CSCE/MATH 4/847 LECTURE NOTES

Part I: Fundamentals

1. Matrix-Vector Multiplication

First Day:

(1) Welcome
(2) Pass out information sheets
(3) Take roll
(4) Open up home page and have students do same to check for login problems
(5) Go through information materials

(8/22/06) We cover (review) the basics: matrix multiplication, addition, scalar multiplication, transposes, block arithmetic, inner and outer vector products and invertible matrices.

We also cover (review) basic vector space concepts such as span, linear independence/dependence, basis and dimension. We also review the vector spaces associated with a matrix (row, column=range and null space.)

By way of review, here are some of the definitions and facts that are used but not formally presented in the text.

Definition 1.1. Let A be a square matrix. Then a *(two-sided) inverse* for A is a square matrix B of the same size as A such that $AB = I = BA$. If such a B exists, then the matrix A is said to be *invertible*.

Definition 1.2. An *(abstract) vector space* is a nonempty set V of elements called vectors, together with operations of vector addition ($+$) and scalar multiplication ($\cdot$), such that the following laws hold for all vectors $\mathbf{u}, \mathbf{v}, \mathbf{w} \in V$ and scalars $a, b \in \mathbb{F}$ (the field of scalars).

(1): (Closure of vector addition) $\mathbf{u} + \mathbf{v} \in V$.
(2): (Commutativity of addition) $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$.
(3): (Associativity of addition) $\mathbf{u} + (\mathbf{v} + \mathbf{w}) = (\mathbf{u} + \mathbf{v}) + \mathbf{w}$.
(4): (Additive identity) There exists an element $\mathbf{0} \in V$ such that $\mathbf{u}+\mathbf{0} = \mathbf{u} = \mathbf{0}+\mathbf{u}$.
(5): (Additive inverse) There exists an element $-\mathbf{u} \in V$ such that $\mathbf{u}+(-\mathbf{u}) = \mathbf{0} = (-\mathbf{u}) + \mathbf{u}$.
(6): (Closure of scalar multiplication) $a \cdot \mathbf{u} \in V$.
(7): (Distributive law) $a \cdot (\mathbf{u} + \mathbf{v}) = a \cdot \mathbf{u} + a \cdot \mathbf{v}$.
(8): (Distributive law) $(a + b) \cdot \mathbf{u} = a \cdot \mathbf{u} + b \cdot \mathbf{u}$.
(9): (Associative law) $(ab) \cdot \mathbf{u} = a \cdot (b \cdot \mathbf{u})$.
(10): (Monoidal law) $1 \cdot \mathbf{u} = \mathbf{u}$.

Definition 1.3. A *subspace* of the vector space V is a subset W of V such that W, together with the binary operations it inherits from V, forms a vector space (over the same field of scalars as V) in its own right.

An example of a subspace is the span of a set of vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ which is defined as

$$\operatorname{span}\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\} \equiv \{c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n \,|\, c_1, c_2, \ldots c_n \text{ are scalars } \}.$$

Definition 1.4. A set of vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ is is said to *span the space* V if

$$V = \operatorname{span}\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}.$$

Definition 1.5. A set of vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ is *linearly independent* if whenever $c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n = 0$ it follows that $c_1 = c_2 = \cdots = c_n = 0$. Otherwise, the set is said to be *linearly dependent*.

Definition 1.6. A *basis* for the vector space V is a spanning set of vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ which is a linearly independent set.

Definition 1.7. The vector space V is called *finite dimensional* if there is a finite set of vectors $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ which is a spanning set for V.

Theorem 1.8. *(Basis Theorem) Every finite dimensional vector space has a basis.*

Theorem 1.9. *(Dimension Theorem) Let V be a finite dimensional vector space. Then any two bases of V have the same number of elements which is called the dimension of the vector space and denoted as* $\dim V$.

Corollary 1.10. *If W is a subspace of the finite dimensional space V, then W is finite dimensional and* $\dim W \leq \dim V$.

Definition 1.11. Let A be an $m \times n$ matrix. Assuming that the field of scalars is $\mathbb{F} = \mathbb{C}$ or $\mathbb{R}$, we have associated subspaces

Row space: $\operatorname{row}(A) = \mathcal{R}(A) = \operatorname{span}\{\text{columns of } A^T\} \subseteq \mathbb{F}^n$.

Column space: $\operatorname{range}(A) = \mathcal{C}(A) = \operatorname{span}\{\text{columns of } A\} = \{A\mathbf{x} \,|\, \mathbf{x} \in \mathbb{F}^n\} \subseteq \mathbb{F}^m$.

Null space: $\operatorname{null}(A) = \mathcal{N}(A) = \{\mathbf{x} \in \mathbb{R}^n \,|\, A\mathbf{x} = \mathbf{0}\} \subseteq \mathbb{F}^n$.

Definition 1.12. $\operatorname{rank}(A) = \dim \operatorname{row}(A)$.

Theorem 1.13. *(Rank Theorem) Let A be an $m \times n$ matrix such that* $\operatorname{rank} A = r$. *Then*

(1) $\dim \mathcal{C}(A) = r$

(2) $\dim \mathcal{R}(A) = r$

(3) $\dim \mathcal{N}(A) = n - r$

Definition 1.14. Let A be a square $n \times n$ matrix. An *eigenvector* of A is a nonzero vector $\mathbf{x}$ in $\mathbb{R}^n$ (or $\mathbb{C}^n$, if we are working over complex numbers), such that for some scalar λ, we have

$$A\mathbf{x} = \lambda\mathbf{x}$$

The scalar λ is called an *eigenvalue* of the matrix A, and we say that the vector $\mathbf{x}$ is an *eigenvector belonging to the eigenvalue* λ. The pair $\{\lambda, \mathbf{x}\}$ is called an *eigenpair* for the matrix A.

Eigenvectors of A, as defined above, are also called *right eigenvectors* of A. Notice that if $A^T\mathbf{x} = \lambda\mathbf{x}$, then

$$\lambda\mathbf{x}^T = (\lambda\mathbf{x})^T = \left(A^T\mathbf{x}\right)^T = \mathbf{x}^T A.$$

For this reason, eigenvectors of A^T are called *left eigenvectors* of A.

Some standard theorems from elementary linear algebra:

Theorem 1.15. *Let A be a square $n \times n$ matrix. Then*

(1): *The eigenvalues of A consist of all scalars λ that are solutions to the nth degree polynomial equation*

$$\det(\lambda I - A) = 0$$

(2): *For a given eigenvalue λ, the eigenvectors of the matrix A belonging to that eigenvalue consist of all nonzero elements of $\mathcal{N}(\lambda I - A)$.*

Definition 1.16. Given an eigenvalue λ of the matrix A, the *eigenspace* corresponding to λ is the subspace $\mathcal{N}(\lambda I - A)$ of $\mathbb{R}^n$ (or $\mathbb{C}^n$). We write $\mathcal{E}_\lambda(A) = \mathcal{N}(\lambda I - A)$.

Definition 1.17. By an *eigensystem* of the matrix A, we mean a list of all the eigenvalues of A and, for each eigenvalue λ, a complete description of the eigenspace corresponding to λ.

Definition 1.18. Matrix A is said to be *similar* to matrix B if there exists an invertible matrix P such that

$$P^{-1}AP = B.$$

The matrix P is called a *similarity transformation* matrix.

Theorem 1.19. *Suppose that A is similar to B, say $P^{-1}AP = B$. Then:*

(1): *For every polynomial $q(x)$,*

$$q(B) = P^{-1}q(A)P$$

(2): *The matrices A and B have the same characteristic polynomial, hence the same eigenvalues.*

2. Orthogonal Vectors and Matrices

(8/24/06) First, we examine the correct version of inner product for possibly complex vectors $\mathbf{u}, \mathbf{v} \in \mathbb{C}^n$:

$$\mathbf{u}^* \mathbf{v} = \sum_{k=1}^{n} \bar{u}_k v_k.$$

We explore the inner product and its relation to the so-called induced norm, followed by a discussion of norm and inner product properties.

Next, we examine orthogonal and orthonormal vectors, and their relation to orthogonal matrices and coordinates.

Finally, as time permits, we see how well Matlab knows vectors and matrices by a quick spin through the Matlab lecture notes.

3. Norms

(8/29/06) We review the definition of norm and study both matrix and vector norms. Discuss induced norms, operator norms, p-norms and Frobenius norm.

4. The Singular Value Decomposition

(8/31/06) We prove the Schur triangularization theorem and use this to prove the principal axes theorem and the SVD.

5. More on the SVD

(9/5/06) We discuss some of the key applications of the SVD, including computation of null space, range and low rank approximations.

Here we need a discussion of change of bases, so here is a more complete rendition of this topic.

