Lecture 5

Matrix, Vector Operations

David Semeraro

University of Illinois at Urbana-Champaign

September 10, 2013

Goals:

- recall linear algebra
- cost analysis of basic operations
- identify basic solution schemes to systems

Why this is important:

- matrix problems arise in many areas of computation (information sciences, graphics, design, etc)
- Basic Linear Algebra Subprograms (BLAS) is an interface standard for operations
- simple systems set the stage for further development: avoiding error, avoiding large costs

Prereq

Linear Algebra is a prerequisite of the course!

- you should feel comfortable with Appendix D.1 in NMC6
- Appendix D.2 in NMC6 should not be a surprise
- You should recognize:
  - Vector
  - Matrix
  - Matrix Vector product
  - Inner product
  - Scalar vector multiply
  - Vector norms

Matrix Inverse

Let A be a square (i.e. $n \times n$) with real elements. The *inverse* of A is designated A^{-1}, and has the property that

$$A^{-1} A = I \qquad \text{and} \qquad A A^{-1} = I$$

The **formal solution** to $Ax = b$ is $x = A^{-1}b$.

$$\begin{aligned} Ax &= b \\ A^{-1} Ax &= A^{-1}b \\ Ix &= A^{-1}b \\ x &= A^{-1}b \end{aligned}$$

Matrix Inverse

- formal solution to $Ax = b$ is $x = A^{-1}b$

Matrix Inverse

- formal solution to $Ax = b$ is $x = A^{-1}b$
- BUT it is *bad* to evaluate x this way

Matrix Inverse

- formal solution to $Ax = b$ is $x = A^{-1}b$
- BUT it is *bad* to evaluate x this way
- why?

Matrix Inverse

- formal solution to $Ax = b$ is $x = A^{-1}b$
- BUT it is *bad* to evaluate x this way
- why?
- we will not form A^{-1}, but solve for x directly using Gaussian elimination.

Why do we care?

Open questions:

- How *expensive* is it to solve $Ax = b$?
- What problems (errors) will we encounter solving $Ax = b$?
- Some matrices are easy/cheap to use: diagonal, tridiagonal, etc.
  - are there others? what makes something a ”good” matrix numerically?
  - are there bad ones? how do we identify them numerically?
- what do actual numerical analysts, engineers, developers, etc use?!?!

Formal Solution when A is $n \times n$

The *formal solution* to $Ax = b$ is

$$x = A^{-1}b$$

where A is $n \times n$.
If A^{-1} exists then A is said to be **nonsingular**.
If A^{-1} does not exist then A is said to be **singular**.

Formal Solution when A is $n \times n$

If A^{-1} exists then

$$Ax = b \quad \Longrightarrow \quad x = A^{-1}b$$

but

Do not compute the solution to $Ax = b$ by finding A^{-1}, and then multiplying b by A^{-1}!

We see: $x = A^{-1}b$

We do: **Solve $Ax = b$ by Gaussian elimination or an equivalent algorithm**

Singularity of A

If an $n \times n$ matrix, A, is **singular** then

- the columns of A are linearly dependent
- the rows of A are linearly dependent
- $\text{rank}(A) < n$
- $\det(A) = 0$
- A^{-1} does not exist
- a solution to $Ax = b$ may not exist
- If a solution to $Ax = b$ exists, it is not unique

Summary of Requirements for Solution of $Ax = b$

Given the $n \times n$ matrix A and the $n \times 1$ vector, b

- the solution to $Ax = b$ exists and is unique for any b if and only if $\mathrm{rank}(A) = n$.

Recall: rank $=$ # of linearly independent rows or columns

Recall: $Range(A)$ $=$ set of vectors y such that $Ax = y$ for some x

Solving a system

$$Ax = b$$

Three situations:

1. A is nonsingular: There exists a unique solution $x = A^{-1}b$
2. A is singular and $b \in Range(A)$: There are infinite solutions.
3. A is singular and $b \notin Range(A)$: There no solutions.

1. $A = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}$ $b = \begin{bmatrix} 1 \\ 8 \end{bmatrix}$, then $x = \begin{bmatrix} 1/2 \\ 2 \end{bmatrix}$.
2. $A = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$ $b = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, then infinitely many solutions. $x = \begin{bmatrix} 1/2 \\ \alpha \end{bmatrix}$.
3. $A = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$ $b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, then no solutions.

