

Lecture 05 - B

§2.1 Counting Operations and Solving Special Systems

T. Takahashi

Department of Mathematics and Statistics
University of New Mexico



We will answer these questions:

- How do you measure the cost of an algorithm?
- How and why do you benchmark code?
- How do you solve $Ax = b$ if A is diagonal?
- How do you solve $Ax = b$ if A is triangular? How expensive is it?
- How expensive is it to solve a banded system?



Counting FLOPs



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.

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

BLAS is highly optimized. Almost everywhere software runs some version of BLAS under the hood.



Definition: FLOP

- Adding, subtracting, multiplying, and dividing two floats is called a **floating point operation** (FLOP)
- The plural will be abbreviated as FLOPs
- FLOP = unit of computational cost
- Example: count cost to evaluate polynomial $p(t)$

$$\begin{aligned} p(t) &= 3t^2 - 4t + 1 \\ &= (3 \times t \times t) - (4 \times t) + 1 \end{aligned}$$

- 3 multiplications, 1 addition, 1 subtraction
- Evaluating $p(x)$ costs 5 FLOPs



Cost of Inner Product

Level 1

Example

How many FLOPs are required to compute the inner product

$$u^T v$$

of vectors of length n ? What is operation count in Big-O notation?

- The algorithm is

$$\begin{aligned} u^T v &= u_1 v_1 + \cdots + u_n v_n \\ &= (u_1 \times v_1) + \cdots + (u_n \times v_n) \end{aligned}$$

- n multiplies
- $n - 1$ additions
- There is exactly $2n - 1$ FLOPs
- Cost is $\mathcal{O}(n)$ FLOPs



Cost of Matrix-Vector Multiplication

Level 2

Example

How many FLOPs are required to compute the matrix-vector product

$$v = Au$$

in Big-O notation?

- The algorithm is

```
1 for i = 1,...,n
2     for j = 1,...,n
3         v(i) ← v(i) + a(i,j)*u(j)    % executed n^2 times
4     end
5 end
```

- n^2 multiplications and n^2 additions
- Cost is $\mathcal{O}(n^2)$ FLOPs



Cost of Matrix-Matrix Multiplication

Level 3

Example

How many FLOPs are required to compute the matrix-matrix product

$$C = AB$$

in Big-O notation? (Assume we are using standard algorithm)

- The algorithm is

```
1 for j = 1, ..., n
2   for i = 1, ..., n
3     for k = 1, ..., n
4       C(k, j) ← C(k, j) + A(k, i) * B(i, j) % n^3 times
5     end
6   end
7 end
```

- n^3 multiplications and n^3 additions
- Cost is $\mathcal{O}(n^3)$ FLOPs



The Order of Computation Matters

Example

Let $A, B \in \mathbb{R}^{n \times n}$ and $x \in \mathbb{R}^n$. Compare the cost of evaluating

$$(AB)x \quad \text{and} \quad A(Bx).$$

Assume dense matrices and standard multiplication.



The Order of Computation Matters

Example

Let $A, B \in \mathbb{R}^{n \times n}$ and $x \in \mathbb{R}^n$. Compare the cost of evaluating

$$(AB)x \quad \text{and} \quad A(Bx).$$

Assume dense matrices and standard multiplication.

- Compute $(AB)x$:
 - Form $C = AB$: $\mathcal{O}(n^3)$ FLOPs.
 - Compute Cx : $\mathcal{O}(n^2)$ FLOPs.
 - Total: $\boxed{\mathcal{O}(n^3)}$.



The Order of Computation Matters

Example

Let $A, B \in \mathbb{R}^{n \times n}$ and $x \in \mathbb{R}^n$. Compare the cost of evaluating

$$(AB)x \quad \text{and} \quad A(Bx).$$

Assume dense matrices and standard multiplication.

- Compute $(AB)x$:
 - Form $C = AB$: $\mathcal{O}(n^3)$ FLOPs.
 - Compute Cx : $\mathcal{O}(n^2)$ FLOPs.
 - Total: $\boxed{\mathcal{O}(n^3)}$.
- Compute $A(Bx)$:
 - Form $y = Bx$: $\mathcal{O}(n^2)$ FLOPs.
 - Compute Ay : $\mathcal{O}(n^2)$ FLOPs.
 - Total: $\boxed{\mathcal{O}(n^2)}$.



