vector norm.

For application to the Perron-Frobenius theorem, let S be a **diagonal matrix** that divides the j^{th} coordinate of a vector by the j^{th} coordinate of a given positive vector v_0 . To say that $Mv_0 \leq \beta v_0$, component by component, is to say that $N(S^{-1}MS) \leq \beta$. This shows that the dominant eigenvalue is bounded above by any such β . This complements our earlier result that said that $Mv_0 \geq \alpha v_0$ implies that α is a lower bound on the dominant eigenvalue.

6. Power methods Positive matrices give good examples of the use of the power method for finding eigenvalues, since we know that the dominant eigenvalue belongs to a positive eigenvector. After some steps of the form $v_{k+1} = Mv_k$ from any positive starting vector v_0 , the bounds on the ratios of corresponding components of v_{k+1} and v_k give bounds on the dominant eigenvalue.

Although the power method seems impressive when first met, it soon begins to feel painfully slow. The **shifted inverse power method** does a good job of improving the rate of convergence while retaining the same ease of analysis. For positive matrices, this change can be done in a way that also involves finding the dominant eigenvalue of a positive matrix.

7. Large linear systems On page 370 of the textbook, Strang gives a (very) brief description of some marvelous results showing the value of **successive overrelaxation** over other iterative methods. A more thorough treatment can be found in David M. Young, *Iterative Solution of Large Linear Systems*, Dover, New York, 2003. In order to appreciate both the power and the limitations of this method, the hypotheses underlying formula (7) need to be given.

The intended problems are those arising from boundary value problems, so the matrix A for which we need to solve $Ax = b$ is something like the tridiagonal **finite difference matrix** appearing in many examples in the textbook.

Another type of matrix that is important in the theory is one having the block form

$$\begin{bmatrix} D_1 & B \\ C & D_2 \end{bmatrix}$$

where D_1 and D_2 are diagonal.

8. Diagonal dominance A matrix $A = [a_{ij}]$ is said to be **strongly diagonally dominant** if, for all i

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

Gershgorin's theorem assures us that A is nonsingular.

In Jacobi's method, we write $A = S - T$ with S equal to the diagonal of A and T the negative of the off-diagonal part. As noted on page 367 of the textbook, the error e_k after k steps of the iteration

$$Sx_{k+1} = Tx_k + b$$

satisfies

$$e_{k+1} = S^{-1}Te_k.$$

so that the rate of convergence is determined by the largest absolute value of an eigenvalue of $S^{-1}T$, which is bounded by any matrix norm of $S^{-1}T$. However, diagonal dominance gives immediately that the max norm of $S^{-1}T$ is less than 1. Thus Jacobi's method converges. However, if there is an eigenvalue close to 1, the convergence can be slow. In particular, if A is the n by n finite difference matrix, with 2 on the main diagonal and -1 on the adjacent diagonal, the vector with i^{th} entry $\sin i\theta$ for $(n+1)\theta = k\pi$ is an eigenvector with eigenvalue $2 - 2\cos\theta$. The $S^{-1}T$ of Jacobi's method has the same eigenvectors and its largest eigenvalue is $\cos\pi/(n+1)$.

The Gauss-Seidel method and its overrelaxations use

$$\begin{aligned} S &= D + \omega L \\ T &= (1 - \omega)D - \omega U \end{aligned}$$

where D is the diagonal of A , L is the part below the diagonal and U is the part above the diagonal, so that $A = D + L + U$ (this is not to be confused with a different use of the same letters in the LDU factorization). The case $\omega = 1$ is the Gauss-Seidel method. A number λ is an eigenvalue of $S^{-1}T$ if $T - \lambda S$ is singular. This matrix is

$$(1 - \omega - \lambda)D - \omega U - \omega\lambda L.$$

9. The tridiagonal lemma Applying Jacobi's method to a tridiagonal matrix calls for finding the eigenvalues of a tridiagonal matrix B with zero diagonal. Theorem 2.1 of chapter 5 of David Young's book gives a general result that is used to prove that the non zero eigenvalues of such a matrix occur in $\pm\lambda$ pairs. The proof consists of multiplying B left and right by the diagonal matrix whose entries are alternately $+1$ and -1 . This shows that B and $-B$ have the same eigenvalues. The same argument is the key to Exercise 5.6.4 of the textbook.

A more general argument uses the diagonal matrix with entries c^k on the right and the diagonal matrix with entries c^{-k} on the left applied to the matrix $T - \lambda S$. The conclusion is that determinant depends only on the ratio of the square of the coefficient of D to the product of the coefficients of L and U . The family of matrices covered by this argument include the matrix associated with the iteration in Jacobi's method, where this ratio is the square of an eigenvalue of that matrix. Thus, with μ equal to an eigenvalue for Jacobi's method, we have

$$(\lambda + \omega - 1)^2 = \lambda\omega^2\mu^2$$

This equation shows that the product of the values of λ is $(\omega - 1)^2$, forcing $\omega < 2$ if the eigenvalues are to remain of absolute value less than 1.

The analysis is simpler when all μ are real, as they will be if A is symmetric. In this case, there are either a pair of real values of λ (which must be positive since the equation has λ equal to a square) or a pair of conjugate complex values. In the complex case, $|\lambda| = |\omega - 1|$. In the real case, the larger choice of λ is easily seen to be an increasing function of μ and a decreasing function of ω . As long as there are real eigenvalues, we do better by increasing ω , reaching a minimum when the largest μ gives λ as a double root.

10. Exercises

Let

$$M = \begin{bmatrix} 4 & 3 & 2 \\ 7 & 1 & 5 \\ 5 & 2 & 2 \end{bmatrix}$$

- A.** Find the **maximum norm** of M .
- B.** Start from $v_0 = [1, 1, 1]^T$ and use the **power method** $v_{k+1} = Mv_k$ to compute v_1, v_2 and v_3 . Use this to get upper and lower bounds on the **dominant eigenvalue** of M .

End of Supplement