What's the big deal? ....cost

Look at 1D:

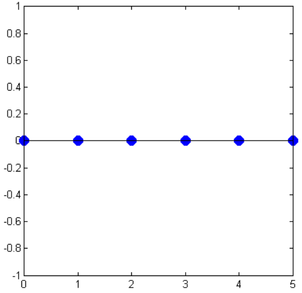
- 3 equations
- 3 unknowns
- each unknown coupled to its neighbor

$$
\begin{array}{cccl}
2x_1 & -x_2 & & = 5.8 \\
-x_1 & 2x_2 & -x_3 & = 13.9 \\
 & -x_2 & 2x_3 & = 0.03
\end{array}
$$

or

$$
\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}
\begin{bmatrix} 5.8 \\ 13.9 \\ 0.03 \end{bmatrix}
$$

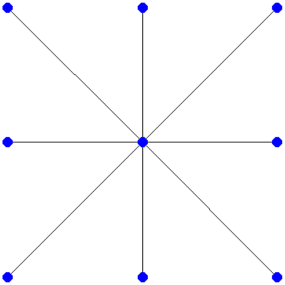
Easy in 1d

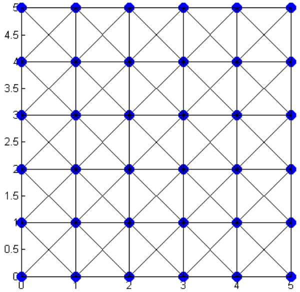


$$\begin{bmatrix} 2 & -1 & & & \\ & \ddots & & & \\ & -1 & 2 & -1 & \\ & & & \ddots & \\ & & & -1 & 2 \end{bmatrix}$$

- n points in the grid
- $3 * (n-2) + 2 + 2$ or about $3n$ nonzeros in the matrix
- trididagonal (easy)

2D: harder

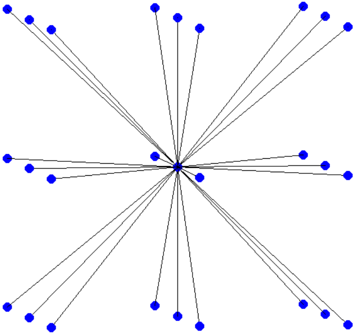


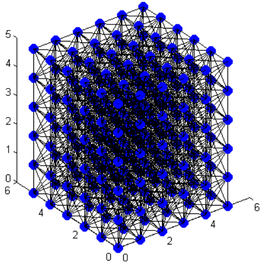


$$\begin{bmatrix} 8 & -1 & & -1 & & \\ & \ddots & & & & \\ -1 & -1 & 8 & -1 & -1 & \\ & & & \ddots & & \\ & -1 & & & -1 & 8 \end{bmatrix}$$

- n points in one direction
- n^2 points in grid
- about 9 nonzeros in each row
- about $9n^2$ nonzeros in the matrix
- n-banded (harder...we will see)

3D: hardest

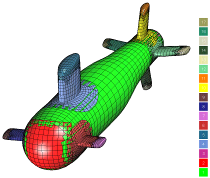




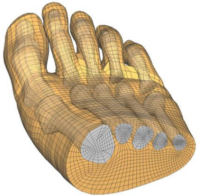
- n points in one direction
- n^3 points in grid

- about 27 nonzeros in each row
- about $27n^3$ nonzeros in the matrix
- n^2-banded (yikes!)

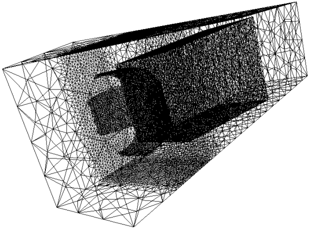
Applications get harder and harder...



