

Computational Methods
CMSC/AMSC/MAPL 460

Vectors, Matrices, Linear Systems, LU
Decomposition,

Ramani Duraiswami,
Dept. of Computer Science

Class Outline

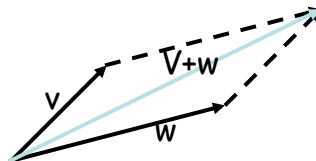
- Much of scientific computation involves solution of linear equations
 - Even non-linear problems are solved by linearization
- Matrices and vectors
- Matrix vector multiplication and complexity
 - Memory organization and access of elements
- Identity, Inverse, Singular Matrices
- Permutation, Lower and Upper Triangular Matrices

Vectors

- Ordered set of numbers: (1,2,3,4)
- Example: (x,y,z) coordinates of a point in space.
- Line joining the origin of coordinates to the point
- Vectors usually indicated with bold lower case letters. Scalars with lower case
- Operations with vectors:
 - Addition operation $\mathbf{u} + \mathbf{v}$, with:
 - Identity $\mathbf{0}$ $\mathbf{v} + \mathbf{0} = \mathbf{v}$
 - Inverse - $\mathbf{v} + (-\mathbf{v}) = \mathbf{0}$
 - Scalar multiplication:
 - Distributive rule: $\alpha(\mathbf{u} + \mathbf{v}) = \alpha(\mathbf{u}) + \alpha(\mathbf{v})$
 $(\alpha + \beta)\mathbf{u} = \alpha\mathbf{u} + \beta\mathbf{u}$

Vector Addition

$$\mathbf{v} + \mathbf{w} = (x_1, x_2) + (y_1, y_2) = (x_1 + y_1, x_2 + y_2)$$



Vector Spaces

- A *linear combination* of vectors results in a new vector:

$$\mathbf{v} = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \dots + \alpha_n \mathbf{v}_n$$

- If the only set of scalars such that

$$\alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \dots + \alpha_n \mathbf{v}_n = \mathbf{0}$$

is $\alpha_1 = \alpha_2 = \dots = \alpha_n = \mathbf{0}$

then we say the vectors $\{\mathbf{v}_i\}$ are *linearly independent*

- The *dimension* of a space is the greatest number of linearly independent vectors possible in a vector set
- For a vector space of dimension n , any set of n linearly independent vectors form a *basis*

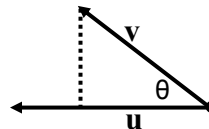
Dot Product

- The *dot product* or, more generally, *inner product* of two vectors is a scalar:

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = x_1 x_2 + y_1 y_2 + z_1 z_2 \quad (\text{in 3D})$$

- Useful for many purposes
 - Computing the length of a vector: $\text{length}(\mathbf{v}) = \sqrt{\mathbf{v} \cdot \mathbf{v}}$
 - *Normalizing* a vector, making it unit-length
 - Computing the angle between two vectors:

$$\mathbf{u} \cdot \mathbf{v} = |\mathbf{u}| |\mathbf{v}| \cos(\theta)$$
 - Checking two vectors for orthogonality
 - *Projecting* one vector onto another



Linear Transformations: Matrices

- A *linear transformation*:
 - Maps one vector to another
 - Preserves linear combinations
- Thus behavior of linear transformation is completely determined by what it does to a basis
- Turns out any linear transform can be represented by a *matrix*
- A $M \times N$ matrix takes a vector with N elements to a vector with M elements

Matrices

- By convention, matrix element \mathbf{M}_{ij} is located at row i and column j :

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_{11} & \mathbf{M}_{12} & \cdots & \mathbf{M}_{1n} \\ \mathbf{M}_{21} & \mathbf{M}_{22} & \cdots & \mathbf{M}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{M}_{m1} & \mathbf{M}_{m2} & \cdots & \mathbf{M}_{mn} \end{bmatrix}$$

- By convention, vectors are columns:

$$\mathbf{v} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \end{bmatrix}$$

How are matrices stored in a computer?

- Matlab and Fortran: column by column
 - Indices start at 1
 - What is the most efficient way to access a matrix?
- C arrays
 - Indices start at 0
 - closely linked to pointers
- C native matrices are row major
 - Alternate ways of construction possible
 - Many issues which must be dealt with by properly defining matrices, or using a set of matrix definitions

Some special matrices

Example: A **tridiagonal** matrix

$$T = \begin{bmatrix} 1 & 3 & 0 & 0 \\ 5 & 2 & 7 & 0 \\ 0 & 9 & 4 & 8 \\ 0 & 0 & 6 & 6 \end{bmatrix}$$

can be defined by

`T = diag([1 2 4 6]) + diag([5 9 6], -1) + diag([3 7 8], 1);`

- Matrices may be built up from “blocks” of smaller matrices

Some special matrices

Example: A **Vandermonde** matrix A is defined by a vector of elements x_1, \dots, x_n . Its first column is all ones. Each later column is the preceding one times this vector.

- Matlab code
- How many operations and memory does this take?
- Vectorized operations
- Matrix may be sparse, i.e. most elements are zero.
- How many operations/memory?
- Answer still N^2 unless we avoid referring to the zero elements altogether

```
n = length(x);
V(:,1) = ones(n,1);
for j=2:n,
    V(:,j) = x.*V(:,j-1);
end
```

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix}$$

```
D = diag([1 2 4 6]);
```

Matrix-vector product

- Matrix-vector multiplication applies a linear transformation to a vector:

$$\mathbf{M} \bullet \mathbf{v} = \begin{bmatrix} \mathbf{M}_{11} & \mathbf{M}_{12} & \mathbf{M}_{13} \\ \mathbf{M}_{21} & \mathbf{M}_{22} & \mathbf{M}_{23} \\ \mathbf{M}_{31} & \mathbf{M}_{32} & \mathbf{M}_{33} \end{bmatrix} \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \\ \mathbf{v}_z \end{bmatrix}$$

- Recall how to do matrix multiplication
- How many operations does this matrix vector product take?
- How many operations does a general matrix vector product take?

Ways to implement a matrix vector product

- Access matrix
 - Element-by-element along rows
 - Element-by-element along columns
 - As column vectors
 - As row vectors
- Discuss advantages

```
[m,n]=size(A);
y = zeros(m,1);
for i=1:m,
    for j=1:n,
        y(i) = y(i) + A(i,j)*x(j);
    end
end
```

```
[m,n]=size(A);
y = zeros(m,1);
for i=1:m,
    y(i) = A(i,:) * x;
end
```

```
[m,n]=size(A);
y = zeros(m,1);
for j=1:n,
    y = y + A(:,j)*x(j);
end
```