# Applied Numerical Linear Algebra. Lecture 4

# Pivot element

- The pivot or pivot element is the element of a matrix, an array, or some other kind of finite set, which is selected first by an algorithm (e.g. Gaussian elimination, Quicksort, Simplex algorithm, etc.), to do certain calculations. In the case of matrix algorithms, a pivot entry is usually required to be at least distinct from zero, and often distant from it; in this case finding this element is called pivoting.
- Pivoting may be followed by an interchange of rows or columns to bring the pivot to a fixed position and allow the algorithm to proceed successfully, and possibly to reduce round-off error.

# Examples of systems that require pivoting

In the case of Gaussian elimination, the algorithm requires that pivot elements not be zero. Interchanging rows or columns in the case of a zero pivot element is necessary. The system below requires the interchange of rows 2 and 3 to perform elimination.

$$\left[\begin{array}{ccc|c} 1 & -1 & 2 & 8 \\ 0 & 0 & -1 & -11 \\ 0 & 2 & -1 & -3 \end{array}\right]$$

The system that results from pivoting is as follows and will allow the elimination algorithm and backwards substitution to output the solution to the system.

$$\left[\begin{array}{ccc|c} 1 & -1 & 2 & 8 \\ 0 & 2 & -1 & -3 \\ 0 & 0 & -1 & -11 \end{array}\right]$$

# Examples of systems that require pivoting

Furthermore, in Gaussian elimination it is generally desirable to choose a pivot element with large absolute value. This improves the numerical stability. The following system is dramatically affected by round-off error when Gaussian elimination and backwards substitution are performed.

$$\left[\begin{array}{cc|c} 0.00300 & 59.14 & 59.17 \\ 5.291 & -6.130 & 46.78 \end{array}\right]$$

This system has the exact solution of $x_1 = 10.00$ and $x_2 = 1.000$, but when the elimination algorithm and backwards substitution are performed using four-digit arithmetic, the small value of $a_{11}$ causes small round-off errors to be propagated. The algorithm without pivoting yields the approximation of $x_1 \approx 9873.3$ and $x_2 \approx 4$.

# Examples of systems that require pivoting

In this case it is desirable that we interchange the two rows so that $a_{21}$ is in the pivot position

$$\left[\begin{array}{cc|c} 5.291 & -6.130 & 46.78 \\ 0.00300 & 59.14 & 59.17 \end{array}\right].$$

Considering this system, the elimination algorithm and backwards substitution using four-digit arithmetic yield the correct values $x1 = 10.00$ and $x2 = 1.000$.

# Partial and complete pivoting

In partial pivoting, the algorithm selects the entry with largest absolute value from the column of the matrix that is currently being considered as the pivot element. Partial pivoting is generally sufficient to adequately reduce round-off error. However for certain systems and algorithms, complete pivoting (or maximal pivoting) may be required for acceptable accuracy. Complete pivoting considers all entries in the whole matrix, interchanging rows and columns to achieve the highest accuracy. Complete pivoting is usually not necessary to ensure numerical stability and, due to the additional computations it introduces, it may not always be the most appropriate pivoting strategy.

# Gaussian elimination

- In linear algebra, Gaussian elimination is an algorithm for solving systems of linear equations. It can also be used to find the rank of a matrix, to calculate the determinant of a matrix, and to calculate the inverse of an invertible square matrix. The method is named after Carl Friedrich Gauss, but it was not invented by him.
- Elementary row operations are used to reduce a matrix to what is called triangular form (in numerical analysis) or row echelon form (in abstract algebra). Gauss-Jordan elimination, an extension of this algorithm, reduces the matrix further to diagonal form, which is also known as reduced row echelon form.

# Gaussian elimination. History

- The method of Gaussian elimination appears in the important Chinese mathematical text Chapter Eight Rectangular Arrays of The Nine Chapters on the Mathematical Art. Its use is illustrated in eighteen problems, with two to five equations. The first reference to the book by this title is dated to 179 CE, but parts of it were written as early as approximately 150 BCE. It was commented on by Liu Hui in the 3rd century.
- The method in Europe stems from the notes of Isaac Newton. In 1670, he wrote that all the algebra books known to him lacked a lesson for solving simultaneous equations, which Newton then supplied. Cambridge University eventually published the notes as Arithmetica Universalis in 1707 long after Newton left academic life.
- Carl Friedrich Gauss in 1810 devised a notation for symmetric elimination that was adopted in the 19th century by professional hand computers to solve the normal equations of least-squares problems. The algorithm that is taught in high school was named for Gauss only in the 1950s as a result of confusion over the history

# Gaussian elimination. Algorithm overview

- The process of Gaussian elimination has two parts. The first part (Forward Elimination) reduces a given system to either triangular or echelon form, or results in a degenerate equation, indicating the system has no unique solution but may have multiple solutions (rank¡order). This is accomplished through the use of elementary row operations. The second step uses back substitution to find the solution of the system above.
- Stated equivalently for matrices, the first part reduces a matrix to row echelon form using elementary row operations while the second reduces it to reduced row echelon form, or row canonical form.
- The three elementary row operations used in the Gaussian elimination (multiplying rows, switching rows, and adding multiples of rows to other rows) amount to multiplying the original matrix with invertible matrices from the left. The first part of the algorithm computes an $LU$ decomposition, while the second part writes the original matrix as the product of a uniquely determined invertible matrix and a uniquely determined reduced row-echelon matrix.

# Gaussian elimination. Example

Suppose the goal is to find and describe the solution(s), if any, of the following system of linear equations:

$$\begin{aligned} 2x &+ y - z &= 8 \quad (L_1)\\ -3x &- y + 2z &= -11 \quad (L_2)\\ -2x &+ y + 2z &= -3 \quad (L_3) \end{aligned}$$

The algorithm is as follows: eliminate $x$ from all equations below $L_1$, and then eliminate $y$ from all equations below $L_2$. This will put the system into triangular form. Then, using back-substitution, each unknown can be solved.

In the example, $x$ is eliminated from $L_2$ by adding $\frac{3}{2}L_1$ to $L_2$. $x$ is then eliminated from $L_3$ by adding $L_1$ to $L_3$. Formally:

$$L_2 + \frac{3}{2}L_1 \to L_2; L_3 + L_1 \to L_3$$

The result is:

$$\begin{aligned} 2x &+ y - z &= 8\\ \frac{1}{2}y &+ \frac{1}{2}z &= 1 \end{aligned}$$

# Gaussian elimination. Example

Now y is eliminated from $L_3$ by adding $-4L_2$ to $L_3$:

$$L_3 + -4L_2 \to L_3$$

The result is:

$$\begin{array}{rcrcrcl} 2x & + & y & - & z & = & 8 \\ & & \frac{1}{2}y & + & \frac{1}{2}z & = & 1 \\ & & & & -z & = & 1 \end{array}$$

This result is a system of linear equations in triangular form, and so the first part of the algorithm is complete.
The last part, back-substitution, consists of solving for the knowns in reverse order. It can thus be seen that

$$z = -1 \quad (L_3)$$

Then, $z$ can be substituted into $L_2$, which can then be solved to obtain

$$y = 3 \quad (L_2)$$

Next, $z$ and $y$ can be substituted into $L_1$, which can be solved to obtain

# Gaussian elimination

The basic algorithm for solving $Ax = b$.

1. Permutation matrices.
2. The algorithm - overview.
3. The algorithm - factorization with pivoting.
4. The algorithm - optimizations and complexity analysis.