*Change of Basis and Linear Operators

We will indicate that $T : V \to W$ is a linear operator, $B = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ is a basis of V and $C = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m\}$ is a basis of W with the notation

$$T : V_B \to W_C \text{ or } V_B \xrightarrow{T} W_C.$$

An example of a linear operator is given by matrix multiplication. Let $A \in \mathbb{C}^{m,n}$and define the operator $T_A : \mathbb{C}^n \to \mathbb{C}^m$ by the formula

$$T_A(\mathbf{x}) = A\mathbf{x}.$$

One checks that linearity holds, that is, for all vectors $\mathbf{x}, \mathbf{y} \in \mathbb{C}^n$and scalars c, d,

$$T_A(c\mathbf{x} + d\mathbf{y}) = cT_A(\mathbf{x}) + dT_A(\mathbf{y}).$$

Now let $\mathbf{v} \in V$ be given. We know that there exists a unique set of scalars, the coordinates $c_1, c_2, \ldots, c_n$ of $\mathbf{v}$ with respect to this basis, such that

$$\mathbf{v} = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n.$$

Thus by linearity of T we see that

$$\begin{aligned} T(\mathbf{v}) &= T(c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n) \\ &= c_1T(\mathbf{v}_1) + c_2T(\mathbf{v}_2) + \cdots + c_nT(\mathbf{v}_n). \end{aligned}$$

It follows that we know everything about the linear operator T if we know the vectors $T(\mathbf{v}_1), T(\mathbf{v}_2), \ldots, T(\mathbf{v}_n)$.

Now go a step further. Each vector $T(\mathbf{v}_j)$ can be expressed uniquely as a linear combination of $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m$, namely

$$T(\mathbf{v}_j) = a_{1,j}\mathbf{w}_1 + a_{2,j}\mathbf{w}_2 + \cdots + a_{m,j}\mathbf{w}_m. \tag{1}$$

In other words, the scalars $a_{1,j}, a_{2,j}, \ldots a_{m,j}$ are the coordinates of $T(\mathbf{v}_j)$ with respect to the basis $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m$. Stack these in columns and we now have the $m \times n$ matrix $A = [a_{i,j}]$ which contains everything we need to know in order to compute $T(\mathbf{v})$. In fact, with the above terminology, we have

$$\begin{aligned} T(\mathbf{v}) &= c_1T(\mathbf{v}_1) + c_2T(\mathbf{v}_2) + \cdots + c_nT(\mathbf{v}_n) \\ &= c_1\left(a_{1,1}\mathbf{w}_1 + a_{2,1}\mathbf{w}_2 + \cdots + a_{m,1}\mathbf{w}_m\right) + \\ &\qquad \cdots + c_n(a_{1,n}\mathbf{w}_1 + a_{2,n}\mathbf{w}_2 + \cdots + a_{m,n}\mathbf{w}_m) \\ &= (a_{1,1}c_1 + a_{1,2}c_2 + \cdots + a_{1,n}c_n)\mathbf{w}_1 + \\ &\qquad \cdots + \left(a_{m,1}c_1 + a_{m,2}c_2 + \cdots + a_{m,n}c_n\right)\mathbf{w}_m. \end{aligned}$$

Look closely and we see that the coefficients of these vectors are themselves coordinates of a matrix product, namely the matrix A times the column vector of coordinates of $\mathbf{v}$ with respect to the chosen basis of V. The result of this matrix multiplication is a column vector whose entries are the coordinates of $T(\mathbf{v})$ relative to the chosen basis of W. So in a certain sense, computing the value of a linear operator amounts to no more than multiplying a (coordinate) vector by the matrix A. Now we make the following definition.

Definition 5.1. The *matrix of the linear operator $T : V_B \to W_C$ relative to the bases B* and C is the matrix $[T]_{B,C} = [a_{i,j}]$ whose entries are specified by Equation (1). In the case that $B = C$, we simply write $[T]_B$.

Recall that we denote the coordinate vector of a vector $\mathbf{v}$ with respect to a basis B by $[\mathbf{v}]_B$. Then the above calculation of $T(\mathbf{v})$ can be stated succinctly in matrix/vector terms as

$$[T(\mathbf{v})]_C = [T]_{B,C}\,[\mathbf{v}]_B \tag{2}$$

This equation has a very interesting application to the standard spaces. Recall that a matrix operator is a linear operator $T_A : \mathbb{R}^n \to \mathbb{R}^m$ defined by the formula $T_A(\mathbf{x}) = A\mathbf{x}$, where A is an $m \times n$ matrix. It turns out that *every* linear operator on the standard vector spaces is a matrix operator. The matrix A for which $T = T_A$ is called the *standard matrix* of T.

Theorem 5.2. *If $T : \mathbb{R}^n \to \mathbb{R}^m$ is a linear operator, B and C the standard bases for $\mathbb{R}^n$ and $\mathbb{R}^m$, respectively, and $A = [T]_{B,C}$, then $T = T_A$.*

Proof. The proof is straightforward: for vector $\mathbf{x}$, $\mathbf{y} = \mathbf{T}(\mathbf{x})$ in standard spaces with standard bases B, C, we have $\mathbf{x} = [\mathbf{x}]_B$ and $\mathbf{y} = [\mathbf{y}]_C$. Therefore,

$$T(\mathbf{x}) = \mathbf{y} = [\mathbf{y}]_C = [T(\mathbf{x})]_C = [T]_{B,C}[\mathbf{x}]_B = [T]_{B,C}\mathbf{x} = A\mathbf{x},$$

which proves the theorem. □

Even in the case of an operator as simple as the identity function $\mathrm{id}_V(\mathbf{v}) = \mathbf{v}$, the matrix of a linear operator can be useful and interesting.

Definition 5.3. Let $\mathrm{id}_V : V_C \to V_B$ be the identity function of V. Then the matrix $[\mathrm{id}_V]_{C,B}$ is called the *change of basis* matrix from the basis B to the basis C.

Observe that this definition is consistent with the discussion in Section **??** since Equation (2) shows us that for any vector $\mathbf{v} \in V$,

$$[\mathbf{v}]_B = [\mathrm{id}_V(\mathbf{v})]_B = [\mathrm{id}_V]_{C,B}[\mathbf{v}]_C.$$

Also note that change of basis matrix from basis B to basis C is quite easy if B is a standard basis: simply form the matrix that has the vectors of C listed as its columns.

Example 5.4. Let $V = \mathbb{R}^2$. What is the change of basis matrix from standard basis $B = \{\mathbf{e}_1, \mathbf{e}_2\}$ to the basis $C = \left\{\mathbf{v}_1 = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix}\right\}$?

Solution. We see that

$$\begin{aligned} \mathbf{v}_1 &= \cos\theta\,\mathbf{e}_1 + \sin\theta\,\mathbf{e}_2 \\ \mathbf{v}_2 &= -\sin\theta\,\mathbf{e}_1 + \cos\theta\,\mathbf{e}_2. \end{aligned}$$

Compare these equations to (1) and we see that the change of basis matrix is

$$[\mathrm{id}_V]_{C,B} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = R(\theta).$$

As predicted, we only have to form the matrix that has the vectors of C listed as its columns. Now compare this to the discussion following Example **??**. □

Next, suppose that $T : V \to W$ and $S : U \to V$ are linear operators. Can we relate the matrices of T, S and the function composition of these operators, $T \circ S$? The answer to this question is a very fundamental fact.

Theorem 5.5. *If* $U_D \xrightarrow{S} V_C \xrightarrow{T} W_D$, *then* $[T \circ S]_{D,C} = [T]_{B,C}[S]_{D,B}$.

