

# Computing the determinant

## Group 12

Catterwell, A.   Smith, M.   Wang, R.   Watson, K.

University of Edinburgh

April 2, 2019

# Overview

- 1 Why calculate determinants?
  - Invertibility of matrices
  - Cramer's Rule
  - Eigenvalues and Eigenvectors
  - Jacobian determinant
- 2 Algorithms for computing determinants
  - Leibniz formula
  - Laplace expansion
  - LU decomposition
  - Bird's algorithm
- 3 Epilogue
  - Summary of determinant algorithms
  - How fast is Maple's implementation?

# Invertibility of matrices

## Theorem

An  $n \times n$  square matrix  $A$  is invertible if and only if

$$\det(A) \neq 0.$$

# Cramer's Rule

## Theorem

Given an equation  $A\mathbf{x} = \mathbf{b}$  The solutions for  $\mathbf{x}$  are given by

$$x_i = \frac{\det(A_i)}{\det(A)}$$

with  $A_i$  being the matrix formed by replacing the  $i$ th column of  $A$  by  $\mathbf{b}$ .

It turns out this method has the same runtime complexity as Gaussian elimination for solving systems of linear equations.

# Eigenvalues and Eigenvectors

## Definition

The Eigenvalues  $\lambda$  of a matrix  $A$  are the roots of the characteristic polynomial as defined

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

# Eigenvalues and Eigenvectors

## Definition

The Eigenvalues  $\lambda$  of a matrix  $A$  are the roots of the characteristic polynomial as defined

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

## Definition

The Eigenvectors of  $A$  are the vectors  $\mathbf{v}$  such that

$$A\mathbf{v} = \lambda\mathbf{v}.$$

# Volume

The absolute value of the determinant of real vectors is equal to the volume of the parallelepiped spanned by those vectors.

$f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ : the linear map represented by the  $A$ .  $S$ : any measurable subset of  $\mathbb{R}^n$ .

$$\text{volume}(f(S)) = |\det(A^T A)| \times \text{volume}(S).$$

# Jacobian determinant

For  $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ , the Jacobian matrix is the  $n \times n$  matrix whose entries are defined as

$$D(f) = \left( \frac{\partial f_i}{\partial x_j} \right)_{1 \leq i, j \leq n}.$$

Its determinant is known as the *Jacobian determinant*.

If the determinant of a continuously differentiable function  $f$  at a point  $p$  is...

- Non-zero,  $f$  is invertible near a point  $p$  in  $\mathbb{R}^n$ .
- Positive, then  $f$  preserves orientation near  $p$ .
- Negative, then  $f$  reverses orientation near  $p$ .



# Big-O notation

We'll be looking at the runtime of algorithms, so this is useful.

## Definition

A function  $f$  is said to be  $\mathcal{O}(g)$ , with  $g$  a function iff

$$\exists k \in \mathbb{R} \ni f(n) < k \cdot g(n)$$

for sufficiently large  $n$ .

# Leibniz formula

## Definition

The Leibniz formula defines the determinant of  $A \in \mathbb{M}(n)$  as

$$\det(A) = \sum_{\sigma \in \mathfrak{S}_n} \left( \operatorname{sgn}(\sigma) \cdot \prod_{i=1}^n a_{i, \sigma(i)} \right)$$

where  $\mathfrak{S}_n$  is the set of permutations length  $n$ .

# Leibniz formula

## Definition

The Leibniz formula defines the determinant of  $A \in \mathbb{M}(n)$  as

$$\det(A) = \sum_{\sigma \in \mathfrak{S}_n} \left( \operatorname{sgn}(\sigma) \cdot \prod_{i=1}^n a_{i,\sigma(i)} \right)$$

where  $\mathfrak{S}_n$  is the set of permutations length  $n$ .

Computing the determinant using this method is slow with runtime  $\mathcal{O}(n \cdot n!)$ .

