Handouts for Introduction/
Linear Algebra Review/Basic
Decompositions/Intro. to MATLAB
Numerical Analysis and Scientific Computing  
(Numerical Linear Algebra)  
Math / CS 143 M  
TTh 5:30 – 6:45 Fall 2009

Leslie Foster, Phone 408-924-5123, foster@mathcs.sjsu.edu, www.math.sjsu.edu/~foster

Scientific computing studies the design, analysis and use of computer algorithms for solving problems in science and engineering. Math / Computer Science 143M focuses on the most widely used algorithms in scientific computing – algorithms relating to linear systems of equations. The course is a nice combination of mathematics, computer science and applications. We discuss triangular, orthogonal, eigenvalue and singular value decompositions and use Gaussian elimination, QR, iterative and other algorithms.

Prerequisites Linear algebra (Math 129A) and a programming course such as CS46A or CS43.

Mathematics in the Course The goal of the course is to understand the use and the relative advantages of the algorithms discussed, not on mathematical theory. Some derivations and proofs that further this goal will be presented.

Computing in the Course We will use Matlab which is the ideal computer language for numerical linear algebra. With Matlab one can often do sophisticated examples with very little coding. The programming demands are substantially less than in a programming course. Students find it easy to learn Matlab as needed in the course. In addition, I have a license that permits lending a professional version of Matlab for home use by students during the semester.

There will be an project assigned where students take state of the art algorithms from internet (www.netlib.org/templates) and use Matlab to try these algorithms on real world matrices from the matrix market (http://math.nist.gov/MatrixMarket). Students can get a feel for the efficiency, accuracy and reliability of these algorithms on real world problems quickly, without coding. Students do well on the project due to its experimental nature.

Applications Image compression, heat flow, vibrating structures, search engines, curve fitting and regression analysis are some applications in the course. Background for these is presented as needed.

Who might take the course This is a good elective course for students in a variety of fields including mathematics, computer science, physics, meteorology, operations research, and engineering. The course discusses important computational tools in these areas.

P.S. My research area is numerical linear algebra. I will occasionally discuss interesting and fun topics that can be easily understood by students in the course and are related to my research.
"The book of the universe is written in the language of mathematics; without mathematics it is not possible to understand a word of it." Galileo Galilei

Course: Math / CS 143M  
Course Title: Numerical Analysis and Scientific Computing (Matrices)

Instructor: Leslie Foster  
Semester: Fall 2009  
Office: MH 312  
Time: TTh 5:30-6:45 in MH 235  
Phone: 924-5123 (my office)  
Web page: www.math.sjsu.edu/~foster  
Phones: 924-5100 (Dept.)  
Email: foster@math.sjsu.edu  
Office Hours (tentative): TTh 9:45-11:00, 3:00-4:15 or by appointment or (for questions related to the course) any time I'm not busy.  
Furlough Days (class cancelled): Tuesday, Sept. 22 and Thursday, Oct. 29.


Material Covered: Selected material from Chapters 1-5 and 7.

Learning Objectives: To understand common matrix decompositions, algorithms to construct them and applications of these decompositions; to be able to compare algorithms in terms of efficiency, accuracy and reliability; to be able to apply programming skills to mathematical problems; to interpret numerical results and to understand the limits of numerical accuracy; to be able to write a technical report using good English, mathematics and computer science.

Languages used: When I hand out subroutines or discuss code in a specific language I will usually use Matlab. I will introduce you to Matlab in class. You may write code in any decent high level language (not Basic). For most assignments it will be a much easier to use Matlab than a traditional programming language. Matlab is available on the departmental computers and probably elsewhere in the university. Also I can lend, for free but for the semester only, copies of Matlab 5.3 for Windows. Finally, a student version of Matlab (PC or Mac) is available. Contact Mathworks (www.mathworks.com) for more information.

Computer Access (in MH 221): Please add Math 110L or pay in the department office. Adding Math 110L costs nothing to you if you are a full-time, resident student. During the project we will hold some classes in MH 221. The project will last approximately 3 weeks and it will be useful to have computer access in MH 221.

Requirements and Points  

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Points Each</th>
<th>Basis of Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 midterm</td>
<td>100</td>
<td>Curve within reason. The curve for</td>
</tr>
<tr>
<td>Final</td>
<td>150</td>
<td>each individual test/quiz/etc. will</td>
</tr>
<tr>
<td>3-6 Quizzes</td>
<td>20</td>
<td>be announced in class when the</td>
</tr>
<tr>
<td>~5 computing assignments</td>
<td>10-20</td>
<td>assignment is returned. If not</td>
</tr>
<tr>
<td>1 report / project</td>
<td>100</td>
<td>announced the curve is 90/80/70/60 for A/B/C/D (+/-: top/bottom 3% of range)</td>
</tr>
<tr>
<td>Total</td>
<td>550</td>
<td>approx.</td>
</tr>
</tbody>
</table>

You will have at least a week’s notice for all assignment due dates/ quizzes / tests. Computing assignments are required in order to pass the course.

Homework: Homework will be collected regularly and at the same time solutions will be provided (no late homework will be accepted). Each student will be required to take at least one turn grading a set of homework (or share grading if there aren’t enough assignments). You automatically get a perfect score on any homework set you grade. If your course grade is in doubt homework and other signs of work will count by helping your grade up to a maximum of 2-3% points in the final grade.

Make-up exams: None except under very unusual circumstances and then only if you contact me (via phone, message, in person, note, ...) before the exam. Make-up exams will be more difficult.

Cheating: Cheating on any quiz, exam, or program may result in an F in the course. On programs you can consult with other students on general matters. A copied program is cheating. Also turning in output that is not produced by your program is cheating.

Additional information / requirements please see http://www.sjsu.edu/math/courses/greensheet.
Your first quiz will be identical to the quiz below, except the numbers will be different. I have listed page numbers in David Lay’s *Linear Algebra and Its Applications, 2nd edition* that contain very similar examples. I believe that any linear algebra text will have similar examples.

1. For \( A = \begin{pmatrix} 1 & 2 & -1 \\ 0 & 1 & 2 \\ 1 & 4 & 4 \end{pmatrix} \) and \( b = \begin{pmatrix} -1 \\ 4 \\ 9 \end{pmatrix} \) solve for \( x \) in \( Ax = b \) using Gaussian elimination applied to the augmented matrix \((A,b)\). Show your work. (Lay pages 5-6 – Lay’s procedure is a small variation of Gaussian elimination, but it is close enough for the quiz).

2. For \( u = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \), \( v = \begin{pmatrix} 2 \\ 3 \\ 2 \\ 4 \end{pmatrix} \) and \( w = \begin{pmatrix} 3 \\ 7 \\ 3 \\ 1 \end{pmatrix} \) find an orthonormal basis for \( \text{span}(u,v,w) \). Show your work. (Lay pages 397-400.)

3. Find the eigenvalues and the corresponding eigenvectors of \( A = \begin{pmatrix} -1 & -6 \\ 2 & 6 \end{pmatrix} \). Show your work. (Lay pages 305 and 298).
1. Compute

(a) the angle between the vectors \( \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \) and \( \begin{pmatrix} 5 \\ 2 \\ -3 \end{pmatrix} \).

(b) \( B = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} A \) where \( A = \begin{pmatrix} 6 & 2 & -1 \\ 1 & 4 & 6 \\ 3 & -5 & 4 \end{pmatrix} \). How are \( A \) and \( B \) related?

(c) \( \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \) \( (3 \ 2 \ 1) \)

(d) \( \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \)

2. Show that \( A = \begin{pmatrix} c & 0 & 0 & \cdots & 0 \\ 0 & c & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & c \end{pmatrix} \).

3. A strictly upper triangular matrix has zeroes on and below the main diagonal. Let \( A \) be a strictly upper triangular matrix of order \( n \). Show that \( A^2 = 0 \).

4. Carry out \( C = AB \) via “flipping rows”, “flipping columns” and outer products. Show enough intermediate steps to illustrate the method used in each case.

\[
C = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 2 & 1 \\ 1 & 0 & 1 & 3 & 2 \\ 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{pmatrix}
\]

5. Briefly describe the idea behind block matrix multiplication. Why can using blocks make matrix multiplication faster on most modern computers.

6. Argue that a symmetric triangular matrix is diagonal.

7. An upper Hessenberg matrix has zeros below the first subdiagonal. Argue that a symmetric Hessenberg matrix is tridiagonal.

8. Do exercise 1.2.4 (page 13) from Watkins.


10. Do exercise 2.1.13 (page 114) from Watkins.

11. Find the one and infinity norm for \( A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 1 & 6 \end{pmatrix} \). Use Matlab to find the two norm and singular values of \( A \).
1. Consider the Taylor's series expansions

\[
    u(x+h) = u(x) + h u'(x) + h^2 u''(x) / 2 + h^3 u'''(x) / 6 + h^4 u''''(\xi_1) / 24
\]

\[
    u(x-h) = u(x) - h u'(x) + h^2 u''(x) / 2 - h^3 u'''(x) / 6 + h^4 u''''(\xi_2) / 24
\]

By adding these equations derive the approximation

\[
    u'(x) \equiv \frac{u(x-h) - 2u(x) + u(x+h)}{h^2}
\]

and show that the error term in the approximation is \(- (u''''(\xi_1) + u''''(\xi_2)) h^2/24\).

2. Consider the differential equation \(u''(x) - u(x) = x^2\) with \(u(1) = u(2) = 0\). Assume that you divide the interval from 1 to 2 into \(n\) equally spaced subintervals.
   (a) Write down a system of three equations in there unknowns in the form \(Au = b\) that can be used to find a (rough) approximate solution to this differential equation. (Hint -- let \(n\) be four.)
   (b) Write down a system of \(n-1\) equations in \(n-1\) unknowns in the form \(Au = b\) that approximates the solution to the differential equations. You do not need to solve these equations (although it is not hard to do using Matlab).