# Permutation matrices

## Definition

Permutation matrix := identity matrix with permuted rows.

## Example

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

# Permutation matrices

## Definition

Permutation matrix := identity matrix with permuted rows.

## Example

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix} \quad \rightarrow \quad \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{matrix} \quad \rightarrow$$

# Permutation matrices

## Definition

Permutation matrix := identity matrix with permuted rows.

## Example

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix} \rightarrow \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{matrix} \rightarrow \begin{matrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{matrix}$$

# Permutation matrices

## Properties

Properties of permutation matrices ($P$, $P_1$, $P_2$):

- $P \cdot X =$ same matrix X with rows permuted
- $P_1 \cdot P_2$ is also a permutation
- $P^{-1} = P^T$ (reverse permutation)
- $\det(P) = \pm 1$ (+1 for even permutations, -1 for odd)

# Gaussian Elimination

## The Algorithm — Overview

Solving $Ax = b$ using Gaussian elimination.

1. Factorize $A$ into $A = PLU$

   Permutation  Unit lower triangular  Non-singular upper triangular

# Gaussian Elimination

## The Algorithm — Overview

Solving $Ax = b$ using Gaussian elimination.

1. Factorize $A$ into $A = PLU$
   Permutation  Unit lower triangular  Non-singular upper triangular
2. Solve $PLUx = b$ (for $LUx$) :

$$LUx = P^{-1}b$$

# Gaussian Elimination

## The Algorithm — Overview

Solving $Ax = b$ using Gaussian elimination.

1. Factorize $A$ into $A = PLU$
   Permutation  Unit lower triangular  Non-singular upper triangular
2. Solve $PLUx = b$ (for $LUx$) :

$$LUx = P^{-1}b$$

3. Solve $LUx = P^{-1}b$ (for $Ux$) by forward substitution:

$$Ux = L^{-1}(P^{-1}b).$$

# Gaussian Elimination

## The Algorithm — Overview

Solving $Ax = b$ using Gaussian elimination.

1. Factorize $A$ into $A = PLU$
   Permutation  Unit lower triangular  Non-singular upper triangular
2. Solve $PLUx = b$ (for $LUx$) :

$$LUx = P^{-1}b$$

3. Solve $LUx = P^{-1}b$ (for $Ux$) by forward substitution:

$$Ux = L^{-1}(P^{-1}b).$$

4. Solve $Ux = L^{-1}(P^{-1}b)$ by backward substitution:

$$x = U^{-1}(L^{-1}P^{-1}b).$$

# The Algorithm — uniqueness of factorization

**Definition**

The leading j-by-j principal submatrix of $A$ is $A(1:j, 1:j)$.

**Theorem**

*The following two statements are equivalent:*
*1. There exists a unique unit lower triangular $L$ and non-singular upper triangular $U$ such that $A = LU$.*
*2. All leading principal submatrices of $A$ are non-singular.*

# The Algorithm — uniqueness of factorization

**Proof.**

We first show that (1) implies (2). $A = LU$ may also be written

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \times \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} = \begin{bmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{bmatrix}$$

where $A_{11}$ is a j-by-j leading principal submatrix, as well as $L_{11}$ and $U_{11}$. Therefore
$detA_{11} = det(L_{11}U_{11}) = detL_{11}detU_{11} = 1 \cdot detU_{11} \neq 0$, since $L$ is unit triangular and $U$ is non-singular.

# The Algorithm — uniqueness of factorization

**Proof.**

(2) implies (1) is proved by induction on n. It is easy for 1-by-1 matrices: $a = 1 \cdot a$. To prove it for n-by-n matrices $\tilde{A}$, we need to find unique (n-1)-by-(n-1) triangular matrices $L$ and $U$, unique (n-1)-by-1 vectors $l$ and $u$, and unique nonzero scalar $\eta$ such that

$$\tilde{A} = \begin{bmatrix} A & b \\ c^T & \delta \end{bmatrix} = \begin{bmatrix} L & 0 \\ l^T & 1 \end{bmatrix} \times \begin{bmatrix} U & u \\ 0 & \eta \end{bmatrix} = \begin{bmatrix} LU & Lu \\ l^T U & l^T u + \eta \end{bmatrix}$$

By induction unique $L$ and $U$ exist such that $A = LU$. Now let $u = L^{-1}b$, $l^T = c^T U^{-1}$, and $\eta = \delta - l^T u$, all of which are unique. The diagonal entries of $U$ are nonzero by induction, and $\eta \neq 0$ since

$$det\tilde{A} = det \begin{bmatrix} L & 0 \\ l^T & 1 \end{bmatrix} \times det \begin{bmatrix} U & u \\ 0 & \eta \end{bmatrix}$$

$$0 \neq det\tilde{A} = det\tilde{L} \cdot det\tilde{U} = det(L) \cdot det(U) \cdot \eta = det(U) \cdot \eta.$$

□

# The Algorithm — factorization with pivoting

**Theorem**

*If $A$ is non-singular, then there exist permutations $P_1$ and $P_2$, a unit lower triangular matrix $L$, and a non-singular upper triangular matrix $U$ such that $P_1AP_2 = LU$. Only one of $P_1$ and $P_2$ is necessary.*

**Proof.**

As with many matrix factorizations, it suffices to understand block 2-by-2 matrices. More formally, we use induction on dimension n. It is easy for 1-by-1 matrices: $P_1 = P_2 = L = 1$ and $U = A$. Assume that it is true for dimension n-1. If $A$ is non-singular, then it has a nonzero entry since non-singularity implies that each row and each column of $A$ has a nonzero entry.

Let us choose permutations $P_1'$ and $P_2'$ such that the entry $(1, 1)$ of the matrix $P_1'AP_2'$ is nonzero. We define it by $a_{11}$.

# The Algorithm — factorization with pivoting

## Proof.

Now we write the desired factorization and solve for the unknown components:

$$P_1' A P_2' = \begin{bmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ L_{21} & I \end{bmatrix} \cdot \begin{bmatrix} u_{11} & U_{12} \\ 0 & \tilde{A}_{22} \end{bmatrix} = \begin{bmatrix} u_{11} & U_{12} \\ L_{21} u_{11} & L_{21} U_{12} + \tilde{A}_{22} \end{bmatrix},$$

where $A_{22}$ and $\tilde{A}_{22}$ are (n-1)-by-(n-1) and $L_{21}$ and $U_{12}^T$ are (n-1)-by-1.

Solving for the components of this 2-by-2 block factorization we get $u_{11} = a_{11} \neq 0$, $U_{12} = A_{12}$, and $L_{21} u_{11} = A_{21}$. Since $u_{11} = a_{11} \neq 0$, we can solve for $L_{21} = \frac{A_{21}}{a_{11}}$. Finally, $L_{21} U_{12} + \tilde{A}_{22} = A_{22}$ implies $\tilde{A}_{22} = A_{22} - L_{21} U_{12}$ what means that we can compute elements of $\tilde{A}_{22}$.

# The Algorithm — factorization with pivoting

**Proof.**

We are going to apply induction to $\tilde{A}_{22}$, but to do so we need to check that $det\tilde{A}_{22} \neq 0$: Since $detP_1'AP_2' = \pm detA \neq 0$ and also $u_{11} \neq 0$ we have

$$0 \neq detP_1'AP_2' = det \begin{bmatrix} 1 & 0 \\ L_{21} & I \end{bmatrix} \cdot det \begin{bmatrix} u_{11} & U_{12} \\ 0 & \tilde{A}_{22} \end{bmatrix} = 1 \cdot (u_{11} \cdot det\tilde{A}_{22}),$$

then $det\tilde{A}_{22}$ must be nonzero.
Therefore by induction there exist permutations $\tilde{P}_1$, $\tilde{P}_2$ so that $\tilde{P}_1\tilde{A}_{22}\tilde{P}_2 = \tilde{L}\tilde{U}$ with $\tilde{L}$ unit lower triangular and $\tilde{U}$ upper triangular and non-singular.