# Laplace expansion

The Laplace (1st row) expansion for computing determinants is usually the first method taught for computing determinants of  $3 \times 3$  matrices and larger.

# Laplace expansion

The Laplace (1st row) expansion for computing determinants is usually the first method taught for computing determinants of  $3 \times 3$  matrices and larger.

## Theorem

The formula for the (1st row) Laplace expansion of  $A \in \mathbb{M}(n)$  is given as:

$$\det(A) = \sum_{j=1}^n a_{1,j} C_{1,j}$$

where  $C_{i,j} = (-1)^{i+j} \det(A\langle i,j \rangle)$  is the  $(i,j)$  cofactor of  $A$ .

# Laplace expansion

The Laplace (1st row) expansion for computing determinants is usually the first method taught for computing determinants of  $3 \times 3$  matrices and larger.

## Theorem

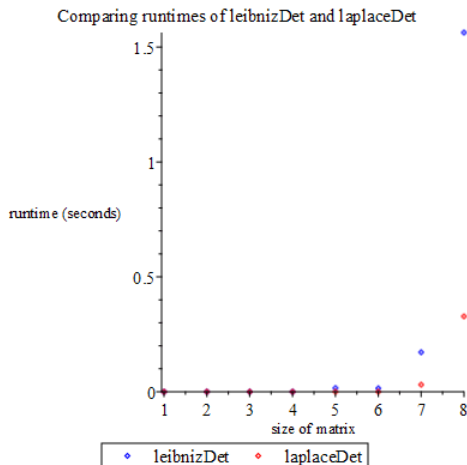
The formula for the (1st row) Laplace expansion of  $A \in \mathbb{M}(n)$  is given as:

$$\det(A) = \sum_{j=1}^n a_{1,j} C_{1,j}$$

where  $C_{i,j} = (-1)^{i+j} \det(A\langle i,j \rangle)$  is the  $(i,j)$  cofactor of  $A$ .

Its runtime complexity of  $\mathcal{O}(n!)$  is poor.

# Laplace expansion vs Leibniz formula



Runtimes are similar — both run in exponential time.

# What is LU decomposition?

## Definition

An LU decomposition of an invertible matrix  $A$  is a factorization

$$A = LU$$

where  $L$  and  $U$  are lower and upper triangular matrices, respectively.



# Is there always an LU decomposition?

No.

An LU decomposition of  $A$  exists if and only if each of its *leading principle minors* (contiguous square submatrices in the top-left corner of  $A$ ), are also invertible.

# Is there always an LU decomposition?

No.

An LU decomposition of  $A$  exists if and only if each of its *leading principle minors* (contiguous square submatrices in the top-left corner of  $A$ ), are also invertible.

## Example

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ = \begin{pmatrix} \textcolor{red}{a}_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} = \begin{pmatrix} \textcolor{red}{l}_{1,1} & 0 \\ l_{2,1} & l_{2,2} \end{pmatrix} \begin{pmatrix} \textcolor{red}{u}_{1,1} & u_{1,2} \\ 0 & u_{2,2} \end{pmatrix}$$

Here we have  $a_{1,1} = l_{1,1}u_{1,1} = 0$ .

This matrix is invertible but has no LU decomposition.

# Is there always an LU decomposition?

No.

An LU decomposition of  $A$  exists if and only if each of its *leading principle minors* (contiguous square submatrices in the top-left corner of  $A$ ), are also invertible.

## Example

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ = \begin{pmatrix} \textcolor{red}{a}_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} = \begin{pmatrix} \textcolor{red}{l}_{1,1} & 0 \\ l_{2,1} & l_{2,2} \end{pmatrix} \begin{pmatrix} \textcolor{red}{u}_{1,1} & u_{1,2} \\ 0 & u_{2,2} \end{pmatrix}$$

Here we have  $a_{1,1} = l_{1,1}u_{1,1} = 0$ .

This matrix is invertible but has no LU decomposition.