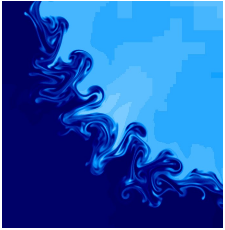
courtesy of LLNL



courtesy of TrueGrid



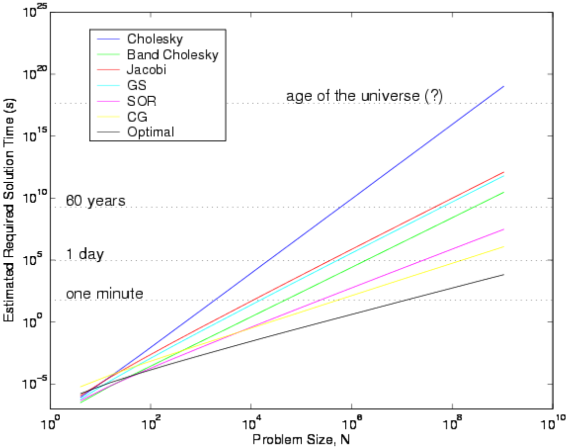
courtesy of Rice



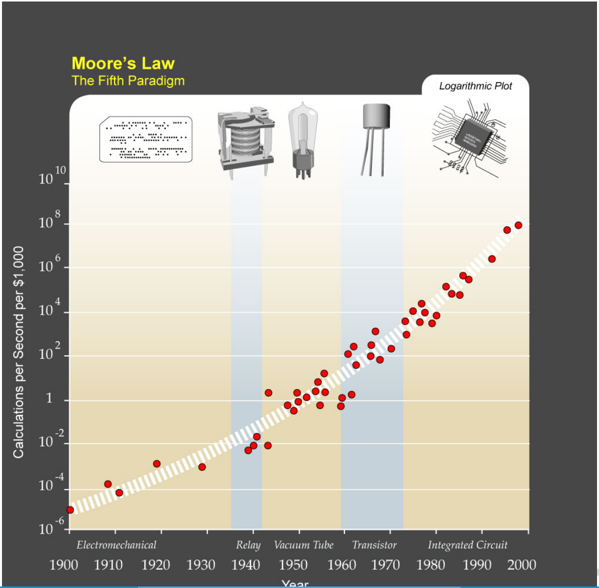
courtesy of Warwick U.

Solving is a problem...

dim	unknowns	storage
1D	n	$3n$
2D	n^2	$9n^2$
3D	n^3	$27n^3$



Moore...


Moore's Law
The Fifth Paradigm
Logarithmic Plot
Calculations per Second per $1,000
10^{10}
10^{8}
10^{6}
10^{4}
10^{2}
1
10^{-2}
10^{-4}
10^{-6}
Electromechanical
Relay
Vacuum Tube
Transistor
Integrated Circuit
1900 1910 1920 1930 1940 1950 1960 1970 1980 1990 2000
Year

What's the problem?

- humans: milliFLOPS
- hand calculators: 10 FLOPS
- desktops: a few GFLOPS (10^9 FLOPS)

look at the basic operations!

Big-O

How to measure the impact of n on algorithmic cost?

$\mathcal{O}(\cdot)$

Let $g(n)$ be a function of n. Then define

$$\mathcal{O}(g(n)) = \{f(n) \mid \exists c, n_0 > 0 \,:\, 0 \leqslant f(n) \leqslant cg(n),\ \forall n \geqslant n_0\}$$

That is, $f(n) \in \mathcal{O}(g(n))$ if there is a constant c such that $0 \leqslant f(n) \leqslant cg(n)$ is satisfied.

- assume non-negative functions (otherwise add $|\cdot|$) to the definitions
- $f(n) \in \mathcal{O}(g(n))$ represents an asymptotic upper bound on $f(n)$ up to a constant
- example: $f(n) = 3\sqrt{n} + 2\log n + 8n + 85n^2 \in \mathcal{O}(n^2)$

Big-O (Omicron)

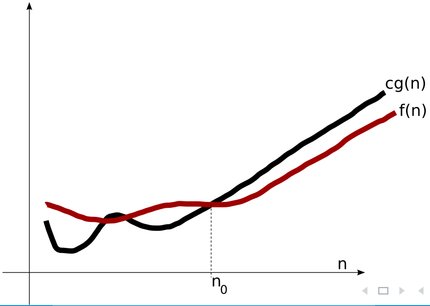
asymptotic upper bound

$\mathcal{O}(\cdot)$

Let $g(n)$ be a function of n. Then define

$$\mathcal{O}(g(n)) = \{f(n) \mid \exists c, n_0 > 0 : 0 \leqslant f(n) \leqslant cg(n), \forall n \geqslant n_0\}$$

That is, $f(n) \in \mathcal{O}(g(n))$ if there is a constant c such that $0 \leqslant f(n) \leqslant cg(n)$ is satisfied.



Big-Omega

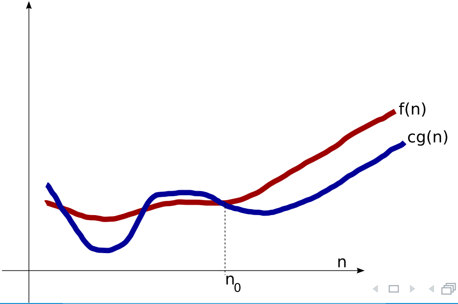
asymptotic lower bound

$\Omega(\cdot)$

Let $g(n)$ be a function of n. Then define

$$\Omega(g(n)) = \{f(n) \mid \exists c, n_0 > 0 \,:\, 0 \leqslant cg(n) \leqslant f(n),\ \forall n \geqslant n_0\}$$

That is, $f(n) \in \Omega(g(n))$ if there is a constant c such that $0 \leqslant cg(n) \leqslant f(n)$ is satisfied.