# The Algorithm — factorization with pivoting

## Proof.

Substituting this in the above 2-by-2 block factorization yields

$$P_1' A P_2' = \begin{bmatrix} 1 & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} u_{11} & U_{12} \\ 0 & \tilde{P}_1^T \tilde{L} \tilde{U} \tilde{P}_2^T \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \tilde{P}_1^T \tilde{L} \end{bmatrix} \begin{bmatrix} u_{11} & U_{12} \\ 0 & \tilde{U} \tilde{P}_2^T \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 \\ L_{21} & \tilde{P}_1^T \tilde{L} \end{bmatrix} \begin{bmatrix} u_{11} & U_{12} \tilde{P}_2 \\ 0 & \tilde{U} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \tilde{P}_2^T \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & \tilde{P}_1^T \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \tilde{P}_1 L_{21} & \tilde{L} \end{bmatrix} \begin{bmatrix} u_{11} & U_{12} \tilde{P}_2 \\ 0 & \tilde{U} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \tilde{P}_2^T \end{bmatrix}$$

## Proof.

Recall that we have obtained:

$$P_1' A P_2' = \begin{bmatrix} 1 & 0 \\ 0 & \tilde{P}_1^T \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \tilde{P}_1 L_{21} & \tilde{L} \end{bmatrix} \begin{bmatrix} u_{11} & U_{12}\tilde{P}_2 \\ 0 & \tilde{U} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \tilde{P}_2^T \end{bmatrix}$$

Now multiplying both sides of the above equation by

$$\begin{bmatrix} 1 & 0 \\ 0 & \tilde{P}_1 \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 \\ 0 & \tilde{P}_2 \end{bmatrix}$$

from the left and right hand side, respectively, we get:

$$P_1 A P_2 = \underbrace{\left( \begin{bmatrix} 1 & 0 \\ 0 & \tilde{P}_1 \end{bmatrix} P_1' \right)}_{P_1} A \underbrace{\left( P_2' \begin{bmatrix} 1 & 0 \\ 0 & \tilde{P}_2 \end{bmatrix} \right)}_{P_2}$$

$$= \underbrace{\begin{bmatrix} 1 & 0 \\ \tilde{P}_1 L_{21} & \tilde{L} \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} u_{11} & U_{12}\tilde{P}_2 \\ 0 & \tilde{U} \end{bmatrix}}_{U}$$

# Corollary about factorization with pivoting

We can choose $P_2' = I$ and $P_1'$ such that $a_{11}$ is the largest entry in absolute value in its column. This is: $L_{21} = \frac{A_{21}}{a_{11}}$ has entries bounded by 1 in absolute value. More generally, at step i of Gaussian elimination, where we are computing the i-th column of $L$, we reorder rows $i$ through $n$ so that the largest entry in the column is on the diagonal. This is called "Gaussian elimination with partial pivoting," or GEPP for short. GEPP guarantees that all entries of $L$ are bounded by one in absolute value.

# Corollary about factorization with pivoting

We can choose $P_1'$ and $P_2'$ so that $a_{11}$ is the largest entry in absolute value in the whole matrix. More generally, at step i of Gaussian elimination, where we are computing the ith column of L, we reorder rows and columns i through n so that the largest entry in this submatrix is on the diagonal. This is called "Gaussian elimination with complete pivoting," or GECP for short.

# LU factorization

## Algorithm A

*LU factorization with pivoting:*

*for* $i = 1$ *to* $n$

*apply permutations so* $a_{ii} \neq 0$*(permute L, U)*

*/* for example for GEPP, swap rows j and i of A and of L where* $|a_{ji}|$ *is the largest entry in* $|A(i:n, i|$*;*
*for GECP, swap rows j and i of A and of L, and columns k and i of A and U, where* $|a_{jk}|$ *is the largest entry in* $|A(i:n, i:n)|$**/*

*/* compute column i of L */*

*for j=i+1 to n*

$l_{ji} = \frac{a_{ji}}{a_{ii}}$

*end for*

*/* compute row i of U */*

*for j=i to n*

$u_{ij} = a_{ij}$ *end for*

# The Algorithm

## Algorithm A

*/* update $A_{22}$ */*
*for j=i+1 to n*
*for k=i+1 to n*
$a_{jk} = a_{jk} - l_{ji} * u_{ik}$
*end for*
*end for*
*end for*

# Discussion on Algorithm A

Note that once column $i$ of $A$ is used to compute column $i$ of $L$, it is never used again. Similarly, row $i$ of $A$ is never used again after computing row $i$ of $U$. This lets us overwrite $L$ and $U$ on top of $A$ as they are computed, so we need no extra space to store them; $L$ occupies the (strict) lower triangle of $A$ (the ones on the diagonal of $L$ are not stored explicitly), and $U$ occupies the upper triangle of $A$. This simplifies the algorithm A to the following algorithm B.

# LU factorization

## Algorithm B

*LU factorization with pivoting, overwriting L and U on A:*

*for i=1 to n*
    *apply permutations (see Algorithm A)*
    *for j=i+1 to n*
        $a_{ji} = \frac{a_{ji}}{a_{ii}}$
    *end for*
    *for j=i+1 to n*
        *for k=i+1 to n*
            $a_{jk} = a_{jk} - a_{ji} * a_{ik}$
        *end for*
    *end for*
*end for*

# PLU factorization with pivoting

PLU factorization with pivoting: calculating the permutation matrix $P$, the unit lower triangular matrix $L$, and the nonsingular upper triangular matrix $U$ such that $LU = PA$ for a given nonsingular $A$.

```
let P=I, L=I, U=A
 for i = 1 to n-1
 find m such that |U(m,i)| is the largest entry in
|U(i:n,i)|
 if m  ~= i
 swap rows m and i in P
 swap rows m and i in U
 if i >= 2  swap elements L(m,1:i-1) and L(i,1:i-1)
 end if
 L(i+1:n,i)=U(i+1:n,i)/U(i,i)
U(i+1:n,i+1:n)=U(i+1:n,i+1:n)-L(i+1:n,i) U(i,i+1:n)
U(i+1:n,i)=0
end for
```

# Forward substitution

The next algorithm is *forward substitution*. We use it to easily solve a given system $Lx = b$ with a unit lower triangular matrix $L$.
**Algorithm:** forward substitution: solving $Lx = b$ with a unit lower triangular matrix $L$.

```
x(1)=b(1)
 for i=2 to n
x(i)=b(i)-L(i,1:(i-1)) x(1:(i-1))
end for
```

# Backward substitution

Using Backward substitution, we easily solve a given system $Ux = b$ with an upper triangular matrix $U$.
**Backward substitution**: solving $Ux = b$ with a nonsingular upper triangular matrix $U$.

```
x(n)=b(n)/U(n,n)
for i=n-1 to 1
x(i)=(b(i)-U(i,(i+1):n)\ x((i+1):n))/U(i,i)
end for
```

# LU factorization

## Example

$$A = \begin{bmatrix} 2 & 6 \\ 4 & 15 \end{bmatrix}; \quad L-? \quad U-? \quad A = LU$$