3 and 4. LU decomposition handout #1, 2

5. Eigenvalue decomposition handout #1 abc

6. QR decomposition handout #1 abc

7. SVD decomposition handout #1 abc

8. Watkins page 98 # 1.8.7 (related to \(PA = LU\))

9. Watkins page 224 #3.4.10 (related to \(A = QR\))

10. Watkins page 263 # 4.1.6 (related to singular values)

11. Watkins page 305 #5.2.2 (related to eigenvalues)

12. Prove that if \(A = V D V^{-1}\) then \(A^n = V D^n V^{-1}\).
Four Ways to do Matrix Multiplication

Example: for

$$A = \begin{pmatrix}
1 & 0 & 1 & 1 \\
0 & 2 & 1 & 2 \\
3 & 1 & 1 & 0 \\
2 & 0 & 1 & 3
\end{pmatrix}, \quad B = \begin{pmatrix}
1 & 2 & 0 & 1 \\
0 & 1 & 0 & 0 \\
0 & 0 & 3 & 0 \\
1 & 0 & 0 & 2
\end{pmatrix}$$

calculate

$$C = AB = \begin{pmatrix}
2 & 2 & 3 & 3 \\
2 & 2 & 3 & 4 \\
3 & 7 & 3 & 3 \\
5 & 4 & 3 & 8
\end{pmatrix}$$

(1) Column flipping:

$$\text{col}_1 = A \begin{pmatrix}
1 \\
0 \\
1 \\
1
\end{pmatrix}, \quad \text{col}_2 = A \begin{pmatrix}
2 \\
1 \\
0 \\
5
\end{pmatrix}, \quad \text{col}_3 = A \begin{pmatrix}
2 \\
1 \\
7 \\
4
\end{pmatrix}, \quad \text{col}_4 = A \begin{pmatrix}
1 \\
1 \\
1 \\
2
\end{pmatrix} = \begin{pmatrix}
1 \\
3 \\
3 \\
3
\end{pmatrix}$$

(2) Row flipping:

$$\text{row}_1 = \begin{pmatrix}
1 & 0 & 1 & 1
\end{pmatrix}, \quad B = \begin{pmatrix}
2 & 2 & 3 & 3
\end{pmatrix},$$

$$\text{row}_2 = \begin{pmatrix}
0 & 2 & 1 & 2
\end{pmatrix}, \quad B = \begin{pmatrix}
2 & 2 & 3 & 4
\end{pmatrix},$$

$$\text{row}_3 = \begin{pmatrix}
3 & 1 & 1 & 0
\end{pmatrix}, \quad B = \begin{pmatrix}
3 & 7 & 3 & 3
\end{pmatrix},$$

$$\text{row}_4 = \begin{pmatrix}
2 & 0 & 1 & 3
\end{pmatrix}, \quad B = \begin{pmatrix}
5 & 4 & 3 & 8
\end{pmatrix}$$

(3) outer product:

$$C = \begin{pmatrix}
1 \\
0 \\
3 \\
2
\end{pmatrix} \begin{pmatrix}
1 & 2 & 0 & 1 \\
0 & 1 & 0 & 0 \\
1 & 1 \\
0 & 3 & 0 & 0
\end{pmatrix} + \begin{pmatrix}
2 \\
0 \\
1 \\
0
\end{pmatrix} \begin{pmatrix}
0 & 1 & 0 & 0 \\
0 & 0 & 3 & 0 \\
1 & 1 \\
0 & 0 & 3 & 0
\end{pmatrix} + \begin{pmatrix}
1 \\
1 \\
1 \\
0
\end{pmatrix} \begin{pmatrix}
0 & 0 & 3 & 0 \\
0 & 0 & 0 & 0 \\
2 & 0 & 0 & 4 \\
0 & 0 & 0 & 0
\end{pmatrix} + \begin{pmatrix}
1 \\
0 \\
0 \\
3
\end{pmatrix} \begin{pmatrix}
1 & 0 & 0 & 2 \\
0 & 0 & 0 & 0 \\
2 & 2 & 3 & 4 \\
3 & 7 & 3 & 3
\end{pmatrix} = \begin{pmatrix}
1 & 2 & 3 & 3 \\
2 & 2 & 3 & 4 \\
3 & 7 & 3 & 3 \\
5 & 4 & 3 & 8
\end{pmatrix}$$

(4) Block Multiplication:

$$A = \begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix}, \quad B = \begin{pmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{pmatrix}$$

$$C = \begin{pmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{pmatrix} = \begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix} \begin{pmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{pmatrix} = \begin{pmatrix}
A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\
A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22}
\end{pmatrix}$$

$$\begin{pmatrix}
(1 & 0 & 1 & 2) + (1 & 1 & 0 & 0) + (1 & 0 & 0 & 1) + (1 & 1 & 3 & 0) &=& \begin{pmatrix}
2 & 2 & 3 & 3 \\
2 & 2 & 3 & 4 \\
3 & 7 & 3 & 3 \\
5 & 4 & 3 & 8
\end{pmatrix}
\end{pmatrix}$$

Note: On a computer block multiplication can make effective use of the fast (cache) memory and can speed up a computation substantially.
The Best of the 20th Century: Editors Name Top 10 Algorithms

By Barry A. Cipra

**Algos** is the Greek word for pain. **Algor** is Latin, to be cold. Neither is the root for algorithm, which stems instead from al-Khwarizmi, the name of the ninth-century Arab scholar whose book *al-jabr wa'l muqabalah* devolved into today’s high school algebra textbooks. Al-Khwarizmi stressed the importance of methodical procedures for solving problems. Were he around today, he’d no doubt be impressed by the advances in his eponymous approach.

Some of the very best algorithms of the computer age are highlighted in the January/February 2000 issue of *Computing in Science & Engineering*, a joint publication of the American Institute of Physics and the IEEE Computer Society. Guest editors Jack Dongarra of the University of Tennessee and Oak Ridge National Laboratory and Francis Sullivan of the Center for Computing Sciences at the Institute for Defense Analyses put together a list they call the “Top Ten Algorithms of the Century.”

“We tried to assemble the 10 algorithms with the greatest influence on the development and practice of science and engineering in the 20th century,” Dongarra and Sullivan write. As with any top-10 list, their selections—and non-selections—are bound to be controversial, they acknowledge. When it comes to picking the algorithmic best, there seems to be no best algorithm.

Without further ado, here’s the CISE top-10 list, in chronological order. (Dates and names associated with the algorithms should be read as first-order approximations. Most algorithms take shape over time, with many contributors.)

1946: John von Neumann, Stan Ulam, and Nick Metropolis, all at the Los Alamos Scientific Laboratory, cook up the Metropolis algorithm, also known as the **Monte Carlo method**.

The Metropolis algorithm aims to obtain approximate solutions to numerical problems with unmanageably many degrees of freedom and to combinatorial problems of factorial size, by mimicking a random process. Given the digital computer’s reputation for deterministic calculation, it’s fitting that one of its earliest applications was the generation of random numbers.

1947: George Dantzig, at the RAND Corporation, creates the **simplex method for linear programming**.

In terms of widespread application, Dantzig’s algorithm is one of the most successful of all time: Linear programming dominates the world of industry, where economic survival depends on the ability to optimize within budgetary and other constraints. (Of course, the “real” problems of industry are often nonlinear; the use of linear programming is sometimes dictated by the computational budget.) The simplex method is an elegant way of arriving at optimal answers. Although theoretically susceptible to exponential delays, the algorithm in practice is highly efficient—which in itself says something interesting about the nature of computation.

1950: Magnus Hestenes, Eduard Stiefel, and Cornelius Lanczos, all from the Institute for Numerical Analysis at the National Bureau of Standards, initiate the development of **Krylov subspace iteration methods**.

These algorithms address the seemingly simple task of solving equations of the form $Ax = b$. The catch, of course, is that $A$ is a huge $n \times n$ matrix, so that the algebraic answer $x = b/A$ is not so easy to compute. (Indeed, matrix “division” is not a particularly useful concept.) Iterative methods—such as solving equations of the form $Kx + 1 = Kx + b - Ax$, with a simpler matrix $K$ that’s ideally “close” to $A$—lead to the study of Krylov subspaces. Named for the Russian mathematician Nikolai Krylov, Krylov subspaces are spanned by powers of a matrix applied to an initial “remainder” vector $r_0 = b - Ax$. Lanczos found a nifty way to generate an orthogonal basis for such a subspace when the matrix is symmetric. Hestenes and Stiefel proposed an even niftier method, known as the conjugate gradient method, for systems that are both symmetric and positive definite. Over the last 50 years, numerous researchers have improved and extended these algorithms. The current suite includes techniques for non-symmetric systems, with acronyms like GMRES and Bi-CGSTAB. (GMRES and Bi-CGSTAB premiered in *SIAM Journal on Scientific and Statistical Computing*, in 1986 and 1992, respectively.)

1951: Alston Householder of Oak Ridge National Laboratory formalizes the **decompositional approach to matrix computations**.

The ability to factor matrices into triangular, diagonal, orthogonal, and other special forms has turned out to be extremely useful. The decompositional approach has enabled software developers to produce flexible and efficient matrix packages. It also facilitates the analysis of rounding errors, one of the big bugbears of numerical linear algebra. (In 1961, James Wilkinson of the National Physical Laboratory in London published a seminal paper in the *Journal of the ACM*, titled “Error Analysis of Direct Methods of Matrix Inversion,” based on the LU decomposition of a matrix as a product of lower and upper triangular factors.)

1957: John Backus leads a team at IBM in developing the **Fortran optimizing compiler**.

The creation of Fortran may rank as the single most important event in the history of computer programming: Finally, scientists
(and others) could tell the computer what they wanted it to do, without having to descend into the netherworld of machine code. Although modest by modern compiler standards—Fortran I consisted of a mere 23,500 assembly-language instructions—the early compiler was nonetheless capable of surprisingly sophisticated computations. As Backus himself recalls in a recent history of Fortran I, II, and III, published in 1998 in the IEEE Annals of the History of Computing, the compiler “produced code of such efficiency that its output would startle the programmers who studied it.”

1959–61: J.G.F. Francis of Ferranti Ltd., London, finds a stable method for computing eigenvalues, known as the QR algorithm. Eigenvalues are arguably the most important numbers associated with matrices—and they can be the trickiest to compute. It’s relatively easy to transform a square matrix into a matrix that’s “almost” upper triangular, meaning one with a single extra set of nonzero entries just below the main diagonal. But chipping away those final nonzeros, without launching an avalanche of error, is nontrivial. The QR algorithm is just the ticket. Based on the QR decomposition, which writes \( A = QR \) into \( A_{i+1} = RQ_i \), with a few bells and whistles for accelerating convergence to upper triangular form. By the mid-1960s, the QR algorithm had turned once-formidable eigenvalue problems into routine calculations.

1962: Tony Hoare of Elliott Brothers, Ltd., London, presents Quicksort. Putting \( N \) things in numerical or alphabetical order is mind-numbingly mundane. The intellectual challenge lies in devising ways of doing so quickly. Hoare’s scheme uses the age-old recursive strategy of divide and conquer to solve the problem: Pick one element as a “pivot,” separate the rest into piles of “big” and “small” elements (as compared with the pivot), and then repeat this procedure on each pile. Although it’s possible to get stuck doing all \( N(N - 1)/2 \) comparisons (especially if you use as your pivot the first item on a list that’s already sorted!), Quicksort runs on average with \( O(N \log N) \) efficiency. Its elegant simplicity has made Quicksort the posterchild of computational complexity.

1965: James Cooley of the IBM T.J. Watson Research Center and John Tukey of Princeton University and AT&T Bell Laboratories unveil the fast Fourier transform. Easily the most far-reaching algorhythm in applied mathematics, the FFT revolutionized signal processing. The underlying idea goes back to Gauss (who needed to calculate orbits of asteroids), but it was the Cooley–Tukey paper that made it clear how easily Fourier transforms can be computed. Like Quicksort, the FFT relies on a divide-and-conquer strategy to reduce an ostensibly \( O(N^2) \) chore to an \( O(N \log N) \) frolic. But unlike Quick-sort, the implementation is (at first sight) nonintuitive and less than straightforward. This in itself gave computer science an impetus to investigate the inherent complexity of computational problems and algorithms.

