

Computational Linear Algebra

- **Course:** (MATH: 6800, CSCI: 6800)
- **Semester:** Fall 1998
- **Instructors:**
 - Joseph E. Flaherty, flaherje@cs.rpi.edu
 - Franklin T. Luk, luk@cs.rpi.edu
 - Wesley Turner, turnerw@cs.rpi.edu

Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180

OUTLINE

1. Introduction
 - 1.1 Notation
 - 1.2 Special matrices
2. Gaussian Elimination
 - 2.1 Vector and matrix norms
 - 2.2 Finite precision arithmetic
 - 2.3 Factorizations
 - 2.4 Pivoting
 - 2.5 Accuracy estimation
3. Special Linear Systems
 - 3.1 Symmetric positive definite systems
 - 3.2 Banded and profile systems
 - 3.3 Block tridiagonal systems
 - 3.4 Symmetric indefinite systems
 - 3.5 Profile methods
 - 3.6 Sparse Systems
4. Orthogonalization and Least Squares
 - 4.1 Orthogonality and the singular value decomposition
 - 4.2 Rotations and reflections
 - 4.3 **QR** factorizations
 - 4.4 Least squares problems

OUTLINE

5. The Algebraic Eigenvalue Problem
 - 5.1 Introduction
 - 5.2 Power methods
 - 5.3 Hessenberg and Schur forms. The **QR** algorithm
 - 5.4 Symmetric problems
 - 5.5 Jacobi iteration
6. Iterative Methods
 - 6.1 Fixed-point methods
 - 6.2 The basic conjugate gradient method
 - 6.3 Preconditioning
 - 6.4 Krylov and generalized-residual methods
7. Introduction to Parallel Computing
 - 7.1 Shared- and distributed-memory computation
 - 7.2 Matrix multiplication
 - 7.3 Gaussian elimination

References

1. O. Axelsson (1994), *Iterative Solution Methods*, Cambridge, Cambridge.
2. R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, V. Pozo, C. Romine, and H. van der Vorst (1994), *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia.
3. Å. Björck (1996), *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia.
4. W. Briggs (1987), *A Multigrid Tutorial*, SIAM, Philadelphia.
5. J.W. Demmel (1997), *Applied Numerical Linear Algebra*, SIAM, philadelphia.
6. A. George and J. Liu (1981), *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs.
7. G.H. Golub and C.F. Van Loan (1993), *Matrix Computations*, Third Edition, Johns Hopkins, Baltimore.
8. A. Greenbaum (1997), *Iterative Methods for Solving Linear Systems*, SIAM Philadelphia.

9. W. Hackbusch (1994), *Iterative Solution of Large Sparse Linear Systems of Equations*, Springer-Verlag, Berlin.
10. N. Higham (1996), *Accuracy and Stability of Numerical Algorithms*, SIAM Philadelphia.
11. B. Parlett (1980), *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs.
12. Y. Saad (1996), *Iterative Methods for Sparse Linear Systems*, PWS, Boston.
13. B. Smith, P. Bjorstad, and W. Gropp (1996), *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge, Cambridge.
14. G.W. Stewart (1973), *Introduction to Matrix Computations*, Academic Press, New York.
15. G.W. Stewart and J.-G. Sun (1990) *Matrix Perturbation Theory*, Academic Press, New York.
16. L.N. Trefethen and D. Bau, III (1997), *Numerical Linear Algebra*, SIAM, Philadelphia. ¹
17. P. Wesseling (1992), *An Introduction to Multigrid Methods*, Wiley, Chichester.

¹Text

18. D. Young (1971), *Iterative Solution of Large Linear Systems*, Academic Press, New York.

1. Introduction

1.1. Notation

- *Motivation:*
 - Linear algebra is involved in approximately 75% of scientific computation
 - * Circuit and network analysis
 - * Potential theory
 - * Signal processing
 - * Acoustics and vibrations
 - * Optimization
 - * Finite difference and finite element methods
 - * Tomography
 - * Optimal control
- *Problems:*
 - Solve linear systems
 - Solve least squares problems
 - Solve eigenvalue-eigenvector problems

Notation

- Reading: Trefethen and Bau (1997), Lecture 1
- An $m \times n$ matrix

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \quad (1)$$

- Bold letters denote vectors and matrices
- Italic letters denote scalars
 - * Lower case letters are elements of a matrix
- $a_{ij} \in \mathfrak{R}$, $i = 1, 2, \dots, m$, $j = 1, 2, \dots, n$, unless stated
- $\mathbf{A} \in \mathfrak{R}^{m \times n}$
 - * $\mathfrak{R}^{m \times n}$ a vector space of $m \times n$ real matrices
- A vector: $\mathbf{x} \in \mathfrak{R}^{m \times 1} := \mathfrak{R}^m$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}, \quad \mathbf{x}^T = [x_1, x_2, \cdots, x_m]$$

MATLAB

- Specify Algorithms using *MATLAB*
- *Vector dot (scalar) product*

$$c = \mathbf{x}^T \mathbf{y} = \sum_{i=1}^n x_i y_i \quad (2)$$

```
function c = dot(x, y)  
% dot: compute a dot product of the n-vectors x and y  
n = length(x)  
c = 0  
for i = 1:n  
    c = c + x(i)*y(i)  
end
```