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 \\ \ell_{21} & \ell_{22} \end{bmatrix} \cdot \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}$$

$$= \begin{bmatrix} u_{11}\ell_{11} & \ell_{11}u_{12} \\ \ell_{21}u_{11} & \ell_{21} \cdot u_{12} + \ell_{22} \cdot u_{22} \end{bmatrix}$$

$$\ell_{11} \cdot u_{11} = a_{11} \Rightarrow L = \begin{bmatrix} 1 & 0 \\ \ell_{21} & 1 \end{bmatrix}$$

## Example

$$\Rightarrow u_{11} = \frac{a_{11}}{\ell_{11}} = a_{11}$$

$$\ell_{11} \cdot u_{12} = a_{12} \Rightarrow u_{12} = a_{12}$$

$$\ell_{21} \cdot u_{11} = a_{21} \Rightarrow \ell_{21} = \frac{a_{21}}{u_{11}} = \frac{a_{21}}{a_{11}} = \frac{4}{2} = 2$$

$$\ell_{21} \cdot u_{12} + \ell_{22} \cdot u_{22} = a_{22} \Rightarrow 2 \cdot a_{12} + 1 \cdot u_{22} = a_{22} \Rightarrow$$

$$u_{22} = a_{22} - 2 \cdot a_{12} = 15 - 2 \cdot 6 = 3$$

$$\underbrace{\begin{bmatrix} 2 & 6 \\ 4 & 15 \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}}_{L} \cdot \underbrace{\begin{bmatrix} 2 & 6 \\ 0 & 3 \end{bmatrix}}_{U}$$

# The need of pivoting. Example.

Let us consider $LU$ factorization without pivoting for

$$A = \begin{bmatrix} 10^{-4} & 1 \\ 1 & 1 \end{bmatrix}$$

in three-decimal-digit floating point.
First, $A$ is well-conditioned since $k(A) = ||A||_\infty \cdot ||A^{-1}||_\infty \approx 4$. Thus, we can expect to solve $Ax = b$ accurately. We apply $LU$ factorization without pivoting to get:

$$L = \begin{bmatrix} 1 & 0 \\ 10^4 & 1 \end{bmatrix}, U = \begin{bmatrix} 10^{-4} & 1 \\ 0 & -10^4 \end{bmatrix}$$

Then $LU$ is not the same as $A$:

$$LU = \begin{bmatrix} 10^{-4} & 1 \\ 1 & 0 \end{bmatrix}$$

# Warning: loss of accuracy

Compare the condition number of $A$ to the condition numbers of $L$ and $U$.

- $k(A) = ||A||_\infty \cdot ||A^{-1}||_\infty \approx 4$
- $k(L) = ||L||_\infty \cdot ||L^{-1}||_\infty \approx 10^8$
- $k(U) = ||U||_\infty \cdot ||U^{-1}||_\infty \approx 10^8$

Since $k(A) << k(L) \cdot k(U)$ - warning:loss of accuracy.

# The need of pivoting. Example.

Let us consider $LU$ factorization with pivoting for matrix $A$ with reversed order of equations:

$$A = \begin{bmatrix} 1 & 1 \\ 10^{-4} & 1 \end{bmatrix}$$

Now we apply $LU$ factorization to the above matrix $A$ and get:

$$L = \begin{bmatrix} 1 & 0 \\ 10^{-4} & 1 \end{bmatrix}, U = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Then $LU$ approximates matrix $A$ accurately:

$$A \approx LU = \begin{bmatrix} 1 & 0 \\ 10^{-4} & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Both $L$ and $U$ are well-conditioned here and the computed solution is also quite accurate.

# Error analysis

Recall our two-step paradigm for obtaining error bounds for the solution of $Ax = b$:

1. Analyze round-off errors to show that the result of solving $Ax = b$ is the exact solution $\hat{x}$ of the perturbed system $(A + \delta A)\hat{x} = b + \delta b$, where $\delta A$ and $\delta b$ are small. This is an example of backward error analysis, and $\delta A$ and $\delta b$ are called the backward errors.

# Error analysis

Recall our two-step paradigm for obtaining error bounds for the solution of $Ax = b$:

1. Analyze round-off errors to show that the result of solving $Ax = b$ is the exact solution $\hat{x}$ of the perturbed system $(A + \delta A)\hat{x} = b + \delta b$, where $\delta A$ and $\delta b$ are small. This is an example of backward error analysis, and $\delta A$ and $\delta b$ are called the backward errors.
2. Apply the perturbation theory to bound the error.

# Error analysis

Now suppose that matrix $A$ has already been pivoted, so the notation is simpler. We simplify Algorithm A. to two equations - one for $a_{jk}, j \leq k$ and one for $j > k$.
What is Algorithm A. doing to$A_{jk}$ when $j \leq k$?
This element is repeatedly updated by subtracting $l_{ji}u_{ik}$ for i=1 to j-1 and is assigned to $u_{jk}$ so that:

$$u_{jk} = a_{jk} - \sum_{i=1}^{j-1} l_{ji}u_{ik}$$

When $j > k$, $a_{jk}$ again has $l_{ji}u_{ik}$ subtracted for i=1 to k-1 and the resulting sum is divided by $u_{kk}$ and assigned to $l_{jk}$:

$$l_{jk} = \frac{a_{jk} - \sum_{i=1}^{k-1} l_{ji}u_{ik}}{u_{kk}}$$

# Error analysis

To do roundoff error analysis of these two formulas, we use the following:

$$fl\left(\sum_{i=1}^{d} x_i y_i\right) = \sum_{i=1}^{d} x_i y_i(1+\delta_i), |\delta_i| \leq d\epsilon$$

We apply this to the formula for $u_{jk}$:

$$u_{jk} = \left(a_{jk} - \sum_{i=1}^{j-1} l_{ji} u_{ik}(1+\delta_i)\right)(1+\delta')$$

with $|\delta_i| \leq (j-1)\epsilon$ and $|\delta'| \leq \epsilon$.

# Error analysis

Solving for $a_{jk}$ we get:

$$a_{jk} = \frac{1}{1+\delta'} u_{jk} \cdot l_{jj} + \sum_{i=1}^{j-1} l_{ji} u_{ik} (1+\delta_i)$$

$$\leq \sum_{i=1}^{j} l_{ji} u_{ik} + \sum_{i=1}^{j} l_{ji} u_{ik} \delta_i \equiv \sum_{i=1}^{j} l_{ji} u_{ik} + E_{jk}$$

where we have used

$$|\delta_i| \leq (j-1)\varepsilon, \quad 1+\delta_j \equiv \frac{1}{1+\delta'}$$

then

$$\equiv \sum_{i=1}^{j} l_{ji} u_{ik} + E_{jk}.$$

In the expression above we can bound $E_{jk}$ by

$$|E_{jk}| = |\sum_{i=1}^{j} l_{ji} u_{ik} \delta_i| \leq \sum_{i=1}^{j} |l_{ji}| \cdot |u_{ik}| \cdot n\epsilon = n\epsilon(|L| \cdot |U|)_{jk}$$

Here $\varepsilon$ is the relative representation error. The maximum of the relative representation error in a floating point arithmetic with $p$ digits and base $\beta$ is $0.5 \times \beta^{1-p}$. IEEE arithmetics includes two kinds of floating point numbers: single precision (32 bits long) and double precision (64 bits long).