1977: Helaman Ferguson and Rodney Forcade of Brigham Young University advance an integer relation detection algorithm. The problem is an old one: Given a bunch of real numbers, say \( x_1, x_2, \ldots, x_n \), are there integers \( a_1, a_2, \ldots, a_n \) (not all 0) for which
\[
a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = 0?
\]

For \( n = 2 \), the venerable Euclidean algorithm does the job, computing terms in the continued-fraction expansion of \( x_1/x_2 \). If \( x_1/x_2 \) is rational, the expansion terminates and, with proper unraveling, gives the “smallest” integers \( a_1 \) and \( a_2 \). If the Euclidean algorithm doesn’t terminate—or if you simply get tired of computing it—then the unraveling procedure at least provides lower bounds on the size of the smallest integer relation. Ferguson and Forcade’s generalization, although much more difficult to implement (and to understand), is also more powerful. Their detection algorithm, for example, has been used to find the precise coefficients of the polynomials satisfied by the third and fourth bifurcation points, \( B_3 = 3.544090 \) and \( B_4 = 3.564407 \), of the logistic map. (The latter polynomial is of degree 120; its largest coefficient is 25710.) It has also proved useful in simplifying calculations with Feynman diagrams in quantum field theory.

1987: Leslie Greengard and Vladimir Rokhlin of Yale University invent the fast multipole algorithm. This algorithm overcomes one of the biggest challenges in \( N \)-body simulations: the fact that accurate calculations of the motions of \( N \) particles interacting via gravitational or electrostatic forces (think stars in a galaxy, or atoms in a protein) would seem to require \( O(N^2) \) computations—one for each pair of particles. The fast multipole algorithm gets by with \( O(N) \) computations. It does so by using multipole expansions (net charge or mass, dipole moment, quadrupole, and so forth) to approximate the effects of a distant group of particles on a local group. A hierarchical decomposition of space is used to define ever-larger groups as distances increase. One of the distinct advantages of the fast multipole algorithm is that it comes equipped with rigorous error estimates, a feature that many methods lack.

What new insights and algorithms will the 21st century bring? The complete answer obviously won’t be known for another hundred years. One thing seems certain, however. As Sullivan writes in the introduction to the top-10 list, “The new century is not going to be very restful for us, but it is not going to be dull either!”

"Barry A. Cipra is a mathematician and writer based in Northfield, Minnesota."
Basic Decompositions: LU Decomposition $A = LU$, $L$ lower triangular, $U$ upper triangular

Application: Suppose that we want to solve $Ax = b$ and we know $A = LU$. Then $LUx = b$ and we can solve for $x$ in two steps:
1) solve $Lz = b$ and then
2) solve $Ux = z$

Remark: You can construct $L$ and $U$ by doing Gaussian elimination, storing the multiplier in $L$

Example: Solve $Ax = b$ where $A = \begin{pmatrix} 3 & 2 & 1 \\ 0 & 4 & 6 \\ 6 & 8 & 3 \end{pmatrix}$, $b = \begin{pmatrix} 7 \\ 6 \\ 15 \end{pmatrix}$.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & 2 & 1 \\ 0 & 4 & 6 \\ 6 & 8 & 3 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & 2 & 1 \\ 0 & 4 & 6 \\ 6 & 8 & 3 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 1 & 1 \end{pmatrix} \begin{pmatrix} 3 & 2 & 1 \\ 0 & 4 & 6 \\ 6 & 8 & 3 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 1 & 1 \end{pmatrix} \begin{pmatrix} 7 \\ 6 \\ 15 \end{pmatrix} \Rightarrow \begin{pmatrix} 7 \\ 6 \\ -5 \end{pmatrix} \Rightarrow \begin{pmatrix} 3 & 2 & 1 \\ 0 & 4 & 6 \\ 0 & 0 & -5 \end{pmatrix}

So $L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 1 & 1 \end{pmatrix}$, $U = \begin{pmatrix} 3 & 2 & 1 \\ 0 & 4 & 6 \\ 0 & 0 & -5 \end{pmatrix}$

To solve $Ax = b$

1) Solve $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 1 & 1 \end{pmatrix} z = \begin{pmatrix} 7 \\ 6 \\ 15 \end{pmatrix} \Rightarrow z = \begin{pmatrix} 7 \\ 6 \\ -5 \end{pmatrix}$

1) Solve $\begin{pmatrix} 0 & 4 & 6 \\ 3 & 2 & 1 \\ 0 & 0 & -5 \end{pmatrix} x = \begin{pmatrix} 7 \\ 6 \\ -5 \end{pmatrix} \Rightarrow x = \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix}$

Remark: Later we will permute the rows of $A$ getting $PA = LU$ (pivoted LU decomposition) where $P$ is a permutation matrix.

HW: For $A = \begin{pmatrix} 2 & 3 & -1 \\ 0 & 2 & 1 \\ 4 & 8 & 2 \end{pmatrix}$, $b = \begin{pmatrix} 4 \\ 2 \\ 16 \end{pmatrix}$. (1) Find $L$ and $U$ so that $A = LU$. Check that $LU$ equals $A$. (2) Find $x$ so that $Ax = b$. Use $L$ and $U$.
Basic Decompositions: Eigenvalue Decomposition $A = VDV^{-1}$, $A, V$ and $D$ $n \times n$, $D$ diagonal, $V$ nonsingular

Application: Calculation of $A^n = A \cdot A \cdot A \ldots A$. If $A = VDV^{-1}$ then $A^n = VD^nV^{-1}$. The calculation of $VD^nV^{-1}$ is much easier than $A \cdot A \cdot A \ldots A$. The need to calculate $A^n$ shows up in solving difference equations. The eigenvalue decomposition is also important in calculating $e^{At}$ which shows up in solving differential equations. These solutions are important in many applications including earthquake vibration analysis (see the first computer assignment at www.math.sjsu.edu/~foster\cs143mf07.html).

Remark To find $V$ and $D$ we will assume that the $n \times n$ matrix $A$ has $n$ eigenvalues $\lambda_1, \lambda_2, \ldots \lambda_n$ and $n$ corresponding eigenvectors $v_1, v_2, \ldots v_n$. Then

$$D = \begin{pmatrix}
\lambda_1 & 0 & \ldots & 0 \\
0 & \lambda_2 & \ldots & 0 \\
& & \ddots & \vdots \\
0 & \ldots & 0 & \lambda_n
\end{pmatrix}$$

and $V = (v_1 \ v_2 \ \ldots \ \ v_n)$.

Example If $A = \begin{pmatrix} 0 & 3 \\ -2 & 5 \end{pmatrix}$ then $\det(A - \lambda I) = \lambda^2 - 5\lambda + 6$ so that $\lambda_1 = 2, \lambda_2 = 3$. For $\lambda_1 = 2$: $(A - 2I)v = 0 \Rightarrow v_1 = c \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$, selecting $c = 1$. For $\lambda_2 = 3$: $(A - 3I)v = 0 \Rightarrow v_2 = c \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, selecting $c = 1$.

So $D = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}$ and $V = \begin{pmatrix} 3 & 1 \\ 2 & 1 \end{pmatrix}$. Also $V^{-1} = \begin{pmatrix} 1 & -1 \\ -2 & 3 \end{pmatrix}$. Note that $VDV^{-1} = A$. Also $A^4 = VD^4V^{-1} = \begin{pmatrix} 3 & 1 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 16 & 0 \\ 0 & 81 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ -2 & 3 \end{pmatrix} = \begin{pmatrix} -114 & 195 \\ -130 & 211 \end{pmatrix}$.

Remark Matlab's $[V,D]=\text{eig}(A)$ will calculate $D$ and a $V$. Note that $V$ is not unique.

Homework The eigenvalues and eigenvectors of $A = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}$ are $\lambda_1 = 2, v_1 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$, $\lambda_2 = 4, v_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$. You don't need to show this. (a) What are $D, V$ and $V^{-1}$ (b) Verify that $A = VDV^{-1}$ (c) Find $A^5$ using $VD^5V^{-1}$
Decomposition 2 \( A = QR \) (QR decomposition). Orthogonalized R right \( \Delta \) column

Application: If \( A \) is rectangular (m by n with \( m \geq n \)), then to solve \( \min \| b - Ax \| \) via least squares:

1. Form \( A = QR \) with \( Q \) \( m \times m \) and \( R \) \( m \times n \) right \( \Delta \)

2. Form \( QR \)\( b \)

3. Solve \( RX = Q^T b \)

Remark: You can get \( Q \) \& \( R \) by Gram-Schmidt orthogonalization (and other techniques)

\[ \min \| b - Ax \| \text{ with } A = \begin{pmatrix} 0 & 4 & 3 & 1 \\ 0 & 3 & 1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} \frac{2}{3} \\ \frac{4}{3} \end{pmatrix} \]

Using Gram-Schmidt:

\[ V_1 = x_1, \quad V_2 = x_2 - \frac{x_2 \cdot V_1}{V_1 \cdot V_1} V_1, \quad V_3 = x_3 - \frac{x_3 \cdot V_1}{V_1 \cdot V_1} V_1 - \frac{x_3 \cdot V_2}{V_2 \cdot V_2} V_2 \]

with \( x_1 = \begin{pmatrix} 0 \\ 3 \end{pmatrix}, \quad x_2 = \begin{pmatrix} 4 \\ 1 \end{pmatrix}, \quad x_3 = \begin{pmatrix} 3 \\ 2 \end{pmatrix} \) we get:

\[ V_1 = \begin{pmatrix} 0 \\ -3 \end{pmatrix}, \quad V_2 = \begin{pmatrix} 4 \\ 1 \end{pmatrix}, \quad V_3 = \begin{pmatrix} 3 \\ 2 \end{pmatrix} \]

We can rewrite (putting \( V_i \)'s in order on one side):

\[ Q = \begin{pmatrix} 0 & 1 & 0 \\ -3 & 0 & 0 \end{pmatrix}, \quad R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \]

Thus:

\[ A = QR \]

\[ A = \begin{pmatrix} 0 & 1 & 0 \\ -3 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \]

\[ Q = \begin{pmatrix} -0.866 & 0.5 & 0 \\ -0.5 & -0.866 & 0 \end{pmatrix}, \quad R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \end{pmatrix} \]

\[ A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \]
Note $Q$ has an orthogonal columns and $R$ is right $\Delta$.