Proof. Given a vector $\mathbf{u} \in U$, set $\mathbf{v} = S(\mathbf{u})$. With the notation of Equation (2) we have that $[T \circ S]_{D,C}[\mathbf{u}]_D = [(T \circ S)(\mathbf{u})]_C$ and by definition of function composition that $(T \circ S)(\mathbf{u}) = T(S(\mathbf{u})) = T(\mathbf{v})$. Therefore

$$[T \circ S]_{D,C}[\mathbf{u}]_D = [(T \circ S)(\mathbf{u})]_C = [T(S(\mathbf{u}))]_C = [T(\mathbf{v})]_C.$$

On the other hand, Equation (2) also implies that $[T(\mathbf{v})]_C = [T]_{B,C}[\mathbf{v}]_B$ and $[S(\mathbf{u})]_B = [S]_{D,C}[\mathbf{u}]_D$. Hence, we deduce that

$$[T \circ S]_{D,C}[\mathbf{u}]_D = [T]_{B,C}[\mathbf{v}]_B = [T]_{B,C}[S]_{D,C}[\mathbf{u}]_D.$$

If we choose $\mathbf{u}$ so that $\mathbf{e}_j = [\mathbf{u}]_D$, where $\mathbf{e}_j$ is the jth standard vector, then we obtain that the jth columns of left and right hand side agree for all j. Hence the matrices themselves agree, which is what we wanted to show. □

We can now also see exactly what happens when we make a change of basis in the domain and target of a linear operator and recalculate the matrix of the operator. Specifically, suppose that $T : V \to W$ and that B, B' are bases of V and C, C' are bases of W. Let P and Q be the change of basis matrices from B' to B and C' to C, respectively. We calculate that that Q^{-1} is the change of basis matrix from C to C'. Identify a matrix with its operator action by multiplication and we have a chain of operators

$$V_{B'} \overset{\mathrm{id}_V}{\to} V_B \overset{T}{\to} W_C \overset{\mathrm{id}_W}{\to} W_{C'}.$$

Application of the Theorem shows that

$$[T]_{B',C'} = [\mathrm{id}_W]_{C,C'} [T]_{B,C} [\mathrm{id}_V]_{B',B} = Q^{-1}[T]_{B,C}P.$$

Corollary 5.6. *Let $T : \mathbb{R}^n \to \mathbb{R}^m$ be a linear operator, B a basis of $\mathbb{R}^n$ and C a basis of $\mathbb{R}^m$. Let P and Q be the change of basis matrices from the standard bases to the bases B and C, respectively. If A is the matrix of T with respect to the standard bases and M the matrix of T with respect to the bases B and C, then*

$$M = Q^{-1}AP.$$

This Corollary gives us an interesting view of the SVD: given $m \times n$ matrix A there exists unitary U, V such that $U^*AV = \Sigma U^{-1}AV$ with Σ $m \times n$ diagonal with non-negative entries down the diagonal means that with respect to suitable orthonormal bases of $\mathbb{C}^n$ and $\mathbb{C}^m$ the action of the matrix multiplication operator T_A amounts to non-negative scalar multiplication, i.e., $T(\mathbf{v}_j) = \sigma_j \mathbf{u}_j$, $j = 1, \ldots, n$.

Example 5.7. Given the linear operator $T : \mathbb{R}^4 \to \mathbb{R}^2$ by the rule

$$T(x_1, x_2, x_3, x_4) = \begin{bmatrix} x_1 + 3x_2 - x_3 \\ 2x_1 + x_2 - x_4 \end{bmatrix},$$

find the standard matrix of T.

Solution. We see that

$$T(\mathbf{e}_1) = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, T(\mathbf{e}_2) = \begin{bmatrix} 3 \\ 1 \end{bmatrix}, T(\mathbf{e}_3) = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, T(\mathbf{e}_4) = \begin{bmatrix} 0 \\ -1 \end{bmatrix}.$$

Since the standard coordinate vector of a standard vector is itself, we have

$$[T] = \begin{bmatrix} 1 & 3 & -1 & 0 \\ 2 & 1 & 0 & -1 \end{bmatrix}.$$

□

Example 5.8. With T as above, find the matrix of T with respect to the domain basis $B = \{(1,0,0,0), (1,1,0,0), (1,0,1,0), (1,0,0,1)\}$ and range basis $C = \{(1,1), (1,-1)\}$

Solution. Let A be the matrix of the previous example, so it represents the standard matrix of T. Let $B' = \{(1,0,0,0), (0,1,0,0), (0,0,1,0), (0,0,0,1)\}$ and $C' = \{(1,0), (0,1)\}$ be the standard bases for the domain and target of T. Then we have

$$A = [T] = [T]_{B',C'}.$$

Further, we only have to stack columns of B and C to obtain change of basis matrices

$$P = [\mathrm{id}_{\mathbb{R}^4}]_{B,B'} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } Q = [\mathrm{id}_{\mathbb{R}^2}]_{C,C'} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Now apply Corollary 5.6 to obtain that

$$\begin{aligned} [T]_{B,C} &= Q^{-1}AP \\ &= -\frac{1}{2}\begin{bmatrix} -1 & -1 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} 1 & 3 & -1 & 0 \\ 2 & 1 & 0 & -1 \end{bmatrix}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \frac{3}{2} & \frac{7}{2} & 1 & 1 \\ -\frac{1}{2} & \frac{1}{2} & -1 & 0 \end{bmatrix}. \end{aligned}$$

□

Matlab Introduction

Download and open up the file "MatlabLecture-447.pdf". Then start up Matlab from the Programs menu. We will do some of the highlights. A more detailed pass through this tutorial is left as an exercise to those students who need it.

Part II: QR Factorization and Least Squares

6. Projectors

(9/7/06) We discuss the idea of projectors from three perspectives: geometrical, analytical and matrix. As in the text, we develop matrix properties in detail and discuss two ways of constructing projectors. One is derived from normal equations, while the other is derived from classical Gram-Schmidt, which gives a sneak preview of Lecture 7. We want to consider some basic numerical algebra notation and ideas, namely flop counts and basic algorithms. This requires some supplemental material that belongs in every applied numerical analyst's vocabulary.

The BLAS

No, it is not a mental condition. From netlib:

The BLAS (Basic Linear Algebra Subprograms) are routines that provide standard building blocks for performing basic vector and matrix operations. The Level 1 BLAS perform scalar, vector and vector-vector operations, the Level 2 BLAS perform matrix-vector operations, and the Level 3 BLAS perform matrix-matrix operations. Because the BLAS are efficient, portable, and widely available, they are commonly used in the development of high quality linear algebra software, LAPACK for example.

From the gnu website:

There are three levels of blas operations,

Level 1

Vector operations, e.g. y = \alpha x + y
Level 2
Matrix-vector operations, e.g. y = \alpha A x + \beta y
Level 3
Matrix-matrix operations, e.g. C = \alpha A B + C
Each routine has a name which specifies the operation, the type of matrices involved and their precisions.

Four Precisions:
S
single real
D
double real
C
single complex
Z
double complex

Common Operations:
DOT
scalar product, x^T*y
AXPY
vector sum, \alpha*x + y
MV
matrix-vector product, A*x
SV
matrix-vector solve, inv(A)*x
MM
matrix-matrix product, A*B
SM
matrix-matrix solve, inv(A)*B

Thus, e.g., SAXPY is single precision real AXPY.
Types of Matrices:
GE
general
GB
general band
SY
symmetric
SB
symmetric band
SP
symmetric packed
HE

hermitian
HB
hermitian band
HP
hermitian packed
TR
triangular
TB
triangular band
TP
triangular packed

Thus, for example, the name sgemm stands for single-precision general matrix-matrix multiply and zgemm stands for double-precision complex matrix-matrix multiply.

LAPACK

From netlib:

LAPACK is written in Fortran77 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision.

If you're uncertain of the LAPACK routine name to address your application's needs, check out the LAPACK Search Engine.

The original goal of the LAPACK project was to make the widely used EISPACK and LINPACK libraries run efficiently on shared-memory vector and parallel processors. On these machines, LINPACK and EISPACK are inefficient because their memory access patterns disregard the multi-layered memory hierarchies of the machines, thereby spending too much time moving data instead of doing useful floating-point operations. LAPACK addresses this problem by reorganizing the algorithms to use block matrix operations, such as matrix multiplication, in the innermost loops. These block operations can be optimized for each architecture to account for the memory hierarchy, and so provide a transportable way to achieve high efficiency on diverse modern machines. We use the term "transportable" instead of "portable" because, for fastest possible performance, LAPACK requires that highly optimized block matrix operations be already implemented on each machine.

LAPACK routines are written so that as much as possible of the computation is performed by calls to the Basic Linear Algebra Subprograms (BLAS). While LINPACK and EISPACK are based on the vector operation kernels of the Level 1 BLAS, LAPACK was designed at the outset to exploit the Level 3 BLAS – a set of specifications for Fortran subprograms that do various types of matrix multiplication and the solution of triangular systems with multiple right-hand sides. Because of the coarse granularity of the

Level 3 BLAS operations, their use promotes high efficiency on many high-performance computers, particularly if specially coded implementations are provided by the manufacturer.

Highly efficient machine-specific implementations of the BLAS are available for many modern high-performance computers. For details of known vendor- or ISV-provided BLAS, consult the BLAS FAQ. Alternatively, the user can download ATLAS to automatically generate an optimized BLAS library for the architecture. A Fortran77 reference implementation of the BLAS in available from netlib; however, its use is discouraged as it will not perform as well as a specially tuned implementation.

