\[ f(x) = \frac{1}{4}x + \frac{1}{2} \]
• Recall the idea of LU factorization. Write $A$ as

$$A = LU$$  \hspace{1cm} (1)$$

where

$L$ is unit lower triangular, i.e.,

$$l_{ii} = 1, \quad i = 1, 2, \ldots n \quad \text{and} \quad j > i \quad \Rightarrow \quad a_{ij} = 0$$

$U$ is upper triangular, i.e.,

$$l_{ii} = 1, \quad i = 1, 2, \ldots n \quad \text{and} \quad j < i \quad \Rightarrow \quad a_{ij} = 0$$

• Given the matrix factorization (1) we solve the system $Ax = LUx = b$ in two steps:

$$Ax = L(Ux) = Lz = b$$

1. solve: $Lz = b$ \hspace{1cm} Forward Substitution
2. solve: $Ux = z$ \hspace{1cm} Backward Substitution
Also recall the idea of Gaussian Elimination:

For \( k = 1, \ldots, n - 1 \)

For \( i = k + 1, \ldots, n \)

\[
a_{ik} = \frac{a_{ik}}{a_{kk}}
\]

For \( j = k + 1, \ldots, n \)

\[
a_{ij} = a_{ij} - a_{ik}a_{kj}
\]

Actually, Gaussian Elimination computes the \( LU \) factorization

- the two procedures are equivalent!

- The standard argument to show this is a mess that involves “elementary matrices” and their inverses, and generous use of groups of three dots . . . .

- However, Gil Strang of MIT came up with a beautifully simple argument.
Consider the evolution of the working array during Gaussian Elimination, for a $4 \times 4$ system.

The letter $x$ denotes an entry in the working array. The symbol $\bigotimes$ denotes an entry that is final and no longer changed in the process. The letter $m$ denotes the multipliers, stored in the lower part of the working array.

We get

\[
\begin{bmatrix}
\bigotimes & \bigotimes & \bigotimes & \bigotimes \\
x & x & x & x \\
x & x & x & x \\
x & x & x & x \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
\bigotimes & \bigotimes & \bigotimes & \bigotimes \\
m_{21} & \bigotimes & \bigotimes & \bigotimes \\
m_{31} & x & x & x \\
m_{41} & x & x & x \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
\bigotimes & \bigotimes & \bigotimes & \bigotimes \\
m_{21} & \bigotimes & \bigotimes & \bigotimes \\
m_{31} & m_{32} & \bigotimes & \bigotimes \\
m_{41} & m_{42} & x & x \\
\end{bmatrix}
\rightarrow
\begin{bmatrix}
\bigotimes & \bigotimes & \bigotimes & \bigotimes \\
m_{21} & \bigotimes & \bigotimes & \bigotimes \\
m_{31} & m_{32} & \bigotimes & \bigotimes \\
m_{41} & m_{42} & m_{43} & \bigotimes \\
\end{bmatrix}
\]

Let’s denote row $i$ of a matrix $A$ by $r_i(A)$ and consider in particular the third row of the working array. We get

\[
r_3(U) = r_3(A) - m_{31}r_1(U) - m_{32}r_2(U)
\]

This can be rewritten as
\[ r_3(A) = m_{31}r_1(U) + m_{32}r_2(U) + r_3(U) \]

That last equation is exactly what we get in the matrix multiplication \( A = LU \!\)!

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
m_{21} & 1 & 0 & 0 \\
m_{31} & m_{32} & 1 & 0 \\
m_{41} & m_{42} & m_{43} & 1
\end{bmatrix}
\begin{bmatrix}
r_1(U) \\
r_2(U) \\
r_3(U) \\
r_4(U)
\end{bmatrix}
= U
\]

\[
\begin{bmatrix}
r_1(A) \\
r_2(A) \\
r_3(A) \\
r_4(A)
\end{bmatrix}
= A
\]

\[
= L
\]

- Clearly, this applies in general, not just to the third row of a \( 4 \times 4 \) matrix!

- \( A = LU \), \( L \) is unit lower triangular, \( U \) is upper triangular.
Pivoting

- We understand pivoting from a computational point of view, but we also want to describe it in terms of matrices.

- A permutation matrix is a square matrix that is all zero except that it has one entry equal to 1 in each row and each column.

- Another view is that a permutation matrix is obtained by permuting the rows or the columns of the identity matrix.

- Multiplying a matrix $A$ with a permutation matrix from the left permutes the rows of $A$ and multiplying from the right permutes the columns of $A$.

- Example: Let

\[
A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \text{and} \quad P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}
\]

Then:

\[
P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = P
\]

\[
A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = AP
\]

\[
P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = PA
\]
• Note that a permutation matrix is orthogonal, i.e.,
\[ P^{-1} = P^T \]

• That’s clear because a permutation matrix contains a bunch of orthonormal vectors in a certain sequence and \( P^T P \) is the matrix of dot products of those orthonormal vectors!

• In the process of Gaussian Elimination with partial pivoting we permute the rows of \( A \). Thus we really factor \( P^T A \) (for a suitable \( P \)). We get
\[ P^T A = LU \]
or
\[ A = PLU. \]

• If we do total pivoting we factor
\[ A = PLUQ \]
where \( P \) and \( Q \) are suitable permutation matrices. Specifically, they are the inverses of the matrices that define the row and column interchanges, respectively.