Application. To solve $\min \| b - A x \|$

$$Q^T b = \begin{pmatrix} 0 & \frac{4}{3 & 1} & 0 & \frac{8}{3} \end{pmatrix} \begin{pmatrix} 25 \ 10 \ 10 \ 5 \end{pmatrix} = \begin{pmatrix} 4 \sqrt{10} \ 5 \end{pmatrix}$$

$$R x = \begin{pmatrix} 4 \sqrt{10} & 3 \sqrt{10} & 2 \sqrt{10} \ 0 & 5 & 0 \ 0 & 0 & 0 \end{pmatrix} x = \begin{pmatrix} 4 \sqrt{10} \ 5 \ 0 \end{pmatrix}$$

$$x = \begin{pmatrix} -\frac{4}{5} \\ 0 \\ 1 \end{pmatrix}$$

$\text{(a)}$ Solve $\min \| b - A x \|$ with $A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 3 & 1 \\ 2 & -1 & 0 \end{pmatrix}$ and $b = \begin{pmatrix} 10 \\ 4 \end{pmatrix}$

For your convenience here is the result of Gram-Schmidt orthogonalization:

$$v_1 = \begin{pmatrix} 0 \\ 2 \\ 0 \end{pmatrix}; \ v_2 = \begin{pmatrix} 3 \\ 3 \\ 0 \end{pmatrix}; \ v_3 = \begin{pmatrix} 4 \\ 0 \\ 3 \end{pmatrix}$$

What are your $A \in R$?

(b) What are the $QER$ produced by Matlab?

$[Q, R] = qr(A, 0)$. Except for signs they should be the same as your $QER$. Are they?

(c) What does Matlab's $x = A \backslash b$ give. Is it the same as your $x$?
Singular Value Decomposition (SVD) - Using the SVD to solve an underdetermined, singular system.

Consider $Ax = b$ where $A = \begin{bmatrix} 2 & 1 & 3 & 2 \\ 1 & 2 & 1 & 1 \\ 3 & 4 & 3 & 3 \end{bmatrix}$, $x = \begin{bmatrix} 8 \\ 4 \\ 3 \end{bmatrix}$.

$A$ is $m \times n$ where $m = 3 \neq n = 4$ and so the system is underdetermined and the solution to $Ax = b$ is not unique. In addition $A$ is singular since the last row of $A$ is the sum of the first two rows (the last row is dependent on the first two rows).

The best (most accurate) way to solve such systems is to use the SVD.

**Solve via the SVD**

1. Form $A = UDV^T$ with $m \times n$ $\mathbb{R}$ diagonal $D$ orthogonal $U \in m \times m$ diagonal $V \in n \times n$ orthogonal $\lambda_i$ diagonal.

2. Let $k$ be the number of nonzero singular values.

3. Let $\hat{u}_i$ be the first $k$ columns of $U$, $\hat{v}_i$ be the first $k$ columns of $V$.

4. Let $\hat{x} = \hat{V}D^{-1}\hat{U}^Tb$; of all the solutions to $Ax = b$, $\hat{x}$ is the solution with smallest Euclidean norm.

5. All possible solutions to $Ax = \hat{x}$ are given by $x = \hat{x} + V_0z$ where $z$ is any vector in $\mathbb{R}^{n-k}$.

**Remark** If $Ax = b$ has no solution (the system is inconsistent) then everything above is invalid if we replace $Ax = b$ with $\min \|b - Ax\|_2$.
For this problem (using Matlab) \( \begin{bmatrix} 7.9199 & 0 & 0 \\ 0 & 2.346 & 0 \\ 0 & 0 & 4.9401 \\ \end{bmatrix} \)

\[ U, D, V ] = \text{svd}(A) \text{ with } D = \begin{bmatrix} 4.846 & -1.233 & 1.193 & -0.8587 \\ -0.3163 & -0.8065 & -0.4441 & 0.2297 \\ 0.599 & 0.562 & -0.4441 & 0.2297 \\ -0.4896 & -1.233 & 0.4676 & 3.984 \\ \end{bmatrix} \]

\[ U = \begin{bmatrix} 0.5473 & 0.6059 \\ 0.2511 & -0.7956 \\ -0.7974 & -0.1710 \\ \end{bmatrix}, \ V = \begin{bmatrix} 0.1846 & 1.1233 \\ 0.3163 & -0.8065 \\ 0.599 & 0.562 \\ -0.4896 & -1.233 \\ \end{bmatrix}, \ V_o = \begin{bmatrix} -0.4441 & 2.2971 \\ 0.4441 & 0.2297 \\ 0.9680 & 3.984 \\ \end{bmatrix} \]

(2) \( K = 2 \) since \( s_2 = 0.9101 \) is (numerically) zero

(3) \[ \hat{U} = \begin{bmatrix} -0.5473 & 0.6059 \\ 0.2511 & -0.7956 \\ -0.7974 & -0.1710 \\ \end{bmatrix}, \hat{V} = \begin{bmatrix} 0.1846 & 1.1233 \\ 0.3163 & -0.8065 \\ 0.599 & 0.562 \\ -0.4896 & -1.233 \\ \end{bmatrix}, \hat{V}_o = \begin{bmatrix} -0.4441 & 2.2971 \\ 0.4441 & 0.2297 \\ 0.9680 & 3.984 \\ \end{bmatrix} \]

\( \hat{U} \hat{D} \hat{V}^T b = (1) \) [In Matlab we can use \( \hat{U} \hat{V}^T b = \text{null}(A) \) \( \hat{V}_o \) is \( \text{null}(A) \)]

(4) \[ \hat{X} = \hat{V} \hat{D}^{-1} \hat{U}^T b = (1) \]

(5) all solutions to \( A \hat{X} = \hat{b} \) are \( \hat{X} = (1) + \begin{bmatrix} -0.4441 & 2.2971 \\ 0.4441 & 0.2297 \\ 0.9680 & 3.984 \\ \end{bmatrix} \) \( \hat{X}_1, \hat{X}_2 \)

For any such \( \hat{X} \), \( \| \hat{X} \|_2 \leq \| \hat{X}_1 \|_2 \leq \| \hat{X}_2 \|_2 \)

Note: Another way to get \( \hat{X} \) is \( \hat{X} = \hat{p} \hat{v} \hat{v}^T (A^T A) \) \( \hat{p} \)

Another way to get \( \hat{V}_o \) is \( \hat{V}_o = \text{null}(A) \)

(\text{HW})\) Consider \( A \hat{X} = \hat{b} \) with \( A = \begin{bmatrix} 3 & 2 & 4 & 3 \\ 4 & 2 & 5 & 4 \\ 2 & 3 & 5 & 2 \\ 4 & 3 & 5 & 4 \\ \end{bmatrix} \) \( \hat{b} = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 6 \\ \end{bmatrix} \)

(\text{\text{a}) Carry out steps (1) to (4). Matlab printouts of the steps are fine.}

(\text{b}) Write out a formula for all solutions \( \hat{X} \)

(\text{c}) Calculate \( \text{pinv}(A) \hat{d} \) to \( \text{null}(A) \) using Matlab. Do they agree with \( \hat{X} \) and \( \hat{V}_o \)?
The MathWorks News & Notes - October 2006

Cleve's Corner
By Cleve Moler

Professor SVD

Stanford computer science professor Gene Golub has done more than anyone to make the singular value decomposition one of the most powerful and widely used tools in modern matrix computation.

Gene Golub's license plate, photographed by Professor P. M. Kroonenberg of Leiden University.

The SVD is a recent development. Pete Stewart, author of the 1993 paper "On the Early History of the Singular Value Decomposition", tells me that the term valeurs singulières was first used by Emile Picard around 1910 in connection with integral equations. Picard used the adjective "singular" to mean something exceptional or out of the ordinary. At the time, it had nothing to do with singular matrices.

When I was a graduate student in the early 1960s, the SVD was still regarded as a fairly obscure theoretical concept. A book that George Forsythe and I wrote in 1964 described the SVD as a nonconstructive way of characterizing the norm and condition number of a matrix. We did not yet have a practical way to actually compute it. Gene Golub and W. Kahan published the first effective algorithm in 1965. A variant of that algorithm, published by Gene Golub and Christian Reinsch in 1970 is still the one we use today. By the time the first MATLAB appeared, around 1980, the SVD was one of its highlights.

We can generate a 2-by-2 example by working backwards, computing a matrix from its SVD. Take \( \theta_1 = 2, \theta_2 = 1/2, \theta = \pi/6 \) and \( \phi = \pi/4 \).

Let

\[
U = \begin{pmatrix}
-\cos \theta & \sin \theta \\
\sin \theta & \cos \theta
\end{pmatrix}
\]

\[
\Sigma = \begin{pmatrix}
\sigma & 0 \\
0 & \sigma
\end{pmatrix}
\]

\[
V = \begin{pmatrix}
-\cos \phi & \sin \phi \\
\sin \phi & \cos \phi
\end{pmatrix}
\]

The matrices \( U \) and \( V \) are rotations through angles \( \theta \) and \( \phi \), followed by reflections in the first dimension. The matrix \( \Sigma \) is a diagonal scaling transformation. Generate \( A \) by computing

\[
A = UEV^T
\]

You will find that

\[
A = \begin{pmatrix}
1.4015 & -1.0480 \\
-1.0091 & 1.0133
\end{pmatrix}
\]

This says that the matrix \( A \) can be generated by a rotation through 45\( ^\circ \) and a reflection, followed by independent scalings in each of the two coordinate directions by factors of 2 and 1/2, respectively, followed by a rotation through 30\( ^\circ \) and another reflection.

The MATLAB function eigshow generates a figure that demonstrates the singular value decomposition of a 2-by-2 matrix. Enter the statements

\[
A = \begin{pmatrix}
1.4015 & -1.0480 \\
-1.0091 & 1.0133
\end{pmatrix};
eigshow(A)
\]

Click the SVD button and move the mouse around. You will see Figure 1, but with different labels.

![Figure 1. SVD figure produced by eigshow. Click on image to see enlarged view.](image-url)
The green circle is the unit circle in the plane. The blue ellipse is the image of this circle under transformation by the matrix $A$. The green vectors, $v_1$ and $v_2$, which are the columns of $V$, and the blue vectors, $u_1$ and $u_2$, which are the columns of $U$, are two different orthogonal bases for two-dimensional space. The columns of $V$ are rotated 45° from the axes of the figure, while the columns of $U$, which are the major and minor axes of the ellipse, are rotated 30°. The matrix $A$ transforms $v_1$ into $\sigma_1 u_1$ and $v_2$ into $\sigma_2 u_2$.

Let’s move on to $m$-by-$n$ matrices. One of the most important features of the SVD is its use of orthogonal matrices. A real matrix $U$ is orthogonal, or has orthonormal columns, if

$$U^T U = I$$

This says that the columns of $U$ are perpendicular to each other and have unit length. Geometrically, transformations by orthogonal matrices are generalizations of rotations and reflections; they preserve lengths and angles. Computationally, orthogonal matrices are very desirable because they do not magnify roundoff or other kinds of errors.