Four Precisions:
S REAL
D DOUBLE PRECISION
C COMPLEX
Z COMPLEX*16 or DOUBLE COMPLEX

Types of Matrices:
Most of these two-letter codes apply to both real and complex matrices; a few apply specifically to one or the other.
BD bidiagonal
DI diagonal
GB general band
GE general (i.e., unsymmetric, in some cases rectangular)
GG general matrices, generalized problem (i.e., a pair of general matrices)
GT general tridiagonal
HB (complex) Hermitian band
HE (complex) Hermitian
HG upper Hessenberg matrix, generalized problem (i.e a Hessenberg and a triangular matrix)
HP (complex) Hermitian, packed storage
HS upper Hessenberg
OP (real) orthogonal, packed storage
OR (real) orthogonal
PB symmetric or Hermitian positive definite band
PO symmetric or Hermitian positive definite
PP symmetric or Hermitian positive definite, packed storage
PT symmetric or Hermitian positive definite tridiagonal
SB (real) symmetric band
SP symmetric, packed storage
ST (real) symmetric tridiagonal
SY symmetric
TB triangular band
TG triangular matrices, generalized problem (i.e., a pair of triangular matrices)
TP triangular, packed storage
TR triangular (or in some cases quasi-triangular)

TZ trapezoidal
UN (complex) unitary
UP (complex) unitary, packed storage

Naming Conventions:

The naming scheme of each LAPACK routine is a coded specification of its function (within the very tight limits of standard Fortran 77 6-character names). All driver and computational routines have names of the form XYYZZZ, where for some driver routines the 6th character is blank.

The first letter, X, indicates the data type S, D, C or Z.

The next two letters, YY, indicate the type of matrix (or of the most significant matrix).

The last three letters ZZZ indicate the computation performed. For example, SGEBRD is a single precision routine that performs a bidiagonal reduction (BRD) of a real general matrix. Their meanings are fully explained in the LAPACK Users' Guide.

Indexes of routine names are available in four data types. An individual routine or routine plus dependencies can be selected. For brevity, only driver and computational routine names are listed on these indexes. Auxiliary routines are not listed, but can be downloaded via ftp in the respective subdirectories.

7. QR Factorization

(9/7/06) We develop both the reduced and full QR factorization by way of the classical Gram-Schmidt algorithm. We also discuss issues of complexity via flop counts for a few BLAS routines.

8. Gram-Schmidt Orthogonalizaton

(9/12/06) We discuss both classical and modified forms of Gram-Schmidt and complexity. Here is the non-algorithmic Math 314 style statement of the algorithm: given linearly independent vectors $a_1, \ldots, a_n \in \mathbb{C}^m$, set

$$\begin{aligned} \widetilde{q}_j &= a_j - (q_1^* a_j)\, q_1 - \cdots - \left(q_{j-1}^* a_j\right) q_{j-1} \\ q_j &= \frac{\widetilde{q}_j}{\|\widetilde{q}_j\|}, \quad j = 1, \ldots, n. \end{aligned}$$

For the record, here are the two Gram-Schmidt algorithms that implement the above equations:

Algorithm 8.1. *(Classical G-S) Input* $a_1, a_2, \ldots, a_n$, $m = \text{length}\,(a_i)$

for $j = 1 : n$
  $v_j = a_j$
  for $i = 1 : (j-1)$
    $r_{ij} = q_i^* a_j$
    $v_j = v_j - r_{ij} q_i$
  end
  $r_{jj} = \|v_j\|_2$
  $q_j = v_j / r_{jj}$

end
Return $q_1, \ldots, q_n$, $R = [r_{ij}]$

Algorithm 8.2. *(Modified G-S) Input* $a_1, a_2, \ldots, a_n$, $m = \text{length}\,(a_i)$
for $i = 1 : n$
  $v_i = a_i$
end
for $i = 1 : n$
  $r_{ii} = \|v_i\|$
  $q_i = v_j / r_{ii}$
  for $j = (i+1) : n$
    $r_{ij} = q_i^* v_j$
    $v_j = v_j - r_{ij} q_i$
end
Return $q_1, \ldots, q_n$, $R = [r_{ij}]$

9. Matlab

(9/14/06) We construct the function mgs.m from cgs.m and perform Experiment 2. First, we need to copy the files in the subdirectory MatlabTools of the class home directory to a suitable place in our home files. When we run Matlab, first change the working directory to this directory.

We will construct a random orthogonal matrices U, V of size 80 and singular values $\sigma_i = 2^{-i}$, $i = 1, \ldots, 80$, then let $A = U\Sigma V^*$. Having done so, we can expect that the singular values of A to be reasonably approximated by the diagonal entries of R where $A = QR$ is the QR factorization of A. Reason:

$$A = 2^{-1} u_1 v_1^* + 2^{-2} u_2 v_2^* + \cdots + 2^{-80} u_{80} v_{80}^*$$

so that the jth column of A is

$$a_j = 2^{-1} u_1 v_{j1} + 2^{-2} u_2 v_{j2} + \cdots + 2^{-80} u_{80} v_{j,80}.$$

But the $v_{j,k}$'s are random, so we expect of a similar magnitude of about $80^{-1/2} \approx 0.1$. So we expect that $q_1 \approx u_1$ and $r_{11} \approx 2^{-1} 80^{-1/2}$, then $q_2 \approx u_2$ and $r_{22} \approx 80^{-1/2}$, etc. Here is the code to execute, once we are in the right place:

```
randn('state',0);
[U,X] = qr(randn(80));
[V,X]= qr(randn(80));
S = diag(2.^(-1:-1:-80));
A = U*S*V;
[QC,RC] = clgs(A);
% Pause here and create function mgs
edit clgs
[QM,RM] = mgs(A);
```

```
% Let's hold Matlab's feet to the fire too
[QML,RML] = qr(A);
semilogy(diag(S));
hold on
semilogy(diag(RC));
semilogy(diag(RM));
semilogy(diag(RML));
norm(QC'*QC)
norm(QM'*QM)
norm(QML'*QML)
% now construct the matrix of Exercise 9.2 and play with it
help Tridiag
A = Tridiag(20,0,1,2)
```

10. Householder Triangularization

(9/14/06) We examine Householder reflectors. This is used to develop a QR triangularization algorithm and we discuss complexity, stability and implementation issues.

11. Least Squares Problems

We lay foundations for normal equations, give some examples and three different methods for solving these problems.

Part III: Conditioning and Stability

12. Conditioning and Condition Numbers

We discuss conditioning in a fairly general context of a computable function, then specialize to linear systems and calculate a classical error bound in terms of condition numbers.

13. Floating Point Arithmetic

We follow text and discuss some of the limitations of floating point arithmetic. We also discuss the two fundamental axioms that we expect our machine computations to satisfy, namely the Rounding Axiom (RA) and the Fundamental Axiom of Floating Point Arithmetic (FAFPA).

14. Stability

We discuss absolute and relative stability and settle on the latter as suitable for numerical analysis. Also discuss the "big-Oh" notation.

15. More on Stability

Discuss accuracy, stability and backward stability, and do examples, including the inner and outer products. Conclude with the powerful forward error estimate theorem that connects backward stability, conditioning and accuracy.

16. Stability of Householder Triangularization

We show the backward stability of this algorithm under suitable hypotheses.

17. Stability of Backward Substitution

This was used for Section 16. We work through the details.

18. Conditioning of Least Squares Problems

We discuss the mathematical (forward) stability, i.e., sensitivity, of the least squares problem.

19. Stability of Least Squares Algorithms

We discuss the hypothesis of backward stability by examining some numerical computations on Matlab that suggest it indirectly by way of the forward error estimate theorem.

Here are the Matlab commands we'll use:

% Build a least squares fitting of $f(x) = e^{\sin(4x)}$ on $[0, 1]$ by a polynomial of degree 14 through least squares interpolation of 100 equally spaced points.

Setup:

```
m = 100; n = 15;
t = (0:m-1)'/(m-1);
A = [ ];
for i = 1:n
 A=[A t.^(i-1)];
end
b = exp(sin(4*t));
b = b/2006.787453080206;
```

This odd calculation makes the true value of x_{15} equal to 1 rather than its true value, the denominator above. Makes comparisons easier.

```
x = A\b; y = A*x;
kappa = cond(A)
theta = asin(norm(b-y)/norm(b))
eta = norm(A)*norm(x)/norm(y)
b2y = 1/cos(theta)
b2x = kappa/(eta*cos(theta))
A2y = kappa/cos(theta)
A2x = kappa + kappa^2*tan(theta)/eta
```

Householder Triangularization:

```
[Q,R] = qr(A,0);
x = R\(Q'*b);
x(15)
```

```
  % Now do the implicit version of Q:
  [Q,R] = qr([A b],0);
  Qb = R(1:n,n+1);
  x = R\Qb;
  x(15)
  % or just let Matlab do its job, which uses extra QR plus column
pivoting:
  x = A\b;
  x(15)
```
Gram-Schmidt:
```
  [Q,R] = mgs(A);
  x = R\(Q'*b);
  x(15)
  % However, with a little modification:
  [Q,R] = mgs([A b]);
  Qb = R(1:n, n+1);
  R = R(1:n, 1:n);
  x = R\Qb;
  x(15)
```
Normal Equations:
```
  x = (A'*A)\(A'*b);
  x(15)
  % This warrants discussion...BTW, a satisfactory
  % backward stable method of solution is Cholesky, but...
  % think about the condition number of the coefficient matrix.
```
SVD:
```
  [U,S,V] = svd(A,0); % reduced svd
  x = V*(S\(U'*b));
  x(15)
```

20. Gaussian Elimination

At last. Most numerical linear algebra texts begin with this subject. Trefethen and Bau wanted to loosen the tight connection in most peoples' minds between Gaussian elimination – with its usual error analysis and finite algorithmic feel – and the larger subject of numerical linear algebra. And sure enough, we've had plenty to do up to now. We discuss the basics of the algorithm and emphasize its formulation in terms of an LU factorization. Also, we consider complexity of the algorithm with a flop count analysis.