The Order of Computation Matters

Example

Let $A, B \in \mathbb{R}^{n \times n}$ and $x \in \mathbb{R}^n$. Compare the cost of evaluating

$$(AB)x \quad \text{and} \quad A(Bx).$$

Assume dense matrices and standard multiplication.

- Compute $(AB)x$:
 - Form $C = AB$: $\mathcal{O}(n^3)$ FLOPs.
 - Compute Cx : $\mathcal{O}(n^2)$ FLOPs.
 - Total: $\boxed{\mathcal{O}(n^3)}$.
- Compute $A(Bx)$:
 - Form $y = Bx$: $\mathcal{O}(n^2)$ FLOPs.
 - Compute Ay : $\mathcal{O}(n^2)$ FLOPs.
 - Total: $\boxed{\mathcal{O}(n^2)}$.

Key Idea

The expressions are equal in exact arithmetic, but their computational costs are very different.



Benchmarking Code



Why do we time our code?

Benchmarking code refers to calculating the wallclock runtime of your code from beginning to end.

Reasons:

- To identify bottlenecks in your program.
- To compare different algorithms or implementations.
- To predict how your algorithm scales for large input size.



Tic-Toc

- Matlab provides `tic` and `toc` commands to measure elapsed time.
- Basic usage:

```
1 tic
2 % Code to measure
3 toc
```

- Output: Elapsed time in seconds.

Example:

```
1 A = rand(1000,1000);
2 x = rand(1000,1);
3 tic
4 y = A*x;
5 elapsed_time = toc;
6 fprintf('Elapsed time: %f seconds\n', elapsed_time);
```



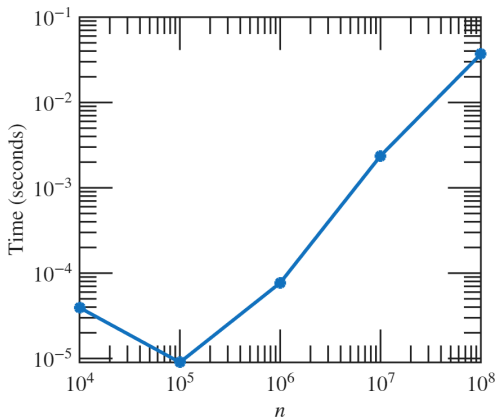
Timing $u'v$ in Matlab

```
1 n = 1000; % size of vectors
2 u = rand(n,1);
3 v = rand(n,1);
4
5 tic % begin clock
6
7 for j = 1 : 10 % do ten trials
8     ip = u'*v;
9 end
10
11 avg_time = toc/10; % average result
```

- The inner product $u'v$ is computed 10 times
- Divide total runtime by 10
- Repeat this for $n = 10^5, 10^6, 10^7, 10^8$



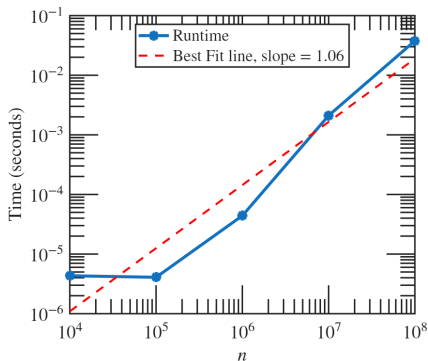
Result Over Ten Trials



- Plot runtime on log-log scale
- Runtime plots rarely look perfect due to **randomness**
- Ten trials is *not enough!*



Counting Ticks



- Calculate slope on log-log plot by counting tick marks
- Counting by orders of 10

$$\Delta x = 4$$

$$\Delta y = 4$$

- Slope $m = \frac{\Delta y}{\Delta x} = 1$ which means cost is $\mathcal{O}(n)$



Benchmarking Tips

Strongly Recommend

- Warm-up runs: Run your code a few times before measuring to ensure consistent performance.
- System noise: Close background processes that can interfere with timing.
- Multiple trials: Always average results over several runs.

Optional

- Vectorize your code: Matlab favors vectorized code over for-loops.
- Profiling: Use Matlab's `profile` command for detailed performance analysis. It is useful for spotting bottle necks.
- Alerts: You can program Matlab to play a sound or send an email when it is finished.