Any real matrix $A$, even a nonsquare one, can be written as the product of three matrices.

$$A = U \Sigma V^T$$

The matrix $U$ is orthogonal and has as many rows as $A$. The matrix $V$ is orthogonal and has as many columns as $A$. The matrix $\Sigma$ is the same size as $A$, but its only nonzero elements are on the main diagonal. The diagonal elements of $\Sigma$ are the singular values, and the columns of $U$ and $V$ are the left and right singular vectors.

In abstract linear algebra terms, a matrix represents a linear transformation from one vector space, the domain, to another, the range. The SVD says that for any linear transformation it is possible to choose an orthonormal basis for the domain and a possibly different orthonormal basis for the range. The transformation becomes independent of scalings or dilatations in each coordinate direction.

The rank of a matrix is the number of linearly independent rows, which is the same as the number of linearly independent columns. The rank of a diagonal matrix is clearly the number of nonzero diagonal elements. Orthogonal transforms preserve linear independence. Thus, the rank of any matrix is the number of nonzero singular values. In MATLAB, enter the statement

`type rank`

to see how we choose a tolerance and count nonnegligible singular values.

Traditional courses in linear algebra make considerable use of the reduced row echelon form (RREF), but the RREF is an unreliable tool for computation in the face of inexact data and arithmetic. The SVD can be regarded as a modern, computationally powerful replacement for the RREF.

A square diagonal matrix is nonsingular if, and only if, its diagonal elements are nonzero. The SVD implies that any square matrix is nonsingular if, and only if, its singular values are nonzero. The most numerically reliable way to determine whether matrices are singular is to test their singular values. This is far better than trying to compute determinants, which have atrocious scaling properties.

With the singular value decomposition, the system of linear equations

$$Ax = b$$

becomes

$$U \Sigma V^T x = b$$

The solution is

$$x = V^T \Sigma^{-1} U^T b$$

Multiply by an orthogonal matrix, divide by the singular values, then multiply by another orthogonal matrix. This is much more computational work than Gaussian elimination, but it has impeccable numerical properties. You can judge whether the singular values are small enough to be regarded as negligible, and if they are, analyze the relevant singular system.

Let $E_k$ denote the outer product of the $k$-th left and right singular vectors, that is

$$E_k = u_k v_k^T$$

Then $A$ can be expressed as a sum of rank-1 matrices,

$$A = \sum_{k=1}^{n} \sigma_k E_k$$

If you order the singular values in decreasing order, $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$, and truncate the sum after $r$ terms, the result is a rank-$r$ approximation to the original matrix. The error in the approximation depends upon the magnitude of the neglected singular values. When you do this with a matrix of data that has been centered, by subtracting the mean of each column from the entire column, the process is known as principal component analysis (PCA). The right singular vectors, $v_k$, are the components, and the scaled left singular vectors, $\sigma_k u_k$, are the scores. PCAs are usually described in terms of the eigenvalues and eigenvectors of the covariance matrix, $A^T A$, but the SVD approach sometimes has better numerical properties.

SVD and matrix approximation are often illustrated by approximating images. Our example starts with the photo on Gene Golub’s
Web page (Figure 2). The image is 897-by-598 pixels. We stack the red, green, and blue JPEG components vertically to produce a 2691-by-598 matrix. We then do just one SVD computation. After computing a low-rank approximation, we repartition the matrix into RGB components. With just rank 12, the colors are accurately reproduced and Gene is recognizable, especially if you squint at the picture to allow your eyes to reconstruct the original image. With rank 50, you can begin to read the mathematics on the white board behind Gene. With rank 120, the image is almost indistinguishable from the full rank 598. (This is not a particularly effective image compression technique. In fact, my friends in image processing call it "image degradation.")

Figure 2. Rank 12, 50, and 120 approximations to a rank 598 color photo of Gene Golub. Click on images to see enlarged view.

So far in this column I have hardly mentioned eigenvalues. I wanted to show that it is possible to discuss singular values without discussing eigenvalues—but, of course, the two are closely related. In fact, if \( A \) is square, symmetric, and positive definite, its singular values and eigenvalues are equal, and its left and right singular vectors are equal to each other and to its eigenvectors. More generally, the singular values of \( A \) are the square roots of the eigenvalues of \( A^*A \) or \( AA^* \).

Singular values are relevant when the matrix is regarded as a transformation from one space to a different space with possibly different dimensions. Eigenvalues are relevant when the matrix is regarded as a transformation from one space into itself—as, for example, in linear ordinary differential equations.

Google finds over 3,000,000 Web pages that mention "singular value decomposition" and almost 200,000 pages that mention "SVD MATLAB." I knew about a few of these pages before I started to write this column. I came across some other interesting ones as I surfed around.

Professor SVD made all of this, and much more, possible. Thanks, Gene.

A Few Results Search for "Singular Value Decomposition"
- The Wikipedia pages on SVD and PCA are quite good and contain a number of useful links, although not to each other. [en.wikipedia.org/wiki/Singular_value_decomposition](http://en.wikipedia.org/wiki/Singular_value_decomposition)
- en.wikipedia.org/wiki/Principal_component_analysis
- Rasmus Bro, a professor at the Royal Veterinary and Agricultural University in Denmark, and Barry Wise, head of Eigenvector Research in Wenatchee, Washington, both do chemometrics using \( SVD \) and \( PCA \). One example involves the analysis of the absorption spectrum of water samples from a lake to identify upstream sources of pollution. [www.models.kvl.dk/users/rasmus](http://www.models.kvl.dk/users/rasmus)
- [www.eigenvector.com](http://www.eigenvector.com)
- Tammy Kolda and Brett Bader, at Sandia National Labs in Livermore, ca, developed the Tensor Toolbox for MATLAB, which provides generalizations of \( PCA \) to multidimensional data sets. [csmr.ca.sandia.gov/~tkolda/TensorToolbox](http://csmr.ca.sandia.gov/~tkolda/TensorToolbox)
- In 2003, Lawrence Sirovich of the Mount Sinai School of Medicine published "A pattern analysis of the second Rehnquist U.S. Supreme Court" in the Proceedings of the US National Academy of Sciences. His paper led to articles in the New York Times and the Washington Post because it provides a nonpolitical, phenomenological model of court decisions. Between 1984 and 2002, the court heard 468 cases. Since there are nine justices, each of whom takes a majority or minority position on each case, the data is a 468-by-9 matrix of +1s and -1s. If the judges had made their decisions by flipping coins, this matrix would almost certainly have rank 9. But Sirovich found that the third singular value is an order of magnitude smaller than the first one, so the matrix is well approximated by a matrix of rank 2. In other words, most of the court's decisions are close to being in a two-dimensional subspace of all possible decisions. [www.pnas.org/cgi/reprint/100/13/7432](http://www.pnas.org/cgi/reprint/100/13/7432)
- The first Google hit on "protein svd" is "Protein Substate Modeling and Identification Using the SVD," by Tod Romo at Rice University. The site provides an electronic exposition of the use of \( SVD \) in the analysis of the structure and motion of proteins, and includes some gorgeous graphics. [bioe.rice.edu/~strom/Spregz/loc.html](http://bioe.rice.edu/~strom/Spregz/loc.html)
- Los Alamos biophysicists Michael Wall, Andreas Rechtsteiner, and Luis Rocha provide a good online reference about \( SVD \) and \( PCA \), phrased in terms of applications to gene expression analysis. [public.lanl.gov/mewall/kisweb2002.html](http://public.lanl.gov/mewall/kisweb2002.html)
- "Representing cyclic human motion using functional analysis" (2005). by Dirk Ormonde, Michael Black, Trevor Hastie, and Hedvig Kjelstrom, describes techniques involving Fourier analysis and principal component analysis for analyzing and modeling motion-capture data from activities such as walking. [www.csr.kth.se/~hedvig/publications/lyc_05.pdf](http://www.csr.kth.se/~hedvig/publications/lyc_05.pdf)
- A related paper is "Decomposing biological motion: a framework for analysis and synthesis of human gait patterns," (2002), by Nicholas Trejo. Trejo's work is the basis for an "eigenwalker" demo. [www.journalofvision.org/2/5/2](http://www.journalofvision.org/2/5/2) [www.mathworks.com/matlab/ncm/walker.m](http://www.mathworks.com/matlab/ncm/walker.m)
- A search at the US Patent and Trademark Office Web page list 1,197 U.S. patents that mention "singular value decomposition." The oldest, issued in 1997, is for "A fiber optic inspection system for use in the inspection of sandwiched..."
solder bonds in integrated circuit packages". Other titles include "Compression of surface light fields", "Method of seismic surveying", "Semantic querying of a peer-to-peer network", "Biochemical markers of brain function", and "Diabetes management."

www.uspto.gov/patft

Resources:
- On the Early History of the Singular Value Decomposition
- Cleves Corner Collection

Help us improve The MathWorks News & Notes!
Take a short survey

1. Least Squares

If A is an m by n rectangular matrix with m \geq n and if c is a vector with n components then A \times c = y can not usually be solved exactly. Matlab can solve these equations approximately with the command c = A \backslash y. The solution c is called the “least squares” solution for reasons that we will describe later in the course. To illustrate an application involving this use of a rectangular matrix suppose that we wish to fit the three points (0,3), (2,5), and (4,6) with a straight line. Then as we described earlier we can let the line be described by c_1 \times x + c_2 \times x, let c = [c_1; c_2] (using Matlab notation for vectors), let y = the vector of y values and let A be a Vandermonde matrix with components a_{ij} = x^{i-1}. To fit the line to the data we need to try to solve the equation A \times c = y. In this example one can create A with the Matlab command A = [1 0; 1 2; 1 4] and y with y = [3; 5; 6]. Then c = A \backslash y gives c = [3.1667; .75]. The commands

x=[0; 2; 4] % x isthe vector of x coordinates
plot(x,y,'o',x,A*c) % note that A*c generates points on the best fit line
xlabel('x')
ylabel('y')
title('given points (o) and least squares fit (line)')

will plot the given points and the line 3.1667 + .75 x. See the figure below.

For homework consider the points (0,0), (1,2), (2,3), (3,9), (4,17), (5,24), (6,37). (1) Use Matlab to find the least squares best fit with a line. Turn in A, y and c and a plot like the one above. Also (2) use Matlab to find the least square best fit with a quadratic. Turn in the same information.
Computer Project: Good Vibrations? –Buildings and Bridges

Purpose: To introduce vibration analysis, the concept of resonance and to provide a physical interpretation of eigenvalues. Also to illustrate the type of analysis required for every large structure in an earthquake zone like California and to explain the picture to the right.

Prerequisite: Understanding of eigenvalues, eigenvectors and diagonalization of a matrix. For example Sections 5.1-5.3 of Elementary Linear Algebra by Spence, Insel and Friedberg or Sections 5.1-5.3 in Linear Algebra and its Applications by David Lay.