- Subscripts are enclosed in parentheses
- The MATLAB function *length* computes the dimension of a vector
- The `:` indicates a range
 - * In the for loop, *i* ranges over integers from 1 to *n*
- `dot(x, y)` is a library function in MATLAB

A SAXPY

- A *saxpy* (“scalar a x plus y”) is the vector

$$\mathbf{z} = a\mathbf{x} + \mathbf{y} \tag{3}$$

```
function z = saxpy(a, x, y)  
% saxpy: compute the vector z = a * x + y where  
% a is a scalar and x and y are vectors  
n = length(x)  
for i = 1:n  
    z(i) = a*x(i) + y(i)  
end
```

- **x** and **y** have the same dimensions n
 - * Production software should check this

Matrix Multiplication

- Let $\mathbf{A} \in \mathfrak{R}^{m \times r}$, $\mathbf{B} \in \mathfrak{R}^{r \times n}$, and $\mathbf{C} \in \mathfrak{R}^{m \times n}$, then

$$c_{ij} = \sum_{k=1}^r a_{ik} b_{kj}, \quad i = 1 : m, \quad j = 1 : n \quad (4)$$

```
function C = matmy(A, B)
```

```
% matmy: compute the matrix product C = AB
```

```
[m r] = size(A);
```

```
[r n] = size(B);
```

```
for i = 1:m
```

```
    for j = 1:n
```

```
        C(i,j) = 0;
```

```
        for k = 1:r
```

```
            C(i,j) = C(i,j) + A(i,k)*B(k,j);
```

```
        end
```

```
    end
```

```
end
```

- `size(A)` returns the row and column dimensions of **A**
- The `;`s suppress printing of results
- The complexity is mnr multiplications and additions
- Some details are omitted

Matrix Multiplication

- Matrix multiplication using the dot product

- Let \mathbf{a}_j denote the j th column of \mathbf{A}

$$\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_r] \quad (5)$$

- Replace the loop in (4) by a dot product

$$c_{ij} = \mathbf{a}_i^T \mathbf{b}_j, \quad i = 1 : m, \quad j = 1 : n \quad (6)$$

* \mathbf{a}_i^T is the i th column of \mathbf{A}^T or the i th row of \mathbf{A}

function $\mathbf{C} = \text{matmy}(\mathbf{A}, \mathbf{B})$

% matmy: compute the matrix product $\mathbf{C} = \mathbf{AB}$ using

% the dot product

```
[m r] = size(A);
```

```
[r n] = size(B);
```

```
for i = 1:m
```

```
for j = 1:n
```

```
    C(i,j) = dot(A(i,1:r)',B(1:r,j));
```

```
end
```

```
end
```

- The range 1:r indicates an operation for an entire row or column
- \mathbf{A}' denotes transposition
- $\mathbf{A}(i, 1 : r)$ is the i th row of \mathbf{A}

Matrix Multiplication with Saxpys

- Reorder the loop structure: regard the columns of \mathbf{C} as linear combinations of those of \mathbf{A}

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 5 \begin{bmatrix} 1 \\ 3 \end{bmatrix} + 7 \begin{bmatrix} 2 \\ 4 \end{bmatrix} & 6 \begin{bmatrix} 1 \\ 3 \end{bmatrix} + 8 \begin{bmatrix} 2 \\ 4 \end{bmatrix} \end{bmatrix}$$

– compute $\mathbf{C} = \mathbf{AB}$ as

$$\mathbf{c}_j = \sum_{k=1}^r b_{kj} \mathbf{a}_k, \quad j = 1 : n \quad (7)$$

– Implement the sum using saxpys

```
function  $\mathbf{C} = \text{smatmy}(\mathbf{A}, \mathbf{B})$ 
% smatmy: compute the matrix product  $\mathbf{C} = \mathbf{AB}$ 
% using saxpys
[m r] = size( $\mathbf{A}$ );
[r n] = size( $\mathbf{B}$ );
 $\mathbf{C} = \text{zeros}(m, n)$ ;
for j = 1:n
    for k = 1:r
        C(1:m,j) = saxpy(B(k,j), A(1:m,k), C(1:m,j));
    end
end
```

– `zeros(m,n)` returns a $m \times n$ matrix of zeros.

Outer Products

- Reorder the loops to perform an outer product
 - An *outer product* of an m -vector \mathbf{x} and an n -vector \mathbf{y} is the $m \times n$ matrix

$$\mathbf{C} = \mathbf{xy}^T \quad (8)$$

or

$$\mathbf{C} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} [y_1 y_2 \cdots y_n] = \begin{bmatrix} x_1 y_1 & x_1 y_2 & \cdots & x_1 y_n \\ x_2 y_1 & x_2 y_2 & \cdots & x_2 y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_m y_1 & x_m y_2 & \cdots & x_m y_n \end{bmatrix}$$

- *Problem 1*: do matrix multiplication using outer products
 - * Show this for the 2×2 problem

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

Outer Product

* Write an algorithm for a general system

- Different loop structures perform differently on different computers

dot product i, j, k

saxpy j, k, i

outer product k, i, j