Solving Special Systems



Cost to Solve $Ax = b$

- Suppose $A \in \mathbb{R}^{n \times n}$
- How many FLOPs does it cost to solve

$$Ax = b?$$

- Let's begin by considering special matrices



Special Matrices

- **Diagonal**

$$\begin{bmatrix} \bullet & 0 & 0 & 0 \\ 0 & \bullet & 0 & 0 \\ 0 & 0 & \bullet & 0 \\ 0 & 0 & 0 & \bullet \end{bmatrix}$$

- **Triangular**

$$\begin{bmatrix} \bullet & \bullet & \bullet & \bullet \\ 0 & \bullet & \bullet & \bullet \\ 0 & 0 & \bullet & \bullet \\ 0 & 0 & 0 & \bullet \end{bmatrix}$$

- **Banded**

$$\begin{bmatrix} \bullet & \bullet & 0 & 0 \\ \bullet & \bullet & \bullet & 0 \\ 0 & \bullet & \bullet & \bullet \\ 0 & 0 & \bullet & \bullet \end{bmatrix}$$



Solving Diagonal Systems

Definition: Diagonal Matrix

A matrix A is **diagonal** if $a_{ij} = 0$ for all $i \neq j$.

Suppose you wish to solve $Ax = b$, where

$$A = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{bmatrix} \quad b = \begin{bmatrix} -1 \\ 6 \\ -15 \end{bmatrix}$$



Solving Diagonal Systems

Definition: Diagonal Matrix

A matrix A is **diagonal** if $a_{ij} = 0$ for all $i \neq j$.

Suppose you wish to solve $Ax = b$, where

$$A = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{bmatrix} \quad b = \begin{bmatrix} -1 \\ 6 \\ -15 \end{bmatrix}$$

Writing out the system

$$\begin{array}{rcl} 2x_1 & = & -1 \\ 3x_2 & = & 6 \\ 5x_3 & = & -15 \end{array}$$



Solving Diagonal Systems

Definition: Diagonal Matrix

A matrix A is **diagonal** if $a_{ij} = 0$ for all $i \neq j$.

Suppose you wish to solve $Ax = b$, where

$$A = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{bmatrix} \quad b = \begin{bmatrix} -1 \\ 6 \\ -15 \end{bmatrix}$$

Writing out the system

$$\begin{aligned} 2x_1 &= -1 \\ 3x_2 &= 6 \\ 5x_3 &= -15 \end{aligned}$$

The solution is

$$x_1 = -\frac{1}{2}$$

$$x_2 = \frac{6}{3} = 2$$

$$x_3 = \frac{-15}{5} = -3$$



Diagonal Solve

```
1 given  $A, b$   
2 for  $i = 1 \dots n$   
3    $x_i = b_i / a_{ii}$   
4 end
```

Listing 1: Diagonal System Solution

Method: Diagonal Solve

- **Given:** Diagonal matrix A , right hand side b
- **Output:** Solution to $Ax = b$
- **Idea:** Take advantage of the structure of A



Operation Count

Question

How expensive is it to solve $Ax = b$ if A is diagonal?

```
1 for  $i = 1 \dots n$   
2    $x_i = b_i / a_{i,i}$  % executed n times  
3 end
```

Key Idea

- Solving $Ax = b$ costs only $\mathcal{O}(n)$ FLOPs
- $\mathcal{O}(n)$ cost is considered to be *very fast*



Triangular System



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

Part 1

Example

Solve $Ax = b$ where

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

and count the total number of FLOPs.



Solving Triangular Systems

Part 1

Example

Solve $Ax = b$ where

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

and count the total number of FLOPs.

- First, write out the system

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



Solving Triangular Systems

Part 1

Example

Solve $Ax = b$ where

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

and count the total number of FLOPs.