MATLAB functions and notation used: eig, real, angle, abs, exp, for, *, i, plot, grid, xlabel, ylabel, title, diag, expm, zeros, eye, ones

What to turn in: Either
- print this assignment, add answers by hand in the spaces provided and attach clearly labeled printouts of any graphs requested or
- (preferable) download the assignment from my web page (www.math.sjsu.edu/~foster/ml29as06.html) as a Word file, cut (from Matlab’s figure window use Edit / Copy Figure), paste and resize (to make the picture smaller) Matlab figures as you answer each question. Also use word or by hand write the answers to the questions.

Note that the items that require a response from you are in bold below.

Background: Consider a complex number \( z = x + iy \), where \( i = \sqrt{-1} \). We can plot this point in the complex plane where the horizontal axis indicates the real part \( x \) of \( z \) and the vertical axis indicates the imaginary part \( y \) of \( z \). Consider the polar form of this point. If \( r \) is the distance of the point to the origin we have \( r^2 = x^2 + y^2 \) and if \( \theta \) is the angle between the positive \( x \) axis and the point then \( x = r \cos \theta \) and \( y = r \sin \theta \). For example if \( z = \sqrt{2}/2 + i\sqrt{2}/2 \) then \( r^2 = (\sqrt{2}/2)^2 + (\sqrt{2}/2)^2 = 1 \) and \( \theta \) is the angle with \( x = \sqrt{2}/2 = r \cos \theta = \cos \theta \) and \( y = \sqrt{2}/2 = r \sin \theta = \sin \theta \) or, using properties of trigonometric functions, \( \theta = 45^\circ \) or \( \pi/4 \) radians. Note that \( r \) is called the magnitude or absolute value of the complex number \( z \).

It will be useful to use Euler’s formula \( e^{i\theta} = \cos \theta + i \sin \theta \). Since \( x = r \cos \theta \) and \( y = r \sin \theta \) we can then write
\[
z = x + iy = r (\cos \theta + i \sin \theta) = re^{i\theta}.
\]

(1)

For example if \( z = \sqrt{2}/2 + i\sqrt{2}/2 \) then \( z = 1 \) \( e^{i\pi/4} = e^{i\pi/4} \). An important use of Euler’s formula is to calculate
\[
z^k = (re^{i\theta})^k = r^k e^{ik\theta} = r^k [\cos (k \theta) + i \sin (k \theta)].
\]

(2)

Now let us assume that \( r = 1 \) and consider \( z^k \) as a function of \( k \). In this case, by (2), the real part of \( z^k \) oscillates following the cosine curve \( \cos (k \theta) \) and the imaginary part of \( z^k \) oscillates following the sine curve \( \sin (k \theta) \). As a function of \( k \) the real and imaginary part of \( z^k \) will repeat every time \( k \) \( \theta \) increases by an amount \( 2 \pi \). Therefore, as a function of \( k \), the period of \( z^k \) is \( 2 \pi / \theta \). Since the period is \( 2 \pi / \theta \) then \( 0 \) is called the (angular) frequency of \( z^k \). In the above example \( z^k = e^{i(\pi/4)k} \) so that the period of \( z^k \) is \( (2 \pi) / (\pi/4) = 8 \) and the frequency is \( \pi/4 \).

We will use Euler’s formula to examine the solution \( y_0, y_1, y_2, \ldots, y_k, \ldots \) to
\[
y_{k+1} = Ay_k \text{ given } y_0 \text{ is known}
\]

(3)

and to
\[
y_{k+1} = Ay_k + \text{real}(z^k \nu) \text{ given } y_0, \nu \text{ and } z \text{ are known}
\]

(4)

In these equations \( k \) can be thought of as a measure of time or the number of time steps and the solution vectors \( y_0, y_1, y_2, \ldots, y_k, \ldots \) to (3) or (4) can be thought of the evolution of the initial vector \( y_0 \) over time. In the case of interest to us we will assume that \( A \) is a real \( n \) by \( n \) matrix whose eigenvalues have magnitudes equal to one and that \( z \) is a complex number whose magnitude is also one. In equation (4) \( \text{real}(z^k \nu) \) indicates the real part of \( z^k \nu \) where \( \nu \) is a vector with \( n \) complex
components. The term real($z^k v$) is called the forcing function. Equation (3) is called an “unforced” system and equation (4) is called a “forced” system.

Since we will assume that the eigenvalues of $A$ have magnitude one the solution to (3) will involve periodic motion (see below). By this we mean that the solution $y_0, y_1, y_2, \ldots, y_k, \ldots$ to (3) will consist of combinations of functions that are periodic in $k$. Since periodic functions repeat after a period it follows that the solution $y_0, y_1, y_2, \ldots, y_k, \ldots$ to (3) will remain bounded as $k$ increases. Also since we are assuming that the magnitude of $z$ is one it follows that the forcing function real($z^k v$) will remain bounded as $k$ increases.

Now with our assumptions the solution to (3) remains bounded and involves periodic functions and in (4) the forcing function is also bounded and periodic. A critical question in many applications is whether the solution to (4) is also bounded.

1. Consider the matrix $A = \begin{bmatrix} 0.95533649 & 0.9850674 \\ -0.088656062 & 0.95533649 \end{bmatrix}$ and the initial value $y_0 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$.

(a) Solve the unforced system (3) for $k = 0$ to 200 and plot the results. You can do this with the Matlab code below. You do not need to type in the comments (which begin at the % sign in each line):

```matlab
z = 0; % this makes the forcing function real($x^k v$) below zero
y0=[-1; 1]; % initial position
steps = 200; % number of time steps
v = y0; % set components of forcing function vector (not used for $z = 0$)

P = y0'; % P is used to remember motion
y = y0; % set first y to initial position y0
for k = 1: steps % loop over time
    y = A * y + real($z^k v$); % calculate y at next time step
    P = [P ; y']; % remember history of motion
end
plot(P,'-'), shg % draw a plot of motion
grid % put grid lines on plot
xlabel('time') % label horizontal axis
ylabel('y') % label vertical axis
```

Turn in your plot and write a sentence or two describing the plot. Do the oscillations grow as $k$ increases?

(b) Calculate the eigenvalues $\lambda_1$ and $\lambda_2$ of $A$ using the Matlab command eig(A). Write each eigenvalue in polar form so that $\lambda_1 = r_1 e^{i\theta_1}$ and $\lambda_2 = r_2 e^{i\theta_2}$. In Matlab abs(eig(A)) will give the magnitudes of the eigenvalues and angle(eig(A)) will give the angles.

(c) As described in class and the text, that if $A$ is diagonalizable so that $A = V D V^{-1}$ with $D$ diagonal then the solution to (3), $y_k = A^k y_0$, can be written as $y_k = V D^k V^{-1} y_0$. Let $u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = V^{-1} y_0$ and $V = [v_1, v_2]$. Then using $y_k = V D^k V^{-1} y_0$ we can write $y_k = v_1 \lambda_1^k u_1 + v_2 \lambda_2^k u_2$. Since $\lambda_1 = r_1 e^{i\theta_1}$ and $\lambda_2 = r_2 e^{i\theta_2}$ with $r_1 = 1$ and $r_2 = 1$ it follows $v_1 \lambda_1^k u_1$ is periodic with
frequency $\theta_1$, and that $v_2 \lambda_2^k u_2$ is periodic with frequency $\theta_2$. These values of $\theta$ are called the natural frequencies of oscillation of the system. The result is important enough that we will put it in a separate box:

If the eigenvalues of the matrix $A$ in (3) have magnitude one then the natural frequencies of oscillation of the system (3) are the angles in the polar representation of the eigenvalues.

The graph from part (a) can be used to support this comment. By looking at the plot in part (a) determine, approximately, the period of oscillation? How does this compare with the value calculated by the formula $(2 \pi) / \theta$ for $\theta = \theta_1$?

2. In most real world applications, for example in a mathematical model of a building or a bridge, the systems (3) and (4) involve matrices that are much bigger than the two by two matrix used in problem 1. An important class of systems of the form (3) and (4) have matrices that can be generated, for an even number $n \geq 2$, by

$$m = n / 2;$$
$$B = -\text{diag(ones(m-1,1),-1)} + \text{diag(2*ones(m,1))} -\text{diag(ones(m-1,1),1)};$$
$$A = \expm([\text{zeros}(m,m)\text{,} \text{eye}(m,m)\text{,} -B, \text{zeros}(m,m)\text{,}]);$$

Note that the Matlab function diag produces matrices that are zero except for entries on a diagonal and the use of diag above produces a matrix $B$ that has 2's down the main diagonal, and -1's on the diagonals next to the main diagonal. The function expm in Matlab calculates the exponential of a matrix. We won't define this here. These matrices arise from an approximate solution to a differential equation called the “wave equation”. We won't present their derivation from this application here but we can still illustrate their use. Let us consider such a matrix for $n = 40$.

(a) Unforced motion: For the above matrix $A$ with $n = 40$ solve the unforced system (3) for $k = 0$ to 300 and plot the results. You can do this with the Matlab code below which is a modification of the code in Question 1. Note that you can cut and paste the code from Word into Matlab (after assigning $n = 40$ and executing the above commands, if you wish.

$$z = 0; \quad \text{% this make the forcing function real( x^k*v ) below zero}$$

$$[V, D] = \text{eig}(A);$$
$$y0 = V*\text{ones}(n,1); \quad \text{% these two commands set an initial position of the structure}$$
$$\text{steps} = 300; \quad \text{% number of time steps}$$
$$v = y0; \quad \text{% set components of forcing function (not used if z = 0)}$$
$$P = y0'; \quad \text{% P is used to remember the motion}$$
$$y = y0; \quad \text{% set first y to initial position y0}$$
$$\text{for k = 1: steps} \quad \text{% loop of time steps}$$
$$\quad y = A*y + \text{real(z^k*v}); \quad \text{% calculate y at next time step}$$
$$\quad P = [P; y']; \quad \text{% remember motion}$$
$$\text{end}$$
$$\text{location = round(n / 7); \quad \% we will plot the motion at a particular location}$$
$$\text{plot(P(:,location),'-'), shg \quad \% draw a plot}$$
$$\text{grid \quad \% add grid lines to the plot}$$
$$\text{xlabel('time (perhaps sec)')} \quad \text{% label horizontal axis}$$
$$\text{ylabel('y (perhaps inches)')} \quad \text{% label vertical axis}$$
$$\text{title('Motion of structure (at your desk)')} \quad \text{% give graph a title}$$

Think of the plot as a picture of the motion of your desk in the classroom versus time. The above model would describe the motion of your desk if, at the beginning of the calculation, the structure (perhaps MacQuarrie Hall) had an initial vibration but there was no excitation or forcing function to add to this initial vibration. Turn in your plot and write a sentence or two
describing the plot (in terms of what you would feel at your desk assuming the vertical scale in the plot is in inches). Do the solutions grow as $k$ increases?