# PLU decomposition

*Partial pivoting.*

We can pivot the matrix into the correct form by multiplication with an orthogonal, permutation matrix  $P$  (representing a permutation  $\sigma_P$ ) which gives us the PLU decomposition:

$$\sigma_P(A) = PA = LU$$

# How it helps us compute determinants

Now that we have  $PA = LU$ , it follows that

$$\begin{aligned} A &= P^{-1}LU \\ &= P^T LU \end{aligned}$$

since  $P^{-1} = P^T$  by the definition of orthogonal matrices.

# How it helps us compute determinants (cont.)

Now that we have  $A = P^T L U$ , it follows that

$$\begin{aligned}\det(A) &= \det(P^T L U) \\ &= \det(P^T) \cdot \det(L) \cdot \det(U) && \text{(Thm. 4.4.1)} \\ &= \det(P) \cdot \det(L) \cdot \det(U) && \text{(Lem. 4.4.4)}\end{aligned}$$

Given that

- the determinant of a triangular matrix is the product of its diagonal elements
- the determinant of a permutation matrix ( $P$ ) is the parity of the permutation it represents ( $\sigma_P$ )

it follows that

$$\det(A) = \operatorname{sgn}(\sigma_P) \cdot \left( \prod_{i=1}^n l_{i,i} \right) \cdot \left( \prod_{i=1}^n u_{i,i} \right)$$

# How do we find the PLU decomposition?

**Input:**  $A \in \mathbb{R}^{n \times n}$

**Output:**  $L, U, P \in \mathbb{R}^{n \times n}$ , with  $PA = LU$ ,  $L$  unit lower triangular,  $U$  non-singular upper triangular, and  $P$  a permutation matrix

```
1:  $U \leftarrow A, L \leftarrow I, P \leftarrow I$ 
2: for  $k \leftarrow 1, \dots, n-1$  do                                ▷ Loop over columns
3:   Choose  $i \in \{k, \dots, n\}$  which maximises  $|u_{ik}|$ 
4:   Exchange row  $(u_{kk}, \dots, u_{kn})$  with  $(u_{ik}, \dots, u_{in})$     ▷ Col. 1 to  $k$  have zeros below the diagonal
5:   Exchange row  $(l_{k1}, \dots, l_{k,k-1})$  with  $(l_{i1}, \dots, l_{i,k-1})$     ▷  $L$  has unit diagonal, zeros above and
                                                below the diagonal in columns  $k+1$  to  $n$ 
6:   Exchange row  $(p_{k1}, \dots, p_{kn})$  with  $(p_{i1}, \dots, p_{in})$ 
7:   for  $j \leftarrow k+1, \dots, n$  do
8:      $l_{jk} \leftarrow u_{jk}/u_{kk}$                                 ▷  $u_{kk}$  now largest possible
9:      $(u_{jk}, \dots, u_{jn}) \leftarrow (u_{jk}, \dots, u_{jn}) - l_{jk}(u_{kk}, \dots, u_{kn})$ 
10:  end for
11: end for
```

This algorithm only works on invertible matrices (line 8 division).

# How do we find the PLU decomposition? (cont.)

$$A = \begin{pmatrix} 2 & 1 \\ -4 & 3 \end{pmatrix}$$

$$PA = LU$$

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ -4 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ -4 & 3 \end{pmatrix}$$



# How do we find the PLU decomposition? (cont.)

$$A = \begin{pmatrix} 2 & 1 \\ -4 & 3 \end{pmatrix}$$

$$PA = LU$$

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ -4 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ -4 & 3 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ -4 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} -4 & 3 \\ 2 & 1 \end{pmatrix}$$

# How do we find the PLU decomposition? (cont.)

$$A = \begin{pmatrix} 2 & 1 \\ -4 & 3 \end{pmatrix}$$

$$PA = LU$$

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ -4 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ -4 & 3 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ -4 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} -4 & 3 \\ 2 & 1 \end{pmatrix}$$

$$R_2 - (-\frac{1}{2})R_1.$$

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ -4 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -\frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} -4 & 3 \\ 0 & \frac{5}{2} \end{pmatrix}$$