# Error analysis

Doing the same analysis for the formula for $l_{jk}$ gives us:

$$l_{jk} = (1+\delta'')\left(\frac{(1+\delta')(a_{jk} - \sum_{i=1}^{k-1} l_{ji}u_{ik}(1+\delta_i))}{u_{kk}}\right)$$

with $|\delta_i| \leq (k-1)\epsilon$, $|\delta'| \leq \epsilon$ and $|\delta''| \leq \epsilon$. Solving for $a_{jk}$ we get

$$a_{jk} = \frac{1}{(1+\delta')(1+\delta'')}u_{kk}l_{jk} + \sum_{i=1}^{k-1} l_{ji}u_{ik}(1+\delta_i)$$

$$\leq \sum_{i=1}^{k} l_{ji}u_{ik} + \sum_{i=1}^{k} l_{ji}u_{ik}\delta_i \equiv \sum_{i=1}^{k} l_{ji}u_{ik} + E_{jk}$$

with $|\delta_i| \leq n\epsilon$, $1+\delta_k = \frac{1}{(1+\delta')(1+\delta'')}$, and so $|E_{jk}| \leq n\epsilon(|L|\cdot|U|)_{jk}$ as before.

# Error analysis

We can summarize this error analysis with the simple formula:

$$A = LU + E$$

where

$$|E| \leq n\epsilon |L| \cdot |U|$$

and taking norms we get

$$\| |E| \| \leq n\epsilon \| \ |L| \ \| \cdot \| \ |U| \ \|$$

If the norm does not depends on the sign of the entries of matrix (this is valid for Frobenius, infinity, one-norms but not for two-norms) we can simplify expression above as

$$\|E\| \leq n\epsilon \|L\| \cdot \|U\|$$

# Error in Gaussian elimination

- Recall that we solve $L\underbrace{Ux}_{y} = b$ via $Ly = b$ and $Ux = y$.
- Solving $Ly = b$ gives as a computed solution $\hat{y}$ such that $(L + \delta L)\hat{y} = b$ where $|\delta L| \leq n\varepsilon|L|$.
- The same is true for $(U + \delta U)\hat{x} = \hat{y}$ with $|\delta U| \leq n\varepsilon|U|$.
- Combining both estimates into one we get

$$\begin{aligned} b = (L + \delta L)\hat{y} &= (L + \delta L)(U + \delta U)\hat{x} \\ &= (LU + L\delta U + \delta LU + \delta L\delta U)\hat{x} \\ &= (A - E + L\delta U + \delta LU + \delta L\delta U)\hat{x} \\ &= (A + \delta A)\hat{x}, \end{aligned}$$

where

$$\delta A = -E + L\delta U + \delta LU + \delta L\delta U.$$

Now we combine all bounds for $E, \delta U, \delta L$ and use triangle inequality to get

$$|\delta A| \leq | - E + L\delta U + \delta L U + \delta L \delta U|$$
$$\leq |E| + |L| \cdot |\delta U| + |\delta L| \cdot |U| + |\delta L| \cdot |\delta U|$$
$$\leq n\varepsilon |L| \cdot |U| + n\varepsilon |L| \cdot |U| + n\varepsilon |L| \cdot |U| + n^2\varepsilon^2 |L| \cdot |U|$$
$$\approx 3n\varepsilon |L| \cdot |U|.$$

Assuming that $\| \, |X| \, \| = \|X\|$ is true (as before for Frobenius, infinity, one-norms but not for two-norms) we obtain $||\delta A|| \leq 3n\varepsilon ||L|| \cdot ||U||$. Thus, the Gaussian elimination is backward stable when $3n\varepsilon ||L|| \cdot ||U|| = O(\varepsilon)||A||$ since then

$$\frac{\|\delta A\|}{\|A\|} = O(\varepsilon).$$

In this analysis we have used $\delta b = 0$.

# Estimating Condition Number

To compute a practical error bound based on a bound

$$\|\delta x\| = \|A^{-1}r\| \leq \|A^{-1}\|\|r\|$$

we need to estimate $||A^{-1}||$. This is also enough to estimate the condition number $k(A) = ||A^{-1}|| \cdot ||A||$, since $||A||$ is easy to compute. One approach is to compute $A^{-1}$ explicitly and compute its norm. However, this would cost $2n^3$, more than the original $\frac{2}{3}n^3$ for Gaussian elimination. It is a fact that most users will not bother to compute error bounds if they are expensive.
So instead of computing $A^{-1}$ we will devise a much cheaper algorithm to *estimate* $||A^{-1}||$.

# Estimating Condition Numbers

Such an algorithm is called a *condition estimator* and should have the following properties:

1. Given the $L$ and $U$ factors of $A$, it should cost $O(n^2)$, which for large enough $n$ is negligible compared to the $\frac{2}{3}n^3$ cost of GEPP.
2. It should provide an estimate which is almost always within a factor of 10 of $||A^{-1}||$. This is all one needs for an error bound which tells you about how many decimal digits of accuracy that you have.

# Estimating Condition Numbers

- There are a variety of such estimators available. We choose one to solve $Ax = b$.
- This estimator is guaranteed to produce only a lower bound on $||A^{-1}||$, not an upper bound.
- It is almost always within a factor of 10, and usually 2 to 3, of $||A^{-1}||$.
- The algorithm estimates the one-norm $||B||_1$ of a matrix $B$, provided that we can compute $Bx$ and $B^T y$ for arbitrary $x$ and $y$. We will apply the algorithm to $B = A^{-1}$, so we need to compute $A^{-1}x$ and $A^{-T}y$, i.e., solve linear systems. This costs just $O(n^2)$ given the $LU$ factorization of $A$.

The algorithm was developed in:
W. W. Hager. Condition estimators. SIAM J. Sci. Statist. Comput., 5:311-316, 1984.
N. J. Higham. A survey of condition number estimation for triangular matrices. SIAM Rev., 29:575-596, 1987.
N. J. Higham. Experience with a matrix norm estimator. SIAM J. Sci. Statist. Comput., 11:804-809, 1990.
with the latest version in [N. J. Higham. FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation. ACM Trans. Math. Software].

Recall that $||B||_1$ is defined by

$$||B||_1 = \max_{x \neq 0} \frac{||Bx||_1}{||x||_1} = \max_j \sum_{i=1}^{n} |b_{ij}|.$$

It is easy to show that the maximum over $x \neq 0$ is attained at $x = e_{j_0}[0, \ldots, 0, 1, 0, \ldots, 0]^T$. (The single nonzero entry is component $j_0$, where $\max_j \sum_i |b_{ij}|$ occurs at $j = j_0$.)
Searching over all $e_j$, $j = 1, \ldots, n$, means computing all columns of $B = A^{-1}$; this is too expensive. Instead, since $||Bx||_1 = \max_{||x||_1 \leq 1} ||Bx||_1$, we can use *hill climbing* or *gradient ascent* on $f(x) \equiv ||Bx||_1$ inside the set $||x||_1 \leq 1$. $||x||_1 \leq 1$ is clearly a convex set of vectors, and $f(x)$ is a convex function, since $0 \leq \alpha \leq 1$ implies $f(\alpha x + (1-\alpha)y) = ||\alpha Bx + (1-\alpha)By||_1 \leq \alpha||Bx||_1 + (1-\alpha)||By||_1 = \alpha f(x) + (1-\alpha)f(y)$.