(b) Use the Matlab command angle(eig(A))' to determine the natural frequencies of oscillation of the system. Note that prime in angle(eig(A))' will make it easier to see the natural frequencies on one screen. List the natural frequencies here. Since there are so many you can cut and paste from Matlab if you wish. Note that the motion in part (a) was more complicated than the motion in problem 1 because the solution is a combination of 20, not just 2, natural frequencies of oscillation.

(c) **Forced motion:** We can now simulate an earthquake by letting $z$ be nonzero so that the forcing function is not zero. Let $z = \exp(0.4 * i)$ rather than $z = 0$, in the your code and solve the system (4). To do this you can replace the line “$z = 0$” with “$z = \exp(0.4 * i)$” in the Word file and copy the new code into Matlab. An alternative is to cut and past the above code into a file called “resonance.m” (say), put a % in front of the line “$z = 0$” to make it an inactive comment, save the file, then from the Matlab prompt (not inside resonance.m) define “$z = \exp(0.4 * i)$” and then type “resonance” from the Matlab prompt. This will execute all the commands in the file resonance.m using the $z$ value $\exp(0.4 i)$. Here the frequency of the forcing function (the earthquake) is $\theta = 0.4$ which does not match any of the natural frequencies in part (b). **Turn in your plot and write a sentence or two describing what you would feel at your desk in this earthquake. Do the solutions grow as $k$ increases?**

(d) Now let $z = \exp(\theta i)$ for $\theta = \text{ the positive natural frequency determined in part (b)}$ that is closest to zero (so $z = \exp(0.1495 * i)$). For this $z$ solve (4) using the earlier code. **Turn in your plot and write a sentence or two describing what you would feel at your desk in this earthquake (recall that we are assuming the vertical scale in the plot is in inches). Do the oscillations continue to grow as $k$ increases?** If you had to make a choice would you rather be in the earthquake in part (c) of part (d). **Why?** Note that the term resonance is used for the case that a bounded forcing function causes the solution to (4) to have oscillations that grow as $k$ increases.
(e) Repeat the calculations in part (d) for other forcing function frequencies \( \theta \) by redefining \( z = \exp(\theta \, i) \) for various values of \( \theta \). Try values of the forcing frequency \( \theta \) that match a natural frequency (such as 0.2981, 0.4450, 0.5895, etc.) and values of \( \theta \) not near any natural frequency (for example 0.2, 0.4, 0.5, etc.). Based on your results make a conjecture about the values of the frequency of the forcing function that result in resonance. **State you conjecture by filling in the blank in the box below. Include a few graphs supporting your conclusion for this question.**

Resonance occurs when the frequency of the forcing function equals to _______________________.
When resonance occurs the solution to equation (4) grows without bound.

Based on your results do you think that it is important to design buildings so that the natural frequencies of oscillation of the structure are far from the forcing frequencies that may occur in an earthquake. Why?

4. Here are some pictures of the Tacoma Narrows Bridge:

These pictures (source http://www.lib.washington.edu/specialcoll/tnb ) show the Tacoma Narrows bridge on opening day, July 1, 1940 (top left) and on November 7, 1940 when a wind of approximately 40 mph caused an oscillation in the bridge that eventually led to its collapse. In this case the wind, not an earthquake, is the forcing function. **Use the concept of resonance to present an explanation for the collapse of the bridge.**
3. Solving a differential equation approximately by solving a (square) linear system

```matlab
% This is a Matlab script file which solves a simple
differential equation numerically.
% The differential equation solved is u''(x) = -sin(x), u(0)=0, u(pi)=0.
The only input to this program is n, the number of subintervals of
the interval 0 to pi.
The output is graphs of the approximate and true solution. Also the values
of the approximate solution at the points is contained in the vector u
and the variable maxerror is the maximum difference between the true
and approximate solution.

%construct the matrix
v1 = -ones(n-2,1);
v2 = 2 * ones(n-1,1);
A = diag(v1,-1) + diag(v2,0) + diag(v1,1);
%construct the right hand side
h = pi / n;
x = [h * (1 : (n-1))'];
b = sin(x) * h^2;
%solve the system
u = A \ b;
%add zeros onto the end of x and add corresponding points to ti
u = [0 ; u ; 0];
x = [0 ; x ; pi];
%plot the numerical solution (u) and the true solution (sin(x))
true = sin(x);
xforgraph = pi * (0:.02:1);
trueforgraph = sin(xforgraph);
plot(x,u,'+',xforgraph,trueforgraph)
%calculate the maximum error
error = u - true;
maxerror = max(abs(error))
xlabel('x')
ylabel('u')
title('numerical (+) versus exact solution (solid) for u''''=-sin(x)')
```

Homework:
(1) Run ode (the file ode.m is on my web site – www.mathcs.sjsu.edu/faculty/foster) with various values of n to find the
smallest n so that the error <= 10^-4. Turn in your n value and a brief description of how you got it.
(2) Try several values of n and experimentally determine the relation between n and the error. When you double n what
happens to the error? Try to discover, by experiment, an equation relating the error and n. I am looking for an equation
like maxerror = 1.75 / n^2 (this formula is not correct -- the power and coefficient will be different). Turn in your formula.
(3) Change my ode program so that you approximately solve u''(x) = -cos(x), u(0) = u(pi) = 0. (Note that the true u(x) is the
second indefinite integral of -cos(x). This is not cos(x) -- there is a constant of integration needed to satisfy u(0) = u(pi) = 0.)
Turn in the graph for n = 10 and the maxerror for n = 5, n = 10 and n = 50.
4. Approximating a picture using the singular value decomposition

If \( A \) is an \( m \times n \) matrix with \( m \geq n \) the singular value decomposition is \( A = UDV^T \) where \( U \) and \( V \) are orthogonal and \( D \) is diagonal. The diagonal entries of \( D \), \( s_1 \geq s_2 \geq \cdots \geq s_n \), are called the singular values of \( A \). As described in class we can write \( UDV^T \) using outer products as

\[
A = UDV^T = \sum_{i=1}^{n} s_i u_i v_i^T,
\]

(1)

where \( u_i, i=1,\ldots,n \), are columns of \( U \) and \( v_i, i=1,\ldots,n \), are columns of \( V \). It is useful at times to truncate the sum \( \sum_{i=1}^{n} s_i u_i v_i^T \) keeping only \( k \leq n \) terms. We get

\[
A_k = \sum_{i=1}^{k} s_i u_i v_i^T = U(:,1:k)*D(1:k,1:k)*V(:,1:k)'.
\]

(2)

In equation (1) we have introduced some Matlab notation: \( U(:,1:k) \) consists of the first \( k \) columns of \( U \), \( V(:,1:k) \) consists of the first \( k \) columns of \( V \) and \( D(1:k,1:k) \) consists of the first \( k \) rows and first \( k \) columns of \( D \). \( A_k \) has some desirable properties including \( \| A - A_k \|_2 \leq \| A - B \|_2 \) for any \( B \) with \( k \) or fewer independent columns. Therefore, when using the 2 norm, there is no better approximation of rank \( k \) to \( A \) than \( A_k \). If \( k \) is small \( A_k \) can be stored by saving the \( s_i \)'s, \( u_i \)'s and \( v_i \)'s and this will require much less storage than storing all of \( A \). Use of \( A_k \) is a scheme for data compression - representation of an object that requires high storage (\( A \)) with one that requires low storage (\( A_k \)).

To picture this we first note that a matrix can represent a surface over a rectangular region. The value of the entries in the matrix will correspond to the height of the surface. For example for the matrix \( A \) defined by

\[
A = \begin{pmatrix}
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0
\end{pmatrix}
\]

and the picture looks approximately like a plus. If we create a large matrix with the same pattern of zeros and ones we get a better picture of a plus but more storage is required. If \( A \) is 90 by 90 (with bands of 30 rows and 30 columns set to 1) then we get a nice picture of a plus but \( A \) requires storing 8100 = 90 * 90 pieces of data.

We can dramatically compress the picture represented by this \( A \) as follows: if \( s = \text{svd}(A) \) then \( s_1 = 61.9 \), \( s_2 = 29.9 \), \( s_3 = 1.56 \times 10^{-14} \), \( s_4 = 6.6 \times 10^{-16} \), \ldots where all the \( s_i \)'s for \( i > 2 \) are exceedingly small. There is a large gap in the singular values in this case. In the sum (1) \( s_3, s_4, \ldots, s_90 \) are all so small that they can be ignored and therefore

\[
A_2 = s_1 u_1 v_1^T + s_2 u_2 v_2^T
\]

is an excellent approximation to \( A \). We can check this in Matlab by forming (with \( k = 2 \))

\[
[U,D,V] = \text{svd}(A); \\
A_k = U(:,1:k)*D(1:k,1:k)*V(:,1:k)';
\]

(3)

mesh(Ak)

The resulting picture looks exactly like mesh(A) but storage of \( s_1, s_2, u_1, u_2, v_1, v_2 \) requires only \( 1+1+90 + 90 + 90 + 90 + 90 = 362 \) values, not \( 8100 \) values. In this case we have a reduction in storage by a factor of more than 20. Note that one need not form \( A_k \) but could directly do mesh( \( U(:,1:k) \)* \( D(1:k,1:k) \)* \( V(:,1:k) \) )

Homework:

(1) I wrote a small Matlab program letterF.m (on my web site – www.mathcs.sjsu.edu/faculty/foster) which creates a matrix \( A \) such that mesh(A) looks like an F. Let \( n = 50 \) and try to get a reduced rank approximation to this matrix. Do this by finding a gap in the singular values: let \( s = \text{svd}(A) \) and select \( k \) by ignoring any small singular values. Follow this with the commands (3). Turn in the value of \( k \) you selected and your plot mesh(Ak).

(2) Do the same for my program letterA.m for which mesh(A) looks like an A. This case is not as easy (due to the A's slanted lines). Is there a gap in the singular values of A? What is the smallest value of \( k \) for which you get a recognizable picture of an A? a good picture of an A? Turn in both values of \( k \), comments about the gap, and the good picture.

(3) Run Matlab's penny demonstration which uses a 128 by 128 matrix to picture a penny. Now run my app_penny.m (at my web site). The program app_penny.m approximates the penny picture using a rank \( k \) (\( k \) is input to the program) approximation. What value of \( k \) gives a recognizable penny? What value of \( k \) gives a good picture of a penny? Turn in these values of \( k \) (but no pictures). Is there a gap in the singular values?

(4) Is the truncated svd a useful tool for approximating pictures? Briefly give your opinion.
MATLAB GUIDE, 08/09, Leslie Foster

Matlab is an easy to use environment for solving problems involving matrices and drawing two and three dimensional graphs of their solutions. Matlab is interactive with all variables automatically saved. On line help facilities are provided as well as demonstrations illustrating Matlab features. The core of Matlab is a large number of powerful matrix functions that usually can be invoked with one command. For problems that cannot be directly solved with these built functions, the ability to write programs is provided. This guide provides an overview to some but certainly not all Matlab commands and features.