Big-Theta

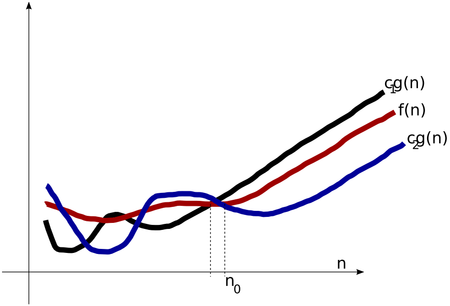
asymptotic tight bound

$\Theta(\cdot)$

Let $g(n)$ be a function of n. Then define

$$\Theta(g(n)) = \{f(n) \mid \exists c_1, c_2, n_0 > 0 : 0 \leqslant c_1 g(n) \leqslant f(n) \leqslant c_2 g(n), \forall n \geqslant n_0\}$$

Equivalently, $\Theta(g(n)) = \mathcal{O}(g(n)) \cap \Omega(g(n))$.



BLAS

Basic Linear Algebra Subprograms (BLAS) interface introduced APIs for common linear algebra tasks

- Level 1: vector operations (dot products, vector norms, etc) e.g.

$$y \leftarrow \alpha x + y$$

- Level 2: matrix-vector operations, e.g.

$$y \leftarrow \alpha Ax + By$$

- Level 3: matrix-matrix operations, e.g.

$$C \leftarrow \alpha AB + \beta C$$

- optimized versions of the reference BLAS are used everyday: ATLAS, etc.

vec-vec, mat-vec, mat-mat

- inner product of u and v both $[n \times 1]$

$$\sigma = u^T v = u_1 v_1 + \cdots + u_n v_n$$

- $\rightarrow n$ multiplies, $n-1$ additions
- $\rightarrow \mathcal{O}(n)$ flops

vec-vec, mat-vec, mat-mat

- mat-vec of A ($[n \times n]$) and u ($[n \times 1]$)

```
for i = 1,...,n
  for j = 1,...,n
    v(i) = a(i,j)u(j) + v(i)
  end
end
```

- $\rightarrow n^2$ multiplies, n^2 additions
- $\rightarrow \mathcal{O}(n^2)$ flops

vec-vec, mat-vec, mat-mat

- mat-mat of A ($[n \times n]$) and B ($[n \times n]$)

```
for j = 1,...,n
   for i = 1,...,n
      for k = 1,...,n
         C(k,j) = A(k,i)B(i,j) + C(k,j)
      end
   end
end
```

- $\rightarrow n^3$ multiplies, n^3 additions
- $\rightarrow \mathcal{O}(n^3)$ flops

vec-vec, mat-vec, mat-mat

Operation	FLOPS
$u^T v$	$\mathcal{O}(n)$
Au	$\mathcal{O}(n^2)$
AB	$\mathcal{O}(n^3)$

Gaussian Elimination

- Solving Diagonal Systems
- Solving Triangular Systems
- Gaussian Elimination Without Pivoting
  - Hand Calculations
  - Cartoon Version
  - The Algorithm
- Gaussian Elimination with Pivoting
  - Row or Column Interchanges, or Both
  - Implementation

Solving Diagonal Systems

The system defined by

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{bmatrix} \qquad b = \begin{bmatrix} -1 \\ 6 \\ -15 \end{bmatrix}$$

Solving Diagonal Systems

The system defined by

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{bmatrix} \qquad b = \begin{bmatrix} -1 \\ 6 \\ -15 \end{bmatrix}$$

is equivalent to

$$\begin{array}{rcrcrcr} x_1 & & & & & = & -1 \\ & & 3x_2 & & & = & 6 \\ & & & & 5x_3 & = & -15 \end{array}$$

Solving Diagonal Systems

The system defined by

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{bmatrix} \qquad b = \begin{bmatrix} -1 \\ 6 \\ -15 \end{bmatrix}$$

is equivalent to

$$\begin{array}{ccccc} x_1 & & & = & -1 \\ & 3x_2 & & = & 6 \\ & & 5x_3 & = & -15 \end{array}$$

The solution is

$$x_1 = -1 \qquad x_2 = \frac{6}{3} = 2 \qquad x_3 = \frac{-15}{5} = -3$$

Solving Diagonal Systems

Listing 1: Diagonal System Solution

```
given A, b
for i = 1...n
    x_i = b_i / a_{i,i}
end
```

In Python:

```
>>> import numpy as np
>>> A = np.array(...            %  A is a diagonal matrix
>> b = np.array(...
>> x = np.linalg.solve(A,b)
```

This is the *only* place where element-by-element division (`./`) has anything to do with solving linear systems of equations.