Doing gradient ascent to maximize $f(x)$ means moving $x$ in the direction of the gradient $\nabla f(x)$ (if it exists) as long as $f(x)$ increases. The convexity of $f(x)$ means $f(y) \geq f(x) + \nabla f(x) \cdot (y - x)$ (if $\nabla f(x)$ exists). To compute $\nabla f(x)$ we assume all $\sum_j b_{ij} x_j \neq 0$ in $f(x) = \sum_i |\sum_j |b_{ij} x_j|$ (this is almost always true). Let $\zeta_i = sign(\sum_j b_{ij} x_j)$, so $\zeta_i = \pm 1$ and $f(x) = \sum_i \sum_j \zeta_i b_{ij} x_j$. Then $\dfrac{\partial f}{\partial x_k} = \sum_i \zeta_i b_{ik}$ and $\nabla f = \zeta^T B = (B^T \zeta)^T$.

In summary, to compute $\nabla f(x)$ takes three steps: $\omega = Bx$, $\zeta = sign(\omega)$ and $\nabla f(x) = \zeta^T B$.

ALGORITHM *Hager's condition estimator returns a lower bound* $||\omega||_1$ *on* $||B||_1$*:*

*choose any* $x$ *such that* $||x||_1 = 1$ $\quad$ $/^* \, e.g. \, x_i = \frac{1}{n} \, ^*/$

*repeat*

$\omega = Bx$, $\zeta = sign(\omega)$, $z = B^T \zeta$, $\quad$ $/^* \, z^T = \nabla f \, ^*/$

*if* $||z||_\infty \leq z^T x$ *then*

$\quad$ *return* $||\omega||_1$

*else*

$\quad x = e_j$ *where* $|z_j| = ||z||_\infty$

*end if*

*end repeat*

THEOREM 1. *When* $||\omega||_1$ *is returned,* $||\omega||_1 = ||Bx||_1$ *is a local maximum of* $||Bx||_1$.
2. *Otherwise,* $||Be_j||$ *(at end of loop)* $> ||Bx||$ *(at start), so the algorithm has made progress in maximizing* $f(x)$.
*Proof.*
1. In this case, $||z||_\infty \leq z^T x$ (*). Near $x$, $f(x) = ||Bx||_1 = \sum_i \sum_j \zeta_i b_{ij} x_j$ is linear in $x$ so $f(y) = f(x) + \nabla f(x) \cdot (y - x) = f(x) + z^T(y - x)$, where $z^T = \nabla f(x)$. To show $x$ is a local maximum we want $z^T(y - x) \leq 0$ when $||y||_1 = 1$. We compute

$$\begin{aligned} z^T(y-x) &= z^T y - z^T x = \textstyle\sum_i z_i \cdot y_i - z^T x \leq \sum_i |z_i| \cdot |y_i| - z^T x \\ &\leq ||z||_\infty \cdot ||y||_1 - z^T x = \underbrace{||z||_\infty - z^T x}_{see(*)} \leq 0. \end{aligned}$$

2. In this case $||z||_\infty > z^T x$. Choose $\widetilde{x} = e_j \cdot sign(z_j)$, where $j$ is chosen so that $|z_j| = ||z||_\infty$. Then

$$\begin{aligned} f(\widetilde{x}) &= f(x) + \nabla f \cdot (\widetilde{x} - x) = f(x) + z^T(\widetilde{x} - x) \\ &= f(x) + z^T\widetilde{x} - z^T x = f(x) + |z_j| - z^T x > f(x), \end{aligned}$$

where the last inequality is true by construction. □

# Remarks on Theorem

- Higham [FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation; Experience with a matrix norm estimator] tested a slightly improved version of this algorithm by trying many random matrices of sizes 10,25,50 and condition numbers $k = 10, 10^3, 10^6, 10^9$; in the worst case the computed $k$ underestimated the true $k$ by a factor 0.44.
- A different condition estimator is available in Matlab as rcond.
- The Matlab routine cond computes the exact condition number $||A^{-1}||_2||A||_2$, it is much more expensive than rcond.

# Estimating the Relative Condition Number

We can apply the Hager's algorithm to estimate the relative condition number $k_{CR}(A) = || \, |A^{-1}| \cdot |A| \, ||_\infty$ or to evaluate the bound $|| \, |A^{-1}| \cdot |r| \, ||_\infty$. We can reduce both to the same problem, that of estimating $|| \, |A^{-1}| \cdot g \, ||_\infty$, where $g$ is a vector of nonnegative entries. To see why, let $e$ be the vector of all ones. From definition of norm, we see that $||X||_\infty = ||Xe||_\infty$ if the matrix $X$ has nonnegative entries. Then

$$|| \, |A^{-1}| \cdot |A| \, ||_\infty = || \, |A^{-1}| \cdot |A|e \, ||_\infty = || \, |A^{-1}| \cdot g \, ||_\infty,$$

where $g = |A|e$.

Here is how we estimate $|| \, |A^{-1}| \cdot g \, ||_\infty$. Let $G = diag(g_1, \ldots, g_n)$; then $g = Ge$. Thus

$$\begin{aligned} || \, |A^{-1}| \cdot g \, ||_\infty &= || \, |A^{-1}| \cdot Ge \, ||_\infty = || \, |A^{-1}| \cdot G \, ||_\infty = \\ &= || \, |A^{-1}G| \, ||_\infty = ||A^{-1}G||_\infty. \end{aligned} \tag{2.12}$$

The last equality is true because $||Y||_\infty = || \, |Y| \, ||_\infty$ for any matrix $Y$. Thus, it suffices to estimate the infinity norm of the matrix $A^{-1}G$. We can do this by applying Hager's algorithm to the matrix $(A^{-1}G)^T = GA^{-T}$, to estimate $||(A^{-1}G)^T||_1 = ||A^{-1}G||_\infty$ (see definition of norm). This requires us to multiply by the matrix $GA^{-T}$ and its transpose $A^{-1}G$. Multiplying by $G$ is easy since it is diagonal, and we multiply by $A^{-1}$ and $A^{-T}$ using the $LU$ factorization of $A$.

# Practical Error Bounds

We present two practical error bounds for our approximate solution $\widetilde{x}$ of $Ax = b$. For the first bound we use inequality $||\widetilde{x} - x||_\infty \leq ||A^{-1}||_\infty \cdot ||r||_\infty$ to get

$$error = \frac{||\widetilde{x} - x||_\infty}{||\widetilde{x}||_\infty} \leq ||A^{-1}||_\infty \cdot \frac{||r||_\infty}{||\widetilde{x}||_\infty}, \tag{2.13}$$

where $r = A\widetilde{x} - b$ is the residual. We estimate $||A^{-1}||_\infty$ by applying Algorithm  to $B = A^{-T}$, estimating $||B||_1 = ||A^{-T}||_1 = ||A^{-1}||_\infty$ (see definition of norm).
Our second error bound comes from the inequality:

$$error = \frac{||\widetilde{x} - x||_\infty}{||\widetilde{x}||_\infty} \leq \frac{|||A^{-1}| \cdot |r|||_\infty}{||\widetilde{x}||_\infty}. \tag{2.14}$$

We estimate $|||A^{-1}| \cdot |r|||_\infty$ using the algorithm based on equation (2.12).