- First, write out the system

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

- Solve for x_3 first.

$$x_3 = \frac{8}{4} = 2 \quad \text{one FLOP}$$



Solving Triangular Systems

Part 2

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

- We have x_3

$$x_3 = \frac{8}{4} = 2 \quad \text{one FLOP}$$



Solving Triangular Systems

Part 2

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

- We have x_3

$$x_3 = \frac{8}{4} = 2 \quad \text{one FLOP}$$

- Next, we solve for x_2

$$x_2 = \frac{-1 + 2 \times x_3}{3} = 1 \quad \text{three FLOPs}$$



Solving Triangular Systems

Part 2

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

- We have x_3

$$x_3 = \frac{8}{4} = 2 \quad \text{one FLOP}$$

- Next, we solve for x_2

$$x_2 = \frac{-1 + 2 \times x_3}{3} = 1 \quad \text{three FLOPs}$$

- Finally, we solve for x_1

$$x_1 = \frac{9 - 1 \times x_2 - 2 \times x_3}{-2} = -2 \quad \text{five FLOPs}$$



Solving Triangular Systems

Part 2

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

- We have x_3

$$x_3 = \frac{8}{4} = 2 \quad \text{one FLOP}$$

- Next, we solve for x_2

$$x_2 = \frac{-1 + 2 \times x_3}{3} = 1 \quad \text{three FLOPs}$$

- Finally, we solve for x_1

$$x_1 = \frac{9 - 1 \times x_2 - 2 \times x_3}{-2} = -2 \quad \text{five FLOPs}$$

- Total: $1+3+5 = 9$ FLOPs



Backward Substitution

This algorithm is called **backward substitution**

```
1 given  $U$  (upper  $\Delta$ ),  $b$ 
2  $x_n = b_n / u_{nn}$ 
3 for  $i = n - 1 \dots 1$  % loop backwards
4      $s = b_i$ 
5     for  $j = i + 1 \dots n$ 
6          $s = s - u_{i,j} x_j$ 
7     end
8      $x_i = s / u_{i,i}$ 
9 end
```

Method: Backward substitution

- **Given:** Upper triangular U , right-hand side b
- **Output:** Solution to $Ux = b$
- **Idea:** Solve backwards $x_n, x_{n-1}, \dots, x_1$ for max efficiency

Lower Triangular System

- Consider this example

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

- When written out, the simplest way to proceed becomes obvious

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

- Solve in forward order

$$\begin{aligned} x_1 &= \frac{8}{4} = 2 \\ x_2 &= \frac{1}{3}(-1 + 2x_3) = \frac{3}{3} = 1 \\ x_3 &= \frac{9 + 2x_1 - x_2}{2} = 6. \end{aligned}$$

- The idea is the same as **backward substitution** but in reverse



Forward Substitution

This algorithm is called **forward substitution**

```
1 given  $L$  (lower  $\triangle$ ),  $b$ 
2  $x_1 = b_1 / \ell_{11}$ 
3 for  $i = 2 \dots n$  % loop forward
4      $s = b_i$ 
5     for  $j = 1 \dots i - 1$ 
6          $s = s - \ell_{i,j} x_j$ 
7     end
8      $x_i = s / \ell_{i,i}$ 
9 end
```

Method: Forward substitution

- **Given:** Lower triangular L , right-hand side b
- **Output:** Solution to $Lx = b$
- **Idea:** Solve in forward sequence $x_1, \dots, x_n$ for max efficiency

Forward and backward substitution are **triangular solves**.



Operation-Counting Formula

Question

How expensive is to perform triangular solves for a matrix of size n ?

Here is a useful formula

Sum of the First n Integers

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

Proof: Let $S = 1 + 2 + \cdots + n$. Then

$$S = 1 + 2 + \cdots + n,$$

$$S = n + (n-1) + \cdots + 1.$$

Adding corresponding terms gives

$$2S = n(n+1), \quad S = \frac{n(n+1)}{2}.$$



Operation Count for $n \times n$

Suppose $A \in \mathbb{R}^{n \times n}$ is lower triangular. Forward substitution will cost:

- row 1: 1 div = 1 FLOP
- row 2: 1 mult, 1 add, 1 div = 3 FLOPs
- row 3: 2 mult, 2 add, 1 div = 5 FLOPs
- row 4: 3 mult, 3 add, 1 div = 7 FLOPs
- $\vdots$
- row j : $(j-1)$ multiplications, $(j-1)$ subtractions, and 1 division:

$$2(j-1) + 1 = 2j - 1 \text{ FLOPs.}$$

Total FLOPs

$$\sum_{j=1}^n (2j-1) = 2 \frac{n(n+1)}{2} - n = \boxed{n^2}.$$

Key Idea

Forward and backward substitution each cost $\mathcal{O}(n^2)$ FLOPs for a dense triangular system.