21. Pivoting

We discuss pivoting strategies and algorithms, including the method of overwriting the input matrix A with the L, U parts of A in a permuted order, returning this matrix plus a permuted index array.

22. Stability of Gaussian Elimination

We discuss stability, and some worse case examples of column pivoting. Also, the statistical graphs in this section.

23. Cholesky Factorization

We prove the existence of a Cholesky factorization for Hermitian matrices and describe the overwriting algorithm for Cholesky. Also noted are the fact that this algorithm is backward stable and that flop count is about half ($m^3/3$) of what one would get with Gaussian elimination.

24. Eigenvalue Problems

This is mainly a review of eigenvector-eigenvalue notation and basic results. Diagonalizability is discussed, along with defective matrices. The Jordan canonical form theorem is discussed. We define Jordan blocks as follows:JA Jordan block is defined to be a $d \times d$ matrix of the form

$$J_d(\lambda) = \begin{bmatrix} \lambda & 1 & & \\ & \lambda & \ddots & \\ & & \ddots & 1 \\ & & & \lambda \end{bmatrix},$$

where the entries off the main diagonal and first superdiagonal are understood to be zeros. This matrix is very close to being a diagonal matrix. Its true value comes from the following classical theorem.

Theorem 24.1. *(Jordan Canonical Form Theorem) A square matrix A is similar to a block diagonal matrix that consists of Jordan blocks down the diagonal (i.e., there is an invertible matrix P such that $P^{-1}AP$ has this form.) Moreover, these blocks are uniquely determined by A up to order.*

25. Overview of Eigenvalue Problems

We observe that the traditional idea of finding the characteristic polynomial and solving it is a loser, and in fact numerical eigenvalue methods can be used effectively to find solutions of roots of polynomials by way of the companion matrix. The two phases of eigensystem computation, namely (1) Reduction to a manageable form such as upper Hessenberg and (2) Finding eigensystems for such matrices are briefly touched upon.

We also discuss and experiment with the notion mentioned earlier, namely that defectiveness is not easy to detect and involves some subtle ideas.

Start with the following problem: What is the rank of $J_d(\lambda) - \mu I$?

Answer: d if $\mu \neq \lambda$ and $d - 1$ otherwise.

Now remember that rank is invariant under a similarity transformation. Therefore questions about rank $\left((A-\lambda I)^k\right)$, $k = 0, 1, \ldots$, may be answered by examining J^k, where J is the Jordan canonical form of A, and conversely.

Question: how much deficiency in rank does a single Jordan block contribute to rank $((A-\lambda I))$? To rank $\left((A-\lambda I)^2\right)$?

Now login, start up Matlab and cd to the correct directory for your class work. Grab a copy of `lect25.m` if you haven't already done so. Now do the following, and let's comment on each output before proceeding to the next line.

```
lec25
A
cond(A)
% Do we believe that A is diagonalizable?
[P,D] = eig(A)
norm(A-P*D*inv(P))
cond(P)
% Well, ....?
% Let's have a look at the eigenvalues of A
format long
d = sort(diag(D))
% So try one of our candidates:
lam = xxx
rank(A-lam*eye(10))
rank((A-lam*eye(10))^2)
% Repeat with the others, then nail the complex roots.
```

26. Reduction to Hessenberg, Tridiagonal or Bidiagonal Form

We exhibit the algorithms, mainly pictorially, leaving the exact write-up to the text. We also discuss backward stability (it's there) and flop counts.

27. Rayleigh Quotient, Inverse Iteration

Follow the text discussion of these standard routines, confinining attention mostly to real symmetric matrices.

Here is a test matrix:

$$A = \begin{bmatrix} -8 & -5 & 8 \\ 6 & 3 & -8 \\ -3 & 1 & 9 \end{bmatrix}.$$

Now we ask three questions about A:

(1) How can we get a ballpark estimate of the location of the eigenvalues of A?

(2) How can we estimate the *dominant* eigenpair $(\lambda, \mathbf{x})$ of A? (Recall that "dominant" means that λ is larger in absolute value than any other eigenvalue of A.)

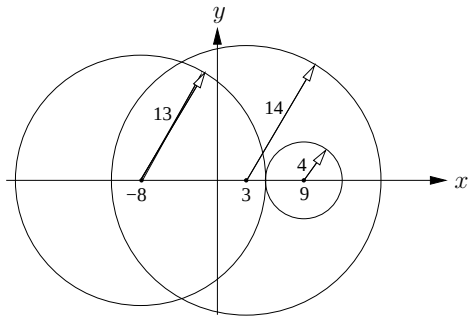

FIGURE 1. Gershgorin disks for A.

(3) Given a good estimate of any eigenvalue λ of A, how can we improve the estimate and compute a corresponding eigenvector?

Theorem 27.1. *(Gerschgorin Disk Theorem) Let $A = [a_{ij}]$ be an $n \times n$ matrix and define disks D_j in the complex plane by*

$$\begin{aligned} r_j &= \sum_{\substack{k=1 \\ k \neq j}}^{n} |a_{jk}|, \\ D_j &= \{ z \,|\, |z - a_{jj}| \leq r_j \}. \end{aligned}$$

(1): *Every eigenvalue of A is contained in some disk D_j.*
(2): *If k of the disks are disjoint from the others, then exactly k eigenvalues are contained in the union of these disks.*

Example 27.2. Apply the Gershgorin circle theorem to the test matrix A and sketch the resulting Gershgorin disks.

Solution. The disks are easily seen to be

$$\begin{aligned} D_1 &= \{ z \,|\, |z + 8| \leq 13 \}, \\ D_2 &= \{ z \,|\, |z - 3| \leq 14 \}, \\ D_3 &= \{ z \,|\, |z - 9| \leq 4 \}. \end{aligned}$$

A sketch of them is provided in Figure 1. □

Now we turn to question (2). One answer to it is contained in the following algorithm, known as the *power method*.

Algorithm (Power Method) To compute an approximate eigenpair $(\lambda, \mathbf{x})$ of A with $\|\mathbf{x}\| = 1$ and λ the dominant eigenvalue.

(1): Input an initial guess $\mathbf{x}_0$ for $\mathbf{x}$
(2): For $k = 0, 1, \ldots$ until convergence of $\lambda^{(k)}$'s:
(a) $\mathbf{y} = A\mathbf{x}_k$,
(b) $\mathbf{x}_{k+1} = \dfrac{\mathbf{y}}{\|\mathbf{y}\|}$,
(c) $\lambda^{(k+1)} = \mathbf{x}_{k+1}^T A \mathbf{x}_{k+1}$.

You might notice also that the argument doesn't require the initial guess to be a real vector. Complex vectors are permissible.

Finally, we turn to question (3). One answer to it is contained in the following algorithm, known as the *inverse iteration method*.

Algorithm (Inverse Iteration) To compute an approximate eigenpair $(\lambda, \mathbf{x})$ of A with $\|\mathbf{x}\| = 1$.

(1): Input an initial guess $\mathbf{x}_0$ for $\mathbf{x}$ and a *close* approximation $\mu = \lambda_0$ to λ.

(2): For $k = 0, 1, \ldots$ until convergence of the $\lambda^{(k)}$'s:

(a): $\mathbf{y} = (A - \mu I)^{-1}\mathbf{x}_k$,

(b): $\mathbf{x}_{k+1} = \dfrac{\mathbf{y}}{\|\mathbf{y}\|}$,

(c): $\lambda^{(k+1)} = \mathbf{x}_{k+1}^T A \mathbf{x}_{k+1}$.