# What Can Go Wrong

- Error bounds (2.13) and (2.14) are not guaranteed to provide bounds in all cases in practice.
- First, the estimate of $||A^{-1}||$ from Algorithm (or similar algorithms) provides only a lower bound, although the probability is very low that it is more than 10 times too small.
- Second, there is a small but non-negligible probability that roundoff in the evaluation of $r = A\hat{x} - b$ might make $||r||$ artificially small, in fact zero, and so also make our computed error bound too small. To take this possibility into account, one can add a small quantity to $|r|$ to account for it: the roundoff in evaluating $r$ is bounded by

$$|(A\hat{x} - b) - fl(A\hat{x} - b)| \leq (n+1)\varepsilon(|A| \cdot |\hat{x}| + |b|), \qquad (2.15)$$

  so we can replace $|r|$ with $|r| + (n+1)\varepsilon(|A| \cdot |\hat{x}| + |b|)$ in bound (2.14) or $||r||$ with $||r|| + (n+1)\varepsilon(||A|| \cdot ||\hat{x}|| + ||b||)$ in bound (2.13).
- Third, roundoff in performing Gaussian elimination on very ill-conditioned matrices can yield such inaccurate $L$ and $U$ that bound (2.14) is much too low.

# Invertible matrix

- In linear algebra an $n$-by-$n$ (square) matrix $A$ is called invertible (some authors use nonsingular or nondegenerate) if there exists an $n$-by-$n$ matrix $B$ such that $\mathbf{AB} = \mathbf{BA} = \mathbf{I}_n$, where $I_n$ denotes the $n$-by-$n$ identity matrix and the multiplication used is ordinary matrix multiplication. If this is the case, then the matrix $B$ is uniquely determined by $A$ and is called the inverse of $A$, denoted by $A^{-1}$. It follows from the theory of matrices that if $\mathbf{AB} = \mathbf{I}$ for finite square matrices $A$ and $B$, then also $\mathbf{BA} = \mathbf{I}$.
- Non-square matrices ($m$-by-$n$ matrices which do not have an inverse). However, in some cases such a matrix may have a left inverse or right inverse. If A is $m$-by-$n$ and the rank of $A$ is equal to $n$, then $A$ has a left inverse: an $n$-by-$m$ matrix $B$ such that $BA = I$. If $A$ has rank $m$, then it has a right inverse: an $n$-by-$m$ matrix $B$ such that $AB = I$.
- A square matrix that is not invertible is called singular or degenerate. A square matrix is singular if and only if its determinant is 0.

# Methods of matrix inversion

- Gaussian elimination
- Gauss-Jordan elimination is an algorithm that can be used to determine whether a given matrix is invertible and to find the inverse.
- An alternative is the LU decomposition which generates upper and lower triangular matrices which are easier to invert. For special purposes, it may be convenient to invert matrices by treating $mn$-by-$mn$ matrices as $m$-by-$m$ matrices of $n$-by-$n$ matrices, and applying one or another formula recursively (other sized matrices can be padded out with dummy rows and columns).
- For other purposes, a variant of Newton's method may be convenient. Newton's method is particularly useful when dealing with families of related matrices: sometimes a good starting point for refining an approximation for the new inverse can be the already obtained inverse of a previous matrix that nearly matches the current matrix.

# Eigendecomposition

Let $A$ be a square $n \times n$ matrix. Let $q_1 ... q_k$ be an eigenvector basis, i.e. an indexed set of $k$ linearly independent eigenvectors, where $k$ is the dimension of the space spanned by the eigenvectors of $A$. If $k = n$, then $A$ can be written

$$\mathbf{A} = \mathbf{QUQ}^{-1}$$

where $Q$ is the square $n \times n$ matrix whose $i$-th column is the basis eigenvector $q_i$ of $A$, and $U$ is the diagonal matrix whose diagonal elements are the corresponding eigenvalues, i.e. $U_{ii} = \lambda_i$.

# Properties

Let $A$ be an $n \times n$ matrix with eigenvalues $\lambda_i, i = 1, 2, \ldots, n$. Then

- Trace of $A$
$$\text{tr}(A) = \sum_{i=1}^{n} \lambda_i = \lambda_1 + \lambda_2 + \cdots + \lambda_n.$$
- Determinant of $A$
$$\det(A) = \prod_{i=1}^{n} \lambda_i = \lambda_1 \lambda_2 \cdots \lambda_n.$$
- Eigenvalues of $A^k$ are $\lambda_1^k, \ldots, \lambda_n^k$.

  These first three results follow by putting the matrix in upper-triangular form, in which case the eigenvalues are on the diagonal and the trace and determinant are respectively the sum and product of the diagonal.
- If $A = A^H$, i.e., $A$ is Hermitian ($A = \overline{A^T}$), every eigenvalue is real.
- Every eigenvalue of unitary matrix U ($U^* U = U U^* = I$) has absolute value $|\lambda| = 1$.

# Example

We take a $2 \times 2$ matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix}$$

and want it to be decomposed into a diagonal matrix. First, we multiply to a non-singular matrix

$$\mathbf{B} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, [a, b, c, d] \in \mathbb{R}.$$

Then

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}_{-1} \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} x & 0 \\ 0 & y \end{bmatrix},$$

for some real diagonal matrix

$$\begin{bmatrix} x & 0 \\ 0 & y \end{bmatrix}.$$

Shifting B to the right hand side:

$$\begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x & 0 \\ 0 & y \end{bmatrix}$$

The above equation can be decomposed into 2 simultaneous equations:

$$\begin{cases} \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} a \\ c \end{bmatrix} = \begin{bmatrix} ax \\ cx \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} by \\ dy \end{bmatrix} \end{cases}$$

Factoring out the eigenvalues $x$ and $y$:

$$\begin{cases} \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} a \\ c \end{bmatrix} = x \begin{bmatrix} a \\ c \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} b \\ d \end{bmatrix} = y \begin{bmatrix} b \\ d \end{bmatrix} \end{cases}$$

Letting

$$\vec{a} = \begin{bmatrix} a \\ c \end{bmatrix}, \vec{b} = \begin{bmatrix} b \\ d \end{bmatrix},$$

this gives us two vector equations:

$$\begin{cases} A\vec{a} = x\vec{a} \\ A\vec{b} = y\vec{b} \end{cases}$$

And can be represented by a single vector equation involving 2 solutions as eigenvalues:

$\mathbf{Au} = \lambda\mathbf{u}$

where $\lambda$ represents the two eigenvalues x and y, $\mathbf{u}$ represents the vectors $\overrightarrow{a}$ and $\overrightarrow{b}$.

Shifting $\lambda\mathbf{u}$ to the left hand side and factorizing $\mathbf{u}$ out

$(\mathbf{A} - \lambda\mathbf{I})\mathbf{u} = 0$

Since $\mathbf{B}$ is non-singular, it is essential that $\mathbf{u}$ is non-zero. Therefore,

$(\mathbf{A} - \lambda\mathbf{I}) = \mathbf{0}$

Considering the determinant of $(\mathbf{A} - \lambda\mathbf{I})$,

$$\begin{bmatrix} 1-\lambda & 0 \\ 1 & 3-\lambda \end{bmatrix} = 0$$

Thus

$(1 - \lambda)(3 - \lambda) = 0$

Giving us the solutions of the eigenvalues for the matrix **A** as $\lambda = 1$ or $\lambda = 3$, and the resulting diagonal matrix from the eigendecomposition of **A** is thus
$\begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix}$.
Putting the solutions back into the above simultaneous equations

$$\begin{cases} \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} a \\ c \end{bmatrix} = 1 \begin{bmatrix} a \\ c \end{bmatrix} \\ \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} b \\ d \end{bmatrix} = 3 \begin{bmatrix} b \\ d \end{bmatrix} \end{cases}$$

Solving the equations, we have $a = -2c, a \in \mathbb{R}$ and $b = 0, d \in \mathbb{R}$
Thus the matrix **B** required for the eigendecomposition of **A** is
$\begin{bmatrix} -2c & 0 \\ c & d \end{bmatrix}$, [c, d]$\in \mathbb{R}$.*i.e.* :

$$\begin{bmatrix} -2c & 0 \\ c & d \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} -2c & 0 \\ c & d \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix}, [c, d] \in \mathbb{R}$$

# Methods of matrix inversion

- Eigendecomposition

  If matrix $A$ can be eigendecomposed and if none of its eigenvalues are zero, then $A$ is nonsingular and its inverse is given by $\mathbf{A}^{-1} = \mathbf{Q}\Lambda^{-1}\mathbf{Q}^{-1}$.