# How do we find the PLU decomposition? (cont.)

So we have

$$L = \begin{pmatrix} 1 & 0 \\ -\frac{1}{2} & 1 \end{pmatrix}, U = \begin{pmatrix} -4 & 3 \\ 0 & \frac{5}{2} \end{pmatrix} \text{ and } \text{sgn}(\sigma_P) = -1.$$

$$\det(A) = \text{sgn}(\sigma_P) \cdot \left( \prod_{i=1}^n l_{i,i} \right) \cdot \left( \prod_{i=1}^n u_{i,i} \right)$$

# How do we find the PLU decomposition? (cont.)

So we have

$$L = \begin{pmatrix} 1 & 0 \\ -\frac{1}{2} & 1 \end{pmatrix}, U = \begin{pmatrix} -4 & 3 \\ 0 & \frac{5}{2} \end{pmatrix} \text{ and } \text{sgn}(\sigma_P) = -1.$$

$$\det(A) = \text{sgn}(\sigma_P) \cdot \left( \prod_{i=1}^n l_{i,i} \right) \cdot \left( \prod_{i=1}^n u_{i,i} \right)$$

Thus we have

$$\det(A) = -1 \cdot -4 \cdot \frac{5}{2} = 10.$$

# Runtime analysis

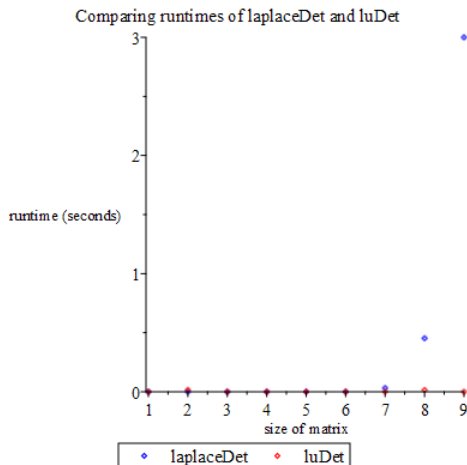
How quick is it?

- The PLU decomposition can be computed in  $\mathcal{O}(n^3)$  time.
- The determinants of the triangular matrices computed in  $\mathcal{O}(n)$  time.
- The parity of the permutation matrix in  $\mathcal{O}(n^2)$  time.

Therefore the total runtime for computing the determinant using the method is

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

# Laplace expansion vs LU decomposition



The difference between the exponential and polynomial-time function is clear.

# Limitations of LU decomposition

The main problem with the LU decomposition algorithm used is that it often requires division.

# Limitations of LU decomposition

The main problem with the LU decomposition algorithm used is that it often requires division.

- Division is not a ring operation, so it won't work on matrices over rings.



# Limitations of LU decomposition

The main problem with the LU decomposition algorithm used is that it often requires division.

- Division is not a ring operation, so it won't work on matrices over rings.
- Unless we compute exactly (difficult on a computer), precision may be lost.

# Limitations of LU decomposition

The main problem with the LU decomposition algorithm used is that it often requires division.

- Division is not a ring operation, so it won't work on matrices over rings.
- Unless we compute exactly (difficult on a computer), precision may be lost.

Let's try something else...

# Bird's algorithm

Define  $\mu : \mathbb{M}(n) \rightarrow \mathbb{M}(n)$ :

$$\mu(X) = \begin{pmatrix} \mu_{2,2} - x_{2,2} & x_{1,2} & \cdots & x_{1,n-1} & x_{1,n} \\ 0 & \mu_{3,3} - x_{3,3} & \cdots & x_{2,n-1} & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \mu_{n,n} - x_{n,n} & x_{n-1,n} \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}$$