Notice that the inverse iteration method is simply the power method applied to the matrix $(A - \mu I)^{-1}$. In fact, it is sometimes called the inverse power method. The scalar μ is called a *shift*. Here is the secret of success for this method: we assume that μ is closer to a definite eigenvalue λ of A than to any other eigenvalue. But we don't want too much accuracy! We need $\mu \neq \lambda$. The eigenvalues of $(A - \mu I)^{-1}$ are of the form $1/(\sigma - \mu)$, where σ runs over the eigenvalues of A. Since μ is closer to λ than to any other eigenvalue of A, the eigenvalue $1/(\lambda - \mu)$ is the dominant eigenvalue of $(A - \mu I)^{-1}$, which is exactly what we need to make the power method work on $(A - \mu I)^{-1}$. Indeed, if μ is *very* close (but not equal!) to λ, convergence should be very rapid.

28. QR Algorithm without Shifts

(11/14/06) Follow text discussion and note the connection between simultaneous iteration and the QR algorithm.

Algorithm (QR)

(1): Input $A^{(0)} = A$, $m \times m$ matrix.

(2): For $k = 1, 2, \ldots$

(a) $Q^{(k)}R^{(k)} = A^{(k-1)}$ % QR factorization

(b) $A^{(k)} = R^{(k)}Q^{(k)}$

A more "practical" version:

Algorithm (Practical QR)

(1): Input $A^{(0)} = A$, $m \times m$ matrix.

(2): $A^{(0)} = {Q^{(0)}}^T A Q^{(0)}$ % $A^{(0)}$ is upper Hessenberg form for A

(2): For $k = 1, 2, \ldots$

(a) Select shift μ_k % e.g., $\mu_k = A_{r,r}^{(k-1)}$, r last index with $A_{r-1,r}^{(k-1)} \neq 0$.

(b) $R^{(k)}Q^{(k)} = A^{(k)} - \mu_k I$ % QR factorization

(c) $Q^{(k)}R^{(k)} = A^{(k-1)} + \mu_k I$

(d) If any subdiagonal element $A_{j-1,j}^{(k)} \approx 0$, set it equal to zero to decouple $A = \begin{bmatrix} A_1 & * \\ & A_2 \end{bmatrix}$ and continue algorithm on A_1 and A_2 portions.

How expensive? This could be HUGELY expensive, given that the cost of a single QR factorization is approximately $\frac{4}{3}m^3$ and explicit formation of Q costs another $\frac{4}{3}m^3$.

The problem is that standard Householder QR involves large scale zeroing out. We need something a bit more delicate. One approach is to use Givens rotations to zero out specific entries. Another is to use a scaled down version of Householder transformations to get the job done.

29. QR Algorithm with Shifts

(11/16/06) Continue text discussion and perform numerical experiments with a non-symmetric and symmetric matrix.

Numerical experiment test matrices:

```
% Script:  lec25.m
% Description:  Generates three matrices for
% analysis of iterative algorithms
disp('A is a non-symmetric matrix with fairly simple Gerschgorin disks and
eigensystem:')
A = [-8 -5 8; 6 3 -8; -3 1 9]
disp('B is a more complex found in exercises:')
m = 11;
B = toeplitz([1;-ones(m-1,1)],[1,zeros(1,m-1)]); B(:,m)=ones(m,1)
disp('C = B^T*B is a symmetric matrix.');
C = B'*B;
disp('D is a random matrix in the spirit of Exercise 12.3,')
disp('but with an inserted (1,1)th entry.')
randn('state',0)
m = 100;
D = randn(m,m)/sqrt(m);
D(1,1) = 1.5;
```

In all experiments below, let's start by naming E as the experimental matrix.

Apply the power iteration algorithm manually to each matrix, starting with a random vector. In the non-symmetric cases of A and D, also try a complex starting point. Keep count of the number of iterations you do before stopping. You can do this conveniently by introducing a counter.

Next, try the inverse power rule with some guess for an eigenvalue other than the ones you just found.

Finally, examine the QR algorithm and follow it up with an experiment with the shifted QR.

33-34. The Arnoldi Iteration and Eigenvalues

(11/21/06) We will do a brief introduction to iterative methods followed by a discussion of the Arnoldi iteration.

Question: Why iterate when we have effective direct solvers of

$$Ax = b$$

which terminate in a well defined finite number of steps?

Answers: (Some)

(1) Speed: If A is very large (try $m = 10^6$), $\mathcal{O}(m^3)$ is very intimidating.
(2) Storage: If we uncritically make space for A we're looking at $8m^2$ bytes, which again could be a lot.

Disclaimer: Iterative methods are not a cure-all for either of the above problems. There are, for example, very good direct solver routines that take advantage of properties like a sparse matrix (enough zeros in it that you should pay attention to them), and sparse matrix technology that gives very effective storage of data elements.

Classical Iteration Theory

One topic that I will discuss that is not in the text is the subject of iterative methods generated by splittings. Here is the basic idea: to solve the linear system

$$Ax = b$$

we "split" A into two parts $A = B - C$. Here B should be a "simple" and easily inverted matrix. Rewrite the equation as

$$Bx = Cx + b.$$

This is a so-called regular splitting, and it inspires the iterative scheme

$$Bx^{(k+1)} = Cx^{(k)} + b$$

or

$$x^{(k+1)} = Gx^{(k)} + d,$$

where $G = B^{-1}C$ and $d = B^{-1}b$. This is an example of a stationary (because G doesn't change from step to step) one step (because we only use the output of the previous step) iteration method.

The basic theory is very simple: this method converges for all initial guesses $x^{(0)}$ if and only if $\rho(G) < 1$, in which case convergence is linear and the asymptotic rate of convergence is $\rho(G)$. Of course, there is vastly more to it than this, and huge amounts of research have been done on this topic.

Here are some of the classical splittings. In some of the following we use this notation

$$A = D - L - U,$$

where $-L$ is the (strictly) lower triangular part of A, D is the diagonal and $-U$ is the (strictly) upper triangular part. For the iterations that use D, it is required that D have nonzero diagonal entries.

- Richardson iteration: Write $\gamma A = I - (I - \gamma A)$ and obtain the iterative scheme

  $$x^{(k+1)} = (I - \gamma A)\, x^{(k)} + \gamma b, \;\; G = (I - \gamma A)\,.$$

  Note that if we allow the acceleration parameter γ to vary at each step (as Richardson actually did), we have a nonstationary iterative method.
- Jacobi iteration:

  $$Dx^{(k+1)} = (L + U)\, x^{(k)} + b, \;\; G = D^{-1}(L + U)\,.$$

- Gauss-Seidel iteration:

  $$(D - L)\, x^{(k+1)} = Ux^{(k)} + b, \;\; G = (D - L)^{-1} U.$$

- Gauss-Seidel SOR (or just SOR):

$$\frac{1}{\omega}\left(D-\omega L\right)x^{(k+1)}=\frac{1}{\omega}\left(\left(1-\omega\right)D+\omega U\right)x^{(k)}+b,\ \ G=\left(D-\omega L\right)^{-1}\left(\left(1-\omega\right)D+\omega U\right).$$

Note that $\omega=1$ is simply Gauss-Seidel. Bear in mind that that what we have above are the theoretical tools for analyzing the methods. The great virtue of many of these is that they can be calculated in place with minimal additional storage requirements. Furthermore, their coding is extremely simple, which is why they are popular to this day among engineers and scientists. For example, here is a practical formulation of GS-SOR:

$$\begin{aligned} y_i^{(k+1)} &= \left(b-\sum_{j=1}^{i-1}a_{ij}x_j^{(k+1)}-\sum_{j=i+1}^{m}a_{ij}x_j^{(k)}\right)/a_{ii} \\ x_i^{(k+1)} &= x_i^{(k)}+\omega\left(y_i^{(k+1)}-x_i^{(k)}\right). \end{aligned}$$

Better yet, here it is in one line:

$$x_i^{(k+1)} = \left(1-\omega\right)x_i^{(k)}+\frac{\omega}{a_{ii}}\left(b-\sum_{j=1}^{i-1}a_{ij}x_j^{(k+1)}-\sum_{j=i+1}^{m}a_{ij}x_j^{(k)}\right).$$

It's remarkably simple to implement this formulation. The problem is determining the acceleration parameter ω... or whether the method will converge at all.

A few well-known facts:

(1) If G_ω is the iteration matrix for GS-SOR, then $\rho\left(G_\omega\right)\geq\left|1-\omega\right|$. Hence the method does not converge for $\omega\notin(0,2)$.
(2) If A is strictly row (or column) diagonally dominant, GS-SOR and the Jacobi method will converge for $\omega\leq 1$.
(3) If A is HPD, then GS-SOR converges for $0<\omega<2$.

Example. This is a famous test problem for iterative methods. Poisson's equation for a two dimensional unit square

$$-\triangle u\equiv-\frac{\partial^2 u\left(x,y\right)}{\partial x^2}-\frac{\partial^2 u\left(x,y\right)}{\partial y^2}=f\left(x,y\right),\ 0\leq x,y\leq 1,$$

with the Dirichlet condition $u\left(x,y\right)=g\left(x,y\right)$ on the boundary of the square, represents a heat equation for a two dimensional insulated plate whose boundaries are kept at constant temperatures and which has an internal heat source whose value may vary from point to point. This is known to be a well-posed problem with a smooth solution $u\left(x,y\right)$.