  Furthermore, because $U$ is a diagonal matrix, its inverse is easy to calculate: $\left[\Lambda^{-1}\right]_{ii} = \frac{1}{\lambda_i}$.

- Cholesky decomposition

  If matrix $A$ is positive definite, then its inverse can be obtained as $\mathbf{A}^{-1} = (\mathbf{L}^*)^{-1}\mathbf{L}^{-1}$ , where $L$ is the lower triangular Cholesky decomposition of $A$.

# Methods of matrix inversion

- Analytic solution

  Writing the transpose of the matrix of cofactors, known as an adjugate matrix, can also be an efficient way to calculate the inverse of small matrices, but this recursive method is inefficient for large matrices. To determine the inverse, we calculate a matrix of cofactors:

$$\mathbf{A}^{-1} = \frac{1}{|\mathbf{A}|} \left(\mathbf{C}^{\mathrm{T}}\right)_{ij} = \frac{1}{|\mathbf{A}|} \left(\mathbf{C}_{ji}\right) = \frac{1}{|\mathbf{A}|} \begin{pmatrix} \mathbf{C}_{11} & \mathbf{C}_{21} & \cdots & \mathbf{C}_{n1} \\ \mathbf{C}_{12} & \mathbf{C}_{22} & \cdots & \mathbf{C}_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}_{1n} & \mathbf{C}_{2n} & \cdots & \mathbf{C}_{nn} \end{pmatrix}$$

  where $|A|$ is the determinant of $A$, $C_{ij}$ is the matrix of cofactors, and $C^T$ represents the matrix transpose.

# Inversion of $2 \times 2$ matrices

The cofactor equation listed above yields the following result for $2 \times 2$ matrices. Inversion of these matrices can be done easily as follows:

$$\mathbf{A}^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{\det(\mathbf{A})} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}.$$

This is possible because $1/(ad - bc)$ is the reciprocal of the determinant of the matrix in question, and the same strategy could be used for other matrix sizes.

# Inversion of $3 \times 3$ matrices

A computationally efficient $3 \times 3$ matrix inversion is given by

$$\mathbf{A}^{-1} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & k \end{bmatrix}^{-1} = \frac{1}{\det(\mathbf{A})} \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & K \end{bmatrix}^{T} = \frac{1}{\det(\mathbf{A})} \begin{bmatrix} A & D & G \\ B & E & H \\ C & F & K \end{bmatrix}$$

where the determinant of $A$ can be computed by applying the rule of Sarrus as follows:
$\det(\mathbf{A}) = a(ek - fh) - b(kd - fg) + c(dh - eg)$.
If the determinant is non-zero, the matrix is invertible, with the elements of the above matrix on the right side given by

$$\begin{array}{lll} A = (ek - fh) & D = (ch - bk) & G = (bf - ce) \\ B = (fg - dk) & E = (ak - cg) & H = (cd - af) \\ C = (dh - eg) & F = (gb - ah) & K = (ae - bd). \end{array}$$

# Eigenvalues and eigenvectors

- The vector $x$ is an eigenvector of the matrix $A$ with eigenvalue $\lambda$ (lambda) if the following equation holds: $\mathbf{Ax} = \lambda\mathbf{x}$.

  If the eigenvalue $\lambda > 1, x$ is stretched by this factor. If $\lambda = 1$, the vector $x$ is not affected at all by multiplication by $A$. If $0 < \lambda < 1$, $x$ is shrunk (or compressed). The case $\lambda = 0$ means that $x$ shrinks to a point (represented by the origin), meaning that $x$ is in the kernel of the linear map given by $A$. If $\lambda < 0$ then $x$ flips and points in the opposite direction as well as being scaled by a factor equal to the absolute value of $\lambda$.

- As a special case, the identity matrix $I$ is the matrix that leaves all vectors unchanged: $I\mathbf{x} = 1\mathbf{x} = \mathbf{x}.$,

- Every non-zero vector $x$ is an eigenvector of the identity matrix with eigenvalue $\lambda = 1$.

# Eigenvalues and eigenvectors

- The eigenvalues of $A$ are precisely the solutions $\lambda$ to the equation $\det(A - \lambda I) = 0$.

  Here *det* is the determinant of the matrix formed by $A - \lambda I$. This equation is called the characteristic equation of $A$. For example, if $A$ is the following matrix (a so-called diagonal matrix):

$$A = \begin{bmatrix} a_{1,1} & 0 & \cdots & 0 \\ 0 & a_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & a_{n,n} \end{bmatrix},$$

  then the characteristic equation reads

$$\det(A - \lambda I) = \det\left(\begin{bmatrix} a_{1,1} & 0 & \cdots & 0 \\ 0 & a_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & a_{n,n} \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\right)$$

$$= \det \begin{bmatrix} a_{1,1} - \lambda & 0 & \cdots & 0 \\ 0 & a_{2,2} - \lambda & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & a_{n,n} - \lambda \end{bmatrix}$$

$$= (a_{1,1} - \lambda)(a_{2,2} - \lambda) \cdots (a_{n,n} - \lambda) = 0.$$

The solutions to this equation are the eigenvalues $\lambda i = ai, i(i = 1, ..., n)$.
The eigenvalue equation for a matrix $A$ can be expressed as
$A\mathbf{x} - \lambda I\mathbf{x} = \mathbf{0}$,
which can be rearranged to $(A - \lambda I)\mathbf{x} = \mathbf{0}$.

A criterion from linear algebra states that a matrix (here: $A - \lambda I$) is non-invertible if and only if its determinant is zero, thus leading to the characteristic equation.

# Example

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}.$$

The characteristic equation of this matrix reads

$$\det(A - \lambda I) = \det \begin{bmatrix} 2-\lambda & 1 \\ 1 & 2-\lambda \end{bmatrix} = 0.$$

Calculating the determinant, this yields the quadratic equation $\lambda^2 - 4\lambda + 3 = 0$, whose solutions (also called roots) are $\lambda = 1$ and $\lambda = 3$. The eigenvectors for the eigenvalue $\lambda = 3$ are determined by using the eigenvalue equation, which in this case reads

$$\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 3 \begin{bmatrix} x \\ y \end{bmatrix}.$$

This equation reduces to a system of the following two linear equations:

$$2x + y = 3x,$$
$$x + 2y = 3y.$$

# Example

Both equations reduce to the single linear equation $x = y$. Or any vector of the form $(x, y)$ with $y = x$ is an eigenvector to the eigenvalue $\lambda = 3$. However, the vector $(0, 0)$ is excluded. A similar calculation shows that the eigenvectors corresponding to the eigenvalue $\lambda = 1$ are given by non-zero vectors $(x, y)$ such that $y = -x$. For example, an eigenvector corresponding to $\lambda = 1$ is

$$\begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

whereas an eigenvector corresponding to $\lambda = 3$ is

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$