Banded Systems



Example: Upper Bidiagonal System

Part 1

Example

Solve the 4×4 **upper bidiagonal** system

$$B = \begin{bmatrix} 2 & 1 & 0 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 5 & 2 \\ 0 & 0 & 0 & 6 \end{bmatrix}, \quad b = \begin{bmatrix} 5 \\ 19 \\ 26 \\ 18 \end{bmatrix}.$$

For a general $n \times n$, what is the operation count?

We use back substitution:

$$x_4 = \frac{b_4}{B_{44}} = \frac{18}{6} = 3 \quad 1 \text{ FLOP}$$

$$x_3 = \frac{b_3 - B_{34}x_4}{B_{33}} = \frac{26 - 2 \cdot 3}{5} = \frac{20}{5} = 4 \quad 3 \text{ FLOPs}$$

$$x_2 = \frac{b_2 - B_{23}x_3}{B_{22}} = \frac{19 - 4 \cdot 4}{3} = \frac{19 - 16}{3} = 1 \quad 3 \text{ FLOPs}$$

$$x_1 = \frac{b_1 - B_{12}x_2}{B_{11}} = \frac{5 - 1 \cdot 1}{2} = \frac{4}{2} = 2 \quad 3 \text{ FLOPs}$$



Example: Upper Bidiagonal System

Part 2

- Each step of back substitution (except the first) uses:
 - 1 multiply
 - 1 subtract
 - 1 divide
- Total number of FLOPs = $1 + 3 + 3 + 3 = 10$
- For an $n \times n$ upper bidiagonal system,

$$\text{Total FLOPs} = 1 + 3(n - 1) = 3n - 2 = \mathcal{O}(n).$$

Key Idea

It is often easier to count operations by rows. For a bidiagonal, the cost to solve is only $\mathcal{O}(n)$ which is very fast!



Tridiagonal

Definition: Tridiagonal Matrix

A **tridiagonal** matrix has the form

$$\begin{bmatrix} d_1 & c_1 & & & & \\ a_1 & d_2 & c_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & a_{i-1} & d_i & c_i & \\ & & & \ddots & \ddots & \ddots \\ & & & & \ddots & \ddots & c_{n-1} \\ & & & & & a_{n-1} & d_n \end{bmatrix}.$$

The empty entries denote zeros.

- Many of the entries are zeros
- We can utilize *row operations* to simplify the system



Review: Elementary Row Operations

The following operations preserve the solution set of $Ax = b$:

- Swap two rows:

$$R_i \leftrightarrow R_j.$$

- Multiply a row by a nonzero scalar:

$$R_i \leftarrow cR_i, \quad c \neq 0.$$

- Add a multiple of another row:

$$R_i \leftarrow R_i + cR_j, \quad i \neq j.$$



Tridiagonal Elimination

Example

Example

Use elimination to turn this system into an upper bidiagonal system:

$$\left[\begin{array}{ccc|c} 2 & 1 & 0 & 4 \\ 4 & 5 & 1 & 17 \\ 0 & 6 & 4 & 24 \end{array} \right].$$



Tridiagonal Elimination

Example

Example

Use elimination to turn this system into an upper bidiagonal system:

$$\left[\begin{array}{ccc|c} 2 & 1 & 0 & 4 \\ 4 & 5 & 1 & 17 \\ 0 & 6 & 4 & 24 \end{array} \right].$$

- Apply $R_2 \leftarrow R_2 - 2R_1$:

$$\left[\begin{array}{ccc|c} 2 & 1 & 0 & 4 \\ 0 & 3 & 1 & 9 \\ 0 & 6 & 4 & 24 \end{array} \right].$$



Tridiagonal Elimination

Example

Example

Use elimination to turn this system into an upper bidiagonal system:

$$\left[\begin{array}{ccc|c} 2 & 1 & 0 & 4 \\ 4 & 5 & 1 & 17 \\ 0 & 6 & 4 & 24 \end{array} \right].$$

- Apply $R_2 \leftarrow R_2 - 2R_1$:

$$\left[\begin{array}{ccc|c} 2 & 1 & 0 & 4 \\ 0 & 3 & 1 & 9 \\ 0 & 6 & 4 & 24 \end{array} \right].$$

- Apply $R_3 \leftarrow R_3 - 2R_2$:

$$\left[\begin{array}{ccc|c} 2 & 1 & 0 & 4 \\ 0 & 3 & 1 & 9 \\ 0 & 0 & 2 & 6 \end{array} \right].$$