• As a practical matter we store a permutation matrix in an integer vector. For example, the \( i \)-th entry of that vector could be the column index of the 1 in the \( i \)-row.

• Note that if \( A \) is singular then we will not be able to find a pivot and the process breaks down.
The Inverse Matrix

- If we knew the inverse matrix we could solve the linear system by multiplying the right hand side with the inverse: \( \mathbf{x} = \mathbf{A}^{-1} \mathbf{A} \mathbf{x} = \mathbf{A}^{-1} \mathbf{b} \)

\[
Ax = b \iff x = A^{-1}b
\]

- The inverse is defined by

\[
\mathbf{A}^{-1} \mathbf{A} = \mathbf{A} \mathbf{A}^{-1} = \mathbf{I}
\]

- Thus the \( j \)-th column of the inverse is the solution of

\[
Ax = e_j = \begin{bmatrix}
0 \\
\vdots \\
0 \\
1 \\
0 \\
\vdots \\
0
\end{bmatrix}
\]

- So computing the inverse amounts to solving \( n \) linear systems all of which have the coefficient matrix \( A \).
Measuring Numerical Effort

- “numerical effort” could mean the time a computer needs to execute an algorithm.

- That time depends on the computer (and also the particular implementation and the computer language being used).

- We want to compare algorithms, independently of computer specifics.

- The standard way to compare linear algebra algorithms is to count the number of flops which usually means the number of multiplications and divisions.

- The reason not to count additions and subtractions, or other operations like assignments and data lookups, is that in typical implementations these operations require and amount of time that is roughly proportional to the number of flops.

- Revisit the Gaussian Elimination Algorithm:
For $k = 1, \ldots, n - 1$

For $i = k + 1, \ldots, n$

$$a_{ik} = \frac{a_{ik}}{a_{kk}}$$

For $j = k + 1, \ldots, n$

$$a_{ij} = a_{ij} - a_{ik}a_{kj}$$

• It's a simple exercise to verify that

$$\#\text{flops} = \sum_{k=1}^{n-1} \left( \sum_{i=k+1}^{n} \left( 1 + \sum_{j=k+1}^{n} 1 \right) \right)$$

$$= \frac{n^3}{3} - \frac{n}{3}$$

$$= \frac{n^3}{3} + \mathcal{O}(n^2) = \mathcal{O}(n^3)$$

• The analysis is dependent on formulas like

$$\sum_{i=1}^{n} i = \frac{n(n + 1)}{2} \quad \text{and} \quad \sum_{i=1}^{n} i^2 = \frac{n(n + 1)(2n + 1)}{6}.$$
• However, since only the leading term matters it is often convenient to approximate the sums by integrals:

\[
\#\text{flops} \approx \int_1^{n-1} \int_{k+1}^n \left( 1 + \int_{k+1}^n dj \right) dik
\]

\[
= \frac{n^3}{3} - \frac{3n^2}{2} + 2n - \frac{2}{3}
\]

\[
= \frac{n^3}{3} + O(n^2).
\]

• The following Table shows the number of flops required for some common operations:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Flops</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A = LU)</td>
<td>(\frac{n^3}{3} + O(n^2))</td>
</tr>
<tr>
<td>(Ly = b)</td>
<td>(\frac{n^2}{2} - \frac{n}{2})</td>
</tr>
<tr>
<td>(Ux = y)</td>
<td>(\frac{n^2}{2} + \frac{n}{2})</td>
</tr>
<tr>
<td>(x = A^{-1}b)</td>
<td>(n^2)</td>
</tr>
</tbody>
</table>

\(2\) note that forward and backward substitution together take exactly the same effort as multiplying with the inverse!

• However, computing the inverse is three times as expensive as computing the \(LU\) factorization.
Moreover, as the first step in computing $A^{-1}$, you compute $LU$ anyway. Thus computing $A^{-1}$ after computing the $LU$ factorization just introduces additional round-off errors.

However, the biggest problem with matrix inversion is that it destroys sparsity. For example (see problem 10 of hw 2) the inverse of a tridiagonal matrix in general is full, whereas $L$ and $U$ in the LU factorization are still tridiagonal.

In general, an $n \times n$ tridiagonal system can be solved in $5n$ flops.

Simplifying things somewhat we arrive at the conclusion

Never Invert a Matrix

Can you get below $O(n^3)$? Yes, but it’s tricky, and beyond our scope.
• bet with Nick Trefethen:

25 June 1985

C. N. Trefethen hereby bets Peter Arfeld
that by 31 December, 1994, a method
will have been found to solve $Ax = b$
(non linear system of eqns.) in $O(n^{2+\varepsilon})$
operations for any $\varepsilon > 0$. Numerical stability
is not required.
The winner gets $100. from

the loser

Peter Arfeld
Lloyd N. Trefethen

Witnesses:
Per Erik Koch
S. P. Nørsett (This is a subtle problem)

Figure 1. A Bet.
\[ A\hat{x} = b \implies \hat{x} = x \]

\[ e = x - \hat{x} = \varepsilon \]

\[ b - A\hat{x} = r \text{ residual} \]

\[ e \leftrightarrow r \]