# Bird's algorithm

Define  $\mu : \mathbb{M}(n) \rightarrow \mathbb{M}(n)$ :

$$\mu(X) = \begin{pmatrix} \mu_{2,2} - x_{2,2} & x_{1,2} & \cdots & x_{1,n-1} & x_{1,n} \\ 0 & \mu_{3,3} - x_{3,3} & \cdots & x_{2,n-1} & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \mu_{n,n} - x_{n,n} & x_{n-1,n} \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}$$

and  $F_A : \mathbb{M}(n) \rightarrow \mathbb{M}(n)$ , with  $A \in \mathbb{M}(n)$

$$F_A(X) = \mu(X) \cdot A$$

$$F_A^2(X) = \mu(F_A(X)) \cdot A$$

$$\vdots$$

$$F_A^n(X) = \mu(F_A^{n-1}(X)) \cdot A.$$

# Bird's algorithm (cont.)

## Bird's Theorem

$$F_A^{n-1}(A) = \begin{pmatrix} d & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 \end{pmatrix} \text{ with } d = \begin{cases} \det(A) & \text{odd } n \\ -\det(A) & \text{even } n. \end{cases}$$

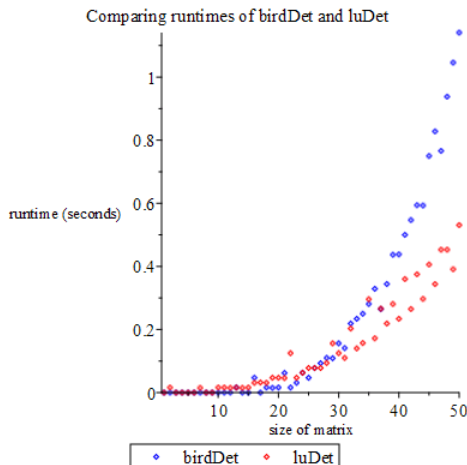
# Bird's algorithm (cont.)

## Bird's Theorem

$$F_A^{n-1}(A) = \begin{pmatrix} d & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 \end{pmatrix} \text{ with } d = \begin{cases} \det(A) & \text{odd } n \\ -\det(A) & \text{even } n. \end{cases}$$

- Enables the *division-free* computation of determinants in  $\mathcal{O}(n \cdot M(n))$  where  $M(n)$  is the runtime complexity of the matrix multiplication algorithm used.
- If the conventional  $\mathcal{O}(n^3)$  matrix multiplication algorithm is used, then Bird's algorithm will run in  $\mathcal{O}(n^4)$  time.
- But this can be reduced to  $\mathcal{O}(n^{3.8})$  by using a faster (e.g. *Strassen*) algorithm for matrix multiplication.

# Bird's algorithm vs LU decomposition



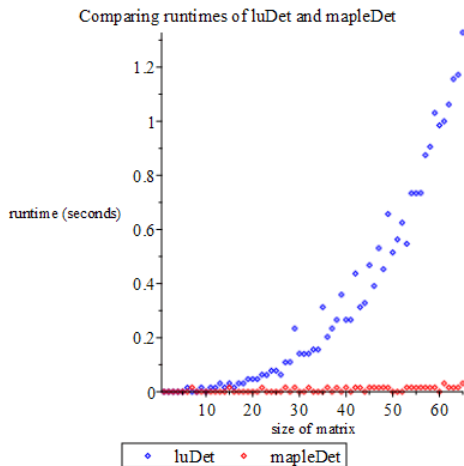
Bird's runtimes increase noticeably more rapidly than LU decomposition, but it's still polynomial.

## Summary of determinant algorithms

<i>Algorithm</i>	<i>Runtime</i>	<i>Exact?</i>
Leibniz formula	$\mathcal{O}(n \cdot n!)$	Yes
Laplace expansion	$\mathcal{O}(n!)$	Yes
LU decomposition	$\mathcal{O}(n^3)$	No
Bird's algorithm	$\mathcal{O}(n^{3.8})$	Yes



# How fast is Maple's built-in determinant function?



Very. Maple's optimisation means a fair comparison cannot be made.

Thanks!