Tridiagonal Elimination

Example

Example

Use elimination to turn this system into an upper bidiagonal system:

$$\left[\begin{array}{ccc|c} 2 & 1 & 0 & 4 \\ 4 & 5 & 1 & 17 \\ 0 & 6 & 4 & 24 \end{array} \right].$$

- Apply $R_2 \leftarrow R_2 - 2R_1$:

$$\left[\begin{array}{ccc|c} 2 & 1 & 0 & 4 \\ 0 & 3 & 1 & 9 \\ 0 & 6 & 4 & 24 \end{array} \right].$$

- Apply $R_3 \leftarrow R_3 - 2R_2$:

$$\left[\begin{array}{ccc|c} 2 & 1 & 0 & 4 \\ 0 & 3 & 1 & 9 \\ 0 & 0 & 2 & 6 \end{array} \right].$$

- The resulting system is upper bidiagonal and can be solved by backward substitution.



Tridiagonal

$$\begin{bmatrix} d_1 & c_1 & & & & \\ a_1 & d_2 & c_2 & & & \\ & a_2 & d_3 & c_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{i-1} & d_i & c_i \\ & & & & \ddots & \ddots & \ddots \\ & & & & & \ddots & \ddots & c_{n-1} \\ & & & & & & a_{n-1} & d_n \end{bmatrix}$$

Start forward elimination (without pivoting)

- 1 subtract a_1/d_1 times row 1 from row 2
- 2 this eliminates a_1 , changes d_2 and does not touch c_2
- 3 continuing:

$$d_i = d_i - \left(\frac{a_{i-1}}{d_{i-1}} c_{i-1} \right)$$

for $i = 2 \dots n$



Tridiagonal

$$\begin{bmatrix} d_1 & c_1 & & & & \\ & \tilde{d}_2 & c_2 & & & \\ & & \tilde{d}_3 & c_3 & & \\ & & & \ddots & \ddots & \\ & & & & \tilde{d}_i & c_i \\ & & & & & \ddots & \ddots \\ & & & & & & \ddots & c_{n-1} \\ & & & & & & & \tilde{d}_n \end{bmatrix}$$

This leaves an upper triangular (2-band). With back substitution:

- 1 $x_n = \tilde{b}_n / \tilde{d}_n$
- 2 $x_i = (1/\tilde{d}_i)(\tilde{b}_i - c_i x_{i+1})$ for $i = n - 1$ to 1



Tridiagonal Algorithm

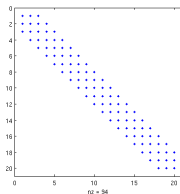
```
1 input:  $n, a, d, c, b$ 
2 for  $i = 2$  to  $n$ 
3      $xmult = a_{i-1}/d_{i-1}$ 
4      $d_i = d_i - xmult \cdot c_{i-1}$ 
5      $b_i = b_i - xmult \cdot b_{i-1}$ 
6 end
7  $x_n = b_n/d_n$ 
8 for  $i = n-1$  down to 1
9      $x_i = (b_i - c_i x_{i+1})/d_i$ 
10 end
```

Listing 2: Thomas Algorithm

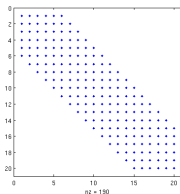
The algorithm to solve a tridiagonal system is also $\mathcal{O}(n)$ - very fast!



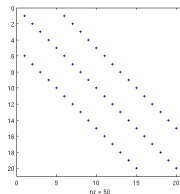
m -band Cost



$m = 5$



$m = 11$



$m = 11$

Cost for banded systems

- If A has bandwidth m , then solving $Ax = b$ cost $\mathcal{O}(m^2n)$ FLOPs.
- This information is not in your textbook.



Summary of Costs

Operations

- inner product $u^T v$: $\mathcal{O}(n)$
- matrix-vector product Ax : $\mathcal{O}(n^2)$
- matrix-matrix product AB : $\mathcal{O}(n^3)$

Solvers

- diagonal solve: $\mathcal{O}(n)$
- tridiagonal solve: $\mathcal{O}(n)$
- banded solve: $\mathcal{O}(m^2 n)$ (usually $m \ll n$)
- triangular solve: $\mathcal{O}(n^2)$