Basics

The most recent version of Matlab is version 7.8. The version that I will lend you is Matlab 5.3. It is close enough to 7 to still be quite good. For some of our work 5.3 is better since it has a command (flops) to keep track of the amount of work in a calculation. The student lab in MH221 has Matlab 7 available. The version that I lend uses windows and requires 95MB on my computer. There is a student edition of Matlab which includes (full) Matlab and a few additional features. It costs around $100.

I suggest that you run "demo" when you enter Matlab for the first time in order to see some of the features.

In the following examples capital letters indicate matrices and small letters indicate vectors.

Matrix and vector definition

1. \( v_1 = [1 \ 2 \ 3] \) defines a row vector whose components are 1, 2 and 3
2. \( v_1 = [1, 2, 3] \) creates the same vector
3. \( v_2 = [4; 5; 6; 7] \) defines a column vector whose components are 4, 5, 6, and 7
4. \( v_2 = [4 5 6 7]' \) uses transpose (') to create the same vector
5. \( A = [1 2 3 4; 5 6 7 8; 9 10 11 12] \) creates a matrix whose first row contains 1, 2, 3, and 4, whose second row contains 5, 6, 7 and 8 and whose last row contains 9, 10, 11 and 12

Matrix operations

1. \( \text{rref}(A) \) calculates the reduced row echelon form of the matrix \( A \).
2. \( \text{rref}(A,B) \) shows the major steps in calculating the reduced row echelon form of a matrix.
3. \( A * v_2 \) is the product of the matrix \( A \) and vector \( v_2 \)
4. \( A * B \) is the matrix product of matrices \( A \) and \( B \)
5. \( A \backslash B \) is \( \text{inv}(A) * B \), where \( \text{inv}(A) \) indicates the inverse of \( A \)
6. \( A / B \) is \( A * \text{inv}(B) \)
7. \( A^4 \) will produce the fourth power of \( A = A * A * A * A \)
8. \( x = A \backslash b \) is the solution to \( A x = b \). If \( A \) is not square this will produce a least squares approximate solution.
9. \( A' \) is the transpose of \( A \)
10. \( \text{sum}(v_2) \) sums all the elements of a vector
11. \( \text{inv}(A) \) is the inverse of \( A \)
12. \( \text{pinv}(A) \) is the generalized inverse of \( A \) (for rectangular matrices)
13. \( \text{cond}(A) \) is the condition number of \( A \)
14. \( \text{norm}(v_2) \) is the usual length or norm of a vector (square root of the sum of the squares of components)
15. \( \text{norm}(A) \) is the norm of \( A \) in the two norm, \( \text{norm}(A,1) \) is the one norm, \( \text{norm}(A,\infty) \) the infinity norm
16. \( \text{rank}(A) \) is the rank of \( A \)
17. \( \text{null}(A) \) determines a basis for the null space of \( A \)
18. \( \text{det}(A) \) is the determinant of \( A \)
19. \( \text{eig}(A) \) produces a vector consisting of the eigenvalues of \( A \)
20. \( [V,D] = \text{eig}(A) \) will produce the eigenvalues in a diagonal matrix \( D \) and the eigenvectors, columns of \( V \)
21. \( \text{svd}(A) \) produces a vector consisting of the singular values of \( A \)
22. \( [U,D,V] = \text{svd}(A) \) gives singular values (diagonal entries in \( D \)) and singular vectors (columns of \( V \) and \( U \)
Some utilities
1. cd - to move to a different working folder. It is more convenient in Matlab 5.3 to click on the Path Browser (its picture has two folders) on the menu bar, click on “Browse” and then navigate to a desired folder. In Matlab 7 click on the three dots next to “Current Directory” on the menu bar and navigate.
2. who - lists all currently defined variables
3. whois - lists the size of all currently defined variables
4. save filename - save on the disk file named all current variables
5. load filename - loads variables that have been previously saved on file filename
6. clear - clear all variables (Note: Matlab always saves all variable used until they are cleared.)
7. clear name - erase the variable named
8. format rat -- display the results with fractions (close to the true value)
9. format short e - displays results using 5 digits in scientific notation
10. format long e - displays results using 16 digits in scientific notation
11. format compact - makes the output on the screen more compact
12. Note that by default the result of any operation is displayed on the screen. Ending a line with a ";" will suppress this. A line of code without a semicolon is the simplest way to see output.

Building vectors and matrices
1. A(2,3) refers to the entry in the second row and third column of A
2. A(:,3) refers to the entire third column of A
3. A(1,:) refers to the entire first column of A
4. v = 1:4 will produce a row vector with components 1 2 3 and 4
5. v = 4:-1:1 will produce a row vector with components 4 3 2 and 1
6. A=rand(m,n) produces an m by n matrix with random number entries
7. A=ones(m,n) produces an m by n matrix of ones
8. A=eye(n) produces an n by n identity matrix
9. V = diag(A) selects the diagonal of A.
10. A=diag(v) produces a matrix whose diagonal has components of the vector v
11. A=diag(v,n) produces a matrix that is zero except that one diagonal contains the elements of v. If n = 1 it is the first superdiagonal and if n=-1 the diagonal is the first subdiagonal.
12. A=[B; C] puts matrix B on top of matrix C building a longer matrix
13. A=[B, C] puts C to the right of B building a wider matrix
14. A([1 3],[2 3]) will produce a 2 by 2 submatrix consisting of the intersection of row 1 and 3 and columns 2 and 3 of A
15. triu(A), tril(A) selects the upper and lower triangular part of A

Built-in Functions
- log, sqrt, tan, sinh, exp, bessel and many more: function(A) (for example log(A)) will produce a matrix whose components are the function applied to each component of A. Also there are functions available (such as the power of a matrix) which form functions of the whole matrix rather than apply the function component by component.

Plotting
1. plot(x,y) - If x and y are vectors this will produce a two dimensional plot of x(i) vs y(i) connecting points by straight lines.
2. plot(x,y,'+') - This will mark the points plotted with a +.
3. copy figure (from the figures edit menu) - to copy a figure to the clipboard
4. grid - draws grid lines
5. xlabel and ylabel - to label the x and y axis
6. legend and title - to include a legend or a title on the graph
7. shg and figure(some number) - to show the current graph or some other (as numbered) figure
8. mesh(A) - This will produce a three dimensional plot where the height plotted corresponds to the value of the ij component of A
9. many other facilities are available- semilog and polar plots; histograms; several graphics windows

Miscellaneous
1. flops (in Matlab 5.3 but not 6.5) - lists the number of floating point operations used in all calculations since Matlab was initiated
2. max and min - the largest and smallest entries of a vector
3. sort(v) - sort v in ascending order

**plus many others**
Programming

Structured programming constructs such as for loops, while loops, if - elseif statements, and subprograms are available if necessary. The simplest way to write a program is to create a script file. This is a disk file consisting of one or more Matlab commands. The disk file should be stored on your current directory and must have a ".m" extension. While inside Matlab typing the name of the file (with no ".m") will cause the execution of all the lines of the file, just as if you had typed them from the keyboard. Lines in a script file beginning with a "%" are comments and typing "help filename" will list any comments that are at the beginning of your script file. This is Matlab's standard procedure for providing help facilities. Script files are easily created as described below. Here is a Matlab program that adds up the even integers less than or equal to n

\[ n = 10; \quad \% \text{ this run is for } n = 10 \]
\[ \text{sum } = 0; \]
\[ \text{for } k = 1:n \]
\[ \quad \text{if ( mod(k,2) } = 0 ) \]
\[ \quad \text{sum } = \text{sum } + k; \]
\[ \text{end} \quad \% \text{ mod is the integer remainder, } = = \text{ test equality} \]
\[ \text{end} \quad \% \text{ end of the if statement} \]
\[ \text{end} \quad \% \text{ end of the loop} \]
\[ \text{sum} \quad \% \text{ display sum (since the line does not end with a semi-colon )} \]

Editing programs

Type edit and an edit window will open up. Also from the file menu select a new file or select open to open an existing file. Make sure that you save the file before you try to use it from Matlab.

Printouts
1. select print from the file menu to print all your work so far (this will probably be too much)
2. select the desired printout on the screen (using the mouse) and from the menus select file, print selection. Perhaps even more convenient is to paste the selection into Word and print it later.
3. to print a graph, select file, print from the graphs window
4. copy figure from the edit menu in the graph's window is useful to cut and later paste a figure into Word. This is the best way to print figures since one can control the size of the figure in Word.
5. diary filename- This will save everything, except graphs, that appears on the screen on a disk file.
   This disk file can then be printed out the same way that any file would be printed on your computer.
6. diary- This toggles diary on or off

References
1. Be sure to make use of the demonstrations (type “demo”) and help files (type “help” or “help function-name”) from the Matlab prompt.
2. Jane Day’s web site http://www.math.sisu.edu/~day/layprojects.html has useful projects. In particular the first project (Getting Started with Matlab) is an excellent tutorial.
4. www.math.utah.edu/lab/ms/matlab/matlab.html is an internet site with an elementary primer on Matlab.
5. http://www.mathworks.com/academia/student_center/tutorials/ has a Matlab tutorial. A google search for matlab tutorials will get lots of hits. Also see http://www.duke.edu/~hpgavin/matlab.html
7. www.mathworks.com is the Mathworks (the producer of Matlab) homepage
8. Also, you can look for the USENET newsgroup "comp.soft-sys.matlab" for tutorials. The newsgroup is a forum for discussing issues related to the use of MATLAB.
9. Finally, check the table of contents / index of your text book.
Example script file (ode.m)

% This is a Matlab script file which solves a simple
differential equation numerically.
% The differential equation solved is u''(x) = -sin(x), u(0)=0, u(pi)=0.
The only input to this program is n, the number of subintervals of
the interval 0 to pi.
The output are graphs of the approximate and true solution. Also the values
of the approximate solution at the points is contained in the vector u
and the variable maxerror is the maximum difference between the true
and approximate solution.

% Leslie Foster, 1-31-00

% construct the matrix
v1 = -ones(n-2,1);
v2 = 2 * ones(n-1,1);
A = diag(v1,-1) + diag(v2,0) + diag(v1,1);

% construct the right hand side
h = pi / n;
xi = h * (1 : (n-1))';
b = sin(xi) * h^2;

% solve the system
u = A \ b;

% add zeros onto the end of x and add corresponding points to xi
u = [0 ; u ; 0];
xi = [0 ; xi ; pi];

% plot the numerical solution (u) and the true solution (sin(x))
true = sin(xi);
xforgraph = pi * (0:.02:1);
tureforgraph = sin(xforgraph);
plot(xi,u,'+',xforgraph,trueforgraph)

% calculate the maximum error
error = u - true;
maxerror = max(abs(error))
xlabel('x')
ylabel('u')
title('numerical (+) versus exact solution (solid) for u''''=-sin(x)')

Example Runs:

% n=4;
% ode
maxerror = 0.0530

% n=8;
% ode
maxerror = 0.0130

% n=32;
% ode
maxerror = 8.0358e-04