Operations?

Try...

Sketch out an operation count to solve a diagonal system of equations...

Operations?

Try...

Sketch out an operation count to solve a diagonal system of equations...

cheap!

one division n times $\longrightarrow \mathcal{O}(n)$ FLOPS

Triangular Systems

The generic lower and upper triangular matrices are

$$L = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & & 0 \\ \vdots & & \ddots & \vdots \\ l_{n1} & & \cdots & l_{nn} \end{bmatrix}$$

and

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & & u_{2n} \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & u_{nn} \end{bmatrix}$$

The triangular systems

$$Ly = b \qquad Ux = c$$

are easily solved by **forward substitution** and **backward substitution**, respectively

Solving Triangular Systems

$$A = \begin{bmatrix} -2 & 1 & 2 \\ 0 & 3 & -2 \\ 0 & 0 & 4 \end{bmatrix} \qquad b = \begin{bmatrix} 9 \\ -1 \\ 8 \end{bmatrix}$$

Solving Triangular Systems

$$A = \begin{bmatrix} -2 & 1 & 2 \\ 0 & 3 & -2 \\ 0 & 0 & 4 \end{bmatrix} \qquad b = \begin{bmatrix} 9 \\ -1 \\ 8 \end{bmatrix}$$

is equivalent to

$$\begin{array}{rcrcrcr} -2x_1 & + & x_2 & + & 2x_3 & = & 9 \\ & & 3x_2 & + & -2x_3 & = & -1 \\ & & & & 4x_3 & = & 8 \end{array}$$

Solving Triangular Systems

$$A = \begin{bmatrix} -2 & 1 & 2 \\ 0 & 3 & -2 \\ 0 & 0 & 4 \end{bmatrix} \qquad b = \begin{bmatrix} 9 \\ -1 \\ 8 \end{bmatrix}$$

is equivalent to

$$\begin{array}{rcrcrcr} -2x_1 & + & x_2 & + & 2x_3 & = & 9 \\ & & 3x_2 & + & -2x_3 & = & -1 \\ & & & & 4x_3 & = & 8 \end{array}$$

Solve in backward order (last equation is solved first)

$$x_3 = \frac{8}{4} = 2$$

$$x_2 = \frac{1}{3}\left(-1 + 2x_3\right) = \frac{3}{3} = 1$$

$$x_1 = \frac{1}{-2}\left(9 - x_2 - 2x_3\right) = \frac{4}{-2} = -2$$

Solving Triangular Systems

Solving for $x_1, x_2, \ldots, x_n$ for a lower triangular system is called **forward substitution**.

```
given L, b
x_1 = b_1/ℓ_11
for i = 2...n
    s = b_i
    for j = 1...i−1
        s = s − ℓ_{i,j} x_j
    end
    x_i = s/ℓ_{i,i}
end
```

Solving Triangular Systems

Solving for $x_1, x_2, \ldots, x_n$ for a lower triangular system is called **forward substitution**.

```
given L, b
x_1 = b_1/ℓ_11
for i = 2...n
    s = b_i
    for j = 1...i-1
        s = s - ℓ_{i,j} x_j
    end
    x_i = s/ℓ_{i,i}
end
```

Using forward or backward substitution is sometimes referred to as performing a **triangular solve**.

Operations?

Try...

Sketch out an operation count to solve a triangular system of equations...

Operations?

Try...

Sketch out an operation count to solve a triangular system of equations...

cheap!

- begin in the bottom corner: 1 div
- row -2: 1 mult, 1 add, 1 div, or 3 FLOPS
- row -3: 2 mult, 2 add, 1 div, or 5 FLOPS
- row -4: 3 mult, 3 add, 1 div, or 7 FLOPS
- $\vdots$
- row -j: about $2j$ FLOPS

Total FLOPS? $\sum_{j=1}^{n} 2j = 2\frac{n(n+1)}{2}$ or $\mathcal{O}(n^2)$ FLOPS