

Math 642:550 — Summer 2007
MTTh 6:00–8:30 PM LSH-A139
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 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**. Some special methods that rely on matrix **multiplication** for most calculations were introduced to facilitate computation with integers.

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 is stopped 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**. A verbal description of the computation is not as useful. You need to be able to find mistakes and correct them. 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 $n \log n$ 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 will not use the matrix. 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 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. 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