This problem can be discretized on a uniform grid of points (x_i,y_j), $0\leq i,j\leq N+1$ with equal spacing $h=1/\left(N+1\right)$ in the x and y directions leads to an approximation

$$u\left(x_i,y_j\right)=u\left(ih,jh\right)\approx u_{ij}$$

where, by using centered second derivative approximations, we obtain the system

$$\frac{-u_{i-1,j}+2u_{i,j}-u_{i+1,j}}{h^2}+\frac{-u_{i,j-1}+2u_{i,j}-u_{i,j+1}}{h^2}=f\left(x_i,y_j\right)=f_{i,j},$$

that is,

$$-u_{i-1,j} - u_{i+1,j} + 4u_{i,j} - u_{i,j-1} - u_{i,j+1} = h^2 f_{i,j},\ 1 \le i, j \le N. \tag{3}$$

A practical observation: even if we solve this linear algebra problem exactly, we will have $\mathcal{O}(h^2)$ error as compared to the true solution to the model problem. This is not due to any numerical error, but rather the mathematical truncation error approximating derivatives by differences. Therefore, it might be a bit silly to want 12 digits of accuracy in the solution of the discretized problem when, say, perhaps at most 4 digits of accuracy to the real problem would be obtained if we were to solve the discretized system exactly. Thus, iterative methods might make more sense in this context. We can easily code up a GS-SOR for this problem leaving the variables in place in a $(N+2)\times(N+2)$ matrix. This is a big system of N^2 equations in N^2 unknowns. Consider its size if, say, we want $h = 0.01$.

Of course, there is an immediate issue: how to order the variables? Here is a convenient first formulation of the problem. Notice that if we let $U = [u_{ij}]_{N+2,N+2}$ be the array of node values of the numerical solution, then although U is not a vector, this is a very natural arrangement of the variables, since it matches the spatial arrangement of the arguments (x_i, y_j). Let's try to keep this form.

To this end, observe that the Laplace operator is a sum of two differentiation operators, one of which works on the rows of U (second derivative with respect to x) and the other of which works on columns of U (second derivative with respect to y). So let's focus on one space derivative. Suppose values of a sufficiently smooth function $f(t)$ are known at points $f_0, f_1, \ldots, f_{N+1}$, where $f_j(x) = jh$, for positive constant h, and that we want to approximate the second derivative at the interior nodes $x_j = jh$, $j = 1, \ldots, N$. By adding up the Taylor series for $f(x+h)$ and $f(x-h)$ centered at x, one can show that

$$\frac{f(x-h) - 2f(x) + f(x+h)}{h^2} = f''(x) + \mathcal{O}\left(h^2\right).$$

(More specifically, the order coefficient in the big-oh term is $\frac{1}{4!}\left\|f^{(4)}\right\|_\infty$, where the maximum is taken over the interval in question.) This translates into the formula

$$\frac{f_{j-1} - 2f_j + f_{j+1}}{h^2} = f''(x_j) + \mathcal{O}\left(h^2\right).$$

If we drop the $\mathcal{O}(h^2)$ term, we obtain the approximation

$$\frac{f_{j-1} - 2f_j + f_{j+1}}{h^2} \approx f''(x_j),\ j = 1, \ldots, N,$$

or, for our purposes, take the negative and obtain

$$\frac{-f_{j-1} + 2f_j - f_{j+1}}{h^2} = b_j \approx -f''(x_j).$$

Multiply both sides by h^2, move boundary terms to the right-hand side, and we see that the whole system can be written out nicely in matrix form as

$$T_N \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-1} \\ f_N \end{bmatrix} \equiv \begin{bmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-1} \\ f_N \end{bmatrix} = \begin{bmatrix} h^2 b_1 + f_0 \\ h^2 b_2 \\ \vdots \\ h^2 b_{N-1} \\ h^2 b_N + f_{N+1} \end{bmatrix}.$$

The matrix T_N is key here. Notice, it doesn't say anything about x or y. Back to our model problem: let's define

- V as the $N \times N$ sub-matrix of actual unknowns, so that $V = U\,(2 : N+1, 2 : N+1)$ in Matlab-ese;
- $F = [f_{ij}]$, with $i, j = 1, \ldots N$ and $f_{ij} = f\,(x_i, y_j)$;
- $G =[g_{ij}]$, with $g_{ij} = 0$ unless $i = 1$, in which case $g\,(x_0, y_j)$ is added in, or $j = 1$, in which case $g\,(x_i, y_0)$ is added on, or $i = N$, in which case $g\,(x_{N+1}, y_j)$ is added in, or $j = N$, in which case $g\,(x_i, y_{N+1})$ is added in.

Now observe that the (i, j)th entry of $VT_N + T_N V$ is exactly the left-hand side of (3), so that the resulting system can be written in the form

$$VT_N + T_N V = h^2 F + G \equiv B. \tag{4}$$

Finally, if we want to put the system in the traditional $Ax = b$ form, we have to do a little more work using tensor products. Here's a whirlwind tour of what we need.

About Tensors...

We are going to develop a powerful "bookkeeping" method that will rearrange the variables of Sylvester's equation automatically. The first basic idea needed here is that of the tensor product of two matrices, which is defined as follows:

Definition 29.1. Let $A = [a_{ij}]$ be an $m \times p$ matrix and $B = [b_{ij}]$ an $n \times q$ matrix. Then the *tensor product* of A and B is the $mn \times pq$ matrix $A \otimes B$ that can be expressed in block form as

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1j}B & \cdots & a_{1p}B \\ a_{21}B & a_{22}B & \cdots & a_{2j}B & \cdots & a_{2p}B \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{i1}B & a_{i2}B & \cdots & a_{ij}B & \cdots & a_{ip}B \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mj}B & \cdots & a_{mp}B \end{bmatrix}.$$

Example. Let $A = \begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 4 \\ -1 \end{bmatrix}$. Exhibit $A \otimes B$, $B \otimes A$, and $I_2 \otimes A$ and conclude that $A \otimes B \neq B \otimes A$.

Solution. From the definition,

$$A \otimes B = \begin{bmatrix} 1B & 3B \\ 2B & 1B \end{bmatrix} = \begin{bmatrix} 4 & 12 \\ -1 & -3 \\ 8 & 4 \\ -2 & -1 \end{bmatrix}, \quad B \otimes A = \begin{bmatrix} 4A \\ -1A \end{bmatrix} = \begin{bmatrix} 4 & 12 \\ -8 & -2 \\ -1 & -3 \\ -2 & -1 \end{bmatrix},$$

$$\text{and} \quad I_2 \otimes A = \begin{bmatrix} 1A & 0A \\ 0A & 1A \end{bmatrix} = \begin{bmatrix} 1 & 3 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 2 & 1 \end{bmatrix}.$$

The following item is an operator that turns matrices into vectors.

Definition 29.2. Let A be an $m \times n$ matrix. Then the $mn \times 1$ vector $\operatorname{vec} A$ is obtained from A by stacking the n columns of A vertically, with the first column at the top and the last column of A at the bottom.

Example. Let $A = \begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix}$. Compute $\operatorname{vec} A$.

Solution. There are two columns to stack, yielding $\operatorname{vec} A = [1, 2, 3, 1]^T$. □

The vec operator is linear ($\operatorname{vec}(aA + bB) = a \operatorname{vec} A + b \operatorname{vec} B$). We leave the proof, along proofs of the following simple tensor facts, to the reader.

Theorem 29.3. *Let A, B, C, D be suitably sized matrices. Then*

(1): $(A + B) \otimes C = A \otimes C + B \otimes C$
(2): $A \otimes (B + C) = A \otimes B + A \otimes C$
(3): $(A \otimes B) \otimes C = A \otimes (B \otimes C)$
(4): $(A \otimes B)^T = A^T \otimes B^T$
(5): $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$
(6): $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$

Here the key bookkeeping between tensor products and the vec operator.

Theorem. *(Bookkeeping Theorem) If A, X, B are matrices conformable for multiplication, then*

$$\operatorname{vec}(AXB) = \left(B^T \otimes A\right) \operatorname{vec} X.$$

Corollary. *The following linear systems in the unknown X are equivalent.*

(1): $A_1 X B_1 + A_2 X B_2 = C$
(2): $\left(\left(B_1^T \otimes A_1\right) + \left(B_2^T \otimes A_2\right)\right) \operatorname{vec} X = \operatorname{vec} C$

Now all we have to do is apply the bookkeeping theorem to (4) and we obtain the system

$$((T_N \otimes I_N) + (I_N \otimes T_N)) \operatorname{vec} V = \operatorname{vec} B \tag{5}$$

(Since both I_N and T_N are symmetric, transposes do not appear in this equation.)

Normally, we prefer to avoid explicitly forming the full coefficient matrix but it is very useful for theoretical purposes. In particular, here is a quick overview of the analysis of the mathematical discretization error. We have that the computed solution satisfies

$$\left(\left(T_N^T \otimes I_N\right) + \left(I_N \otimes T_N\right)\right) \operatorname{vec} V = \operatorname{vec} B,$$

while the matrix of exact solution values $W = [u(x_i, y_j)]$, $i, j = 1, \ldots, N$ satisfies

$$\left(\left(T_N^T \otimes I_N\right) + \left(I_N \otimes T_N\right)\right) \operatorname{vec} W = \operatorname{vec} B + \mathcal{O}\left(h^4\right).$$

The reason for h^4 is that we multiplied both sides of the original equation, which involved $\mathcal{O}(h^2)$, by h^2. Let $A = \left(\left(T_N^T \otimes I_N\right) + \left(I_N \otimes T_N\right)\right)$ and $e = \operatorname{vec} W - \operatorname{vec} V$ (the error vector), and we subtract the above equations to obtain

$$Ae = \mathcal{O}\left(h^4\right),$$

from which it follows that

$$\|e\| \leq \left\|A^{-1}\right\| \mathcal{O}\left(h^4\right).$$

So we can see that the error estimate depends on an eigenanalysis of A. As a matter of fact, A is symmetric positive definite with smallest eigenvalue equal to twice the smallest eigenvalue of T_N, which happens to be

$$2\left(1 - \cos\frac{\pi}{N+1}\right) = \left(\frac{\pi}{N+1}\right)^2 + \mathcal{O}\left(\left(\frac{1}{N+1}\right)^4\right).$$

Now $h = 1/(N+1)$, so one deduces that the error vector is $\mathcal{O}(h^2)$.

One final application of this explicit matrix/vector form of the model problem is that we can use Matlab to rather easily solve the system. Here's the code:

```
% script:  modelproblem.m
% description:  construct coefficient matrix and
% solution to model problem
N = 32
m = N^2
T = Tridiag(N,-1,2,-1);
h = 1/(N+1)
xnodes = linspace(0,1,N+2);
[X,Y] = meshgrid(xnodes,xnodes);
F = 13*pi^2*sin(2*pi*X).*sin(3*pi*Y);
G = X.*Y;
G(2:N+1,2:N+1) = 0;
G(2:N+1,2)=G(2:N+1,1);
G(2:N+1,N+1) = G(2:N+1,N+2);
G(2, 2:N+1) = G(2,2:N+1) + G(1,2:N+1);
G(N+1,2:N+1) = G(N+1,2:N+1) + G(N+2,2:N+1);
B = h^2*F+G;
B = B(2:N+1,2:N+1);
A = kron(T,eye(N)) + kron(eye(N),T);
b = reshape(B,N^2,1); % vec(B)
```

```
x = A\b; % solve the system
V = reshape(x,N,N);
W = sin(2*pi*X).*sin(3*pi*Y)+X.*Y;
W = W(2:N+1,2:N+1);
err = norm(reshape(V-W,N^2,1),inf)/norm(reshape(W,N^2,1),inf)
xint = xnodes(2:N+1);
[Xint,Yint] = meshgrid(xint,xint);
mesh(Xint,Yint,V-W);
```

N1. (Non-text exercise) Write up a script that applies Jacobi and Gauss-Seidel to the model problem where

$$\begin{aligned} g(x,y) &= xy \\ f(x,y) &= 13\pi^2 \sin(2\pi x)\sin(3\pi y). \end{aligned}$$

The exact solution is really $u(x,y) = \sin(2\pi x)\sin(3\pi y) + xy$. The numerical solution is generated by the above script. Iterate your method until the relative error between your solution and the numerical solution is at most $1e{-}3$. Use $N = 32$.

An Arnoldi Experiment with Eigenvalues

```
lec28
A = D;
size(A)
A(1,1)
evals = eig(A);
plot(evals,'o')
n = 5
b = randn(n,1);
[Q,H] = arnoldi(A,b,n);
evaln = eig(H(1:n,:))
hold on
plot(evaln,'x')
% now repeat this with n = 8, 10 and compare eigenvalues
```

35. GMRES

(11/28/06) We largely follow text development. At some point, we do the following experiment, which requires that the Matlab tool gmresid.m be downloaded. Just for the record: Matlab does have a gmres routine. It's much fancier than the one we'll use, which is a naive implementation of the algorithm detailed in class.

```
m = 200
n = 10
A = 2*eye(m) + 0.5*randn(m)/sqrt(m);
cond(A) % A well-behaved?
plot(eig(A))
axis([1, 3, -1, 1])
b = ones(m,1);
X = gmresid(A,b,n);
R = A*X - ones(m,n)
x = A\b;
r = zeros(n,1);
for ii = 1:n, r(ii) = norm(R(:, ii)); end
semilogy(r)
norm(X(:,n) - x, inf)
```

38. Conjugate Gradients

(11/30/06) We will outline the text development – too much here to do in detail in one period – and will make this comparison between conjugate gradients and GMRES. We have to load up the files cg.m and lec38.m from MatlabTools. Here is the content of lec38.m

```
% script:  lec38.m
```

```
% description:  performs some experiments with CG and GMRES.
m = 200
n = 40
randn('state',1)
A = triu(2*eye(m) + 0.5*randn(m)/sqrt(m));
A = A'*A; % create a SPD matrix
condA = cond(A) % A well-behaved?
b = ones(m,1);
x = A\b;
X = gmresid(A,b,n);
Y = cg(A,b,n);
R = A*X - ones(m,n);
S = A*Y - ones(m,n);
r = zeros(n,1);
s = r;
for ii = 1:n, r(ii) = norm(R(:, ii)); s(ii) = norm(S(:, ii)); end
figure;
plot(r);
hold on
plot(s);
grid
normx = norm(x);
norm(X(:,n) - x)/normx
norm(Y(:,n) - x)/normx
E = (X - repmat(x,1,n))/normx;
F = (Y - repmat(x,1,n))/normx;
for ii = 1:n, r(ii) = norm(E(:, ii)); s(ii) = norm(F(:, ii)); end
figure;
plot(r);
hold on
plot(s);
```

Run this program and examine the results. Then edit the line defining the matrix A by doubling the coefficient of the random part and repeat the experiment. Repeat until the coefficient is 8.

40. Preconditioning

(12/05/06) There are many big ideas here – domain decomposition, multigrid methods, etc. We're going to leave the extensive discussion to the reader and illustrate the idea with a simple example of preconditioning.

In the spirit of the lecture notes for today, we will actually do some simple preconditioning on a SPD matrix similar to the text. The mathematical formulation of such precondtioners we have discussed: $M = C^*C$ and

$$\widetilde{A}\widetilde{x} = \left(C^{-*}AC^{-1}\right)(Cx) = C^{-*}b = \widetilde{b}.$$

Of course, we want $M \approx A^{-1}$ in some sense. We will not worry about implementation niceties here. Edit and run the script lec40.m. We will use the simplest – Jacobi – preconditioner.

Example. $M = \mathrm{diag}\,(A)$, $C = M^{1/2}$.

Niceties Notes: Actually, there is a remarkable advantage to using SPD conditioners: any such matrix M has a square root, that is, a SPD matrix C such that $M = C^2$. This is an improvement over the form given above, because it allows us to write the system as

$$\widetilde{A}\widetilde{x} = \left(C^{-1}AC^{-1}\right)(Cx) = C^{-1}b = \widetilde{b}.$$

If we stick to real matrices, then a careful analysis of the CG method applied to this alternate system leads to a very simple formulation of preconditioned CG. See the file pcg.m for details and let's edit and run the script lec40alt.m.

One last comment about preconditioners and classical iterative methods: Consider solving $Ax = b$ by way of an iteration scheme derived from a regular splitting $A = M - N$, so that

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b = Gx^{(k)} + d.$$

We want $\rho\,(G)$ small because this gives the asymptotic convergence rate coefficient. Since $\rho\,(G) \leq \|G\|$, we would like the latter term small. However

$$I - M^{-1}A = I - M^{-1}\,(M - N) = G,$$

so we want $\|I - M^{-1}A\|$ small. From this, we see that M^{-1} would be a good preconditioner if we make the $\|G\|$ small, and we are in the curious situation where the interests of (classical) iteration fans and (contemporary) Arnoldi iteration fans dovetail.

Finally, then, our authors assert:

"**...we find ourselves at the philosophical center of the scientific computing of the future....**"

41. Review

(12/07/06) We will review and do class evaluations today. The take-home final will be placed on the web (our Class Resources) on Friday, December