Determinants

Group 12

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

University of Edinburgh

March 30, 2019

Overview

1. Using LU decomposition to compute determinants
   - What is LU decomposition?
   - Does it always work
   - How it helps us compute determinants
2. Other algorithms for computing determinants
   - Leibniz formula
   - Laplace expansion
   - Gaussian elimination
   - Bareiss algorithm
   - Bird's algorithm
3. Epilogue
   - Summary of determinant algorithms
   - How fast is Maple's implementation?

What is LU decomposition?

Somebody else do this section

- What LU decomposition is
- Its limitations
- How PLU decomposition addresses these limitations

Does the LU decomposition always work

Cases when the LU factorization exists.

- A PLU factorization will always exist for a square matrix A.
- If a square matrix A is invertible and all its leading principal minors are nonzero, then a LU factorization will exist.

Does the LU decomposition always work (cont.)

Cases where LU factorization will not always exist.

- If A is a singular matrix (i.e. $\det(A) = 0$), with $\operatorname{rank}(A) = k$, we can't know for sure if the LU factorization will exist.
  - If the first k leading principal minors are non-zero, the LU factorization exists.
  - For an $n \times n$ matrix A the LU factorization exists if

$$\operatorname{rank}(A_{1,1}) + n \geq \operatorname{rank}\begin{pmatrix} A_{1,1} & A_{1,2} \end{pmatrix} + \operatorname{rank}\begin{pmatrix} A_{1,1} \\ A_{2,1} \end{pmatrix}$$

Does LU decomposition always work (cont.)

Simple example of when LU decomposition fails

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

How it helps us compute determinants

We first start with $PA = LU$

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

since $P^{-1} = P^{T}$ by definition of an orthogonal matrix.

How it helps us compute determinants (cont.)

Now we have $A = P^T LU$. So in a form where we can easily calculate the determinant.

$$\begin{aligned} \det(A) &= \det(P^T) \cdot \det(L) \cdot \det(U) \\ &= \det(P) \cdot \det(L) \cdot \det(U) \end{aligned}$$

We can make this step as we know from algebra that, $\det(AB) = \det(A)\det(B)$ and also that $\det(A) = \det(A^T)$.

How it helps us compute determinants (cont.)

Finally we have

$$\det(A) = (-1)^s \left(\prod_{i=1}^{n} l_{i,i}\right)\left(\prod_{i=1}^{n} u_{i,i}\right)$$

with s being the number of row exchanges in the decomposition, which is the parity of the permutation matrix.

Leibniz formula

Definition

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

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

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

Computing the determinant using this method is slow with runtime $\mathcal{O}((N+1)!)$.

Laplace expansion

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

Theorem

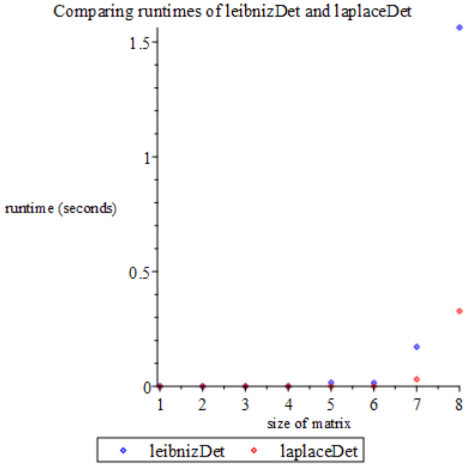
The formula for the (1st row) Laplace expansion of $A \in \mathbb{M}(n)$ is given as:

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

where $C_{i,j}$ is the (i,j) cofactor of A.

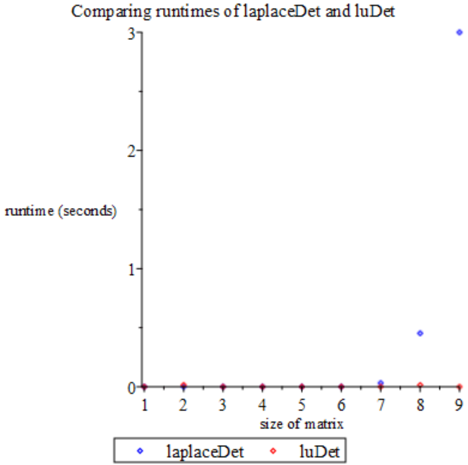
Its runtime complexity of $\mathcal{O}(N!)$ is poor.

Laplace expansion vs Leibniz formula



Runtimes are similar — both run in exponential time.

Laplace expansion vs LU decomposition



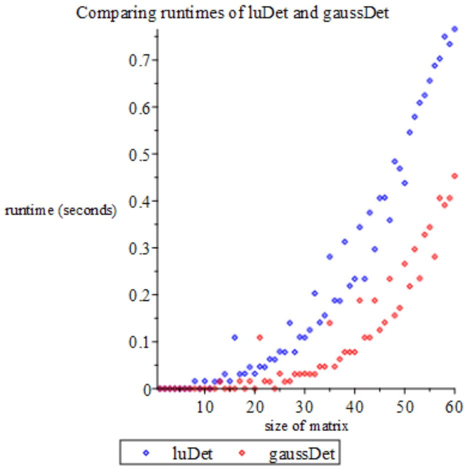
The difference between exponential and polynomial-time functions is clear.

Gaussian elimination

- The determinant of a triangular matrix can be computed by taking the product of its diagonal entries (which is a quick $\mathcal{O}(N)$ operation).
- Any invertible square matrix can be transformed into echelon form by performing Gaussian elimination, which takes $\mathcal{O}(N^3)$ time.

So how does it compare to LU decomposition?

Gaussian elimination vs LU decomposition



The difference in runtimes is small (a constant factor).

Gaussian elimination (cont.)

Conventional Gaussian elimination requires division, meaning that solutions maybe inexact, so precision is lost.
This is can be addressed by using...

Bareiss algorithm

- Addresses the issue of precision-loss by performing *integer-preserving* Gaussian elimination on integer matrices.
- The runtime complexity is $\mathcal{O}(N^3)$ which is the same as conventional Gaussian Elimination, whilst preserving exactness.

Bird's algorithm

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

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

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

$$\begin{aligned} F_A(X) &= \mu(X) \cdot A \\ F_A^2(X) &= \mu(F_A(X)) \cdot A \\ &\vdots \\ F_A^n(X) &= \mu(F_A^{n-1}(X)) \cdot A \end{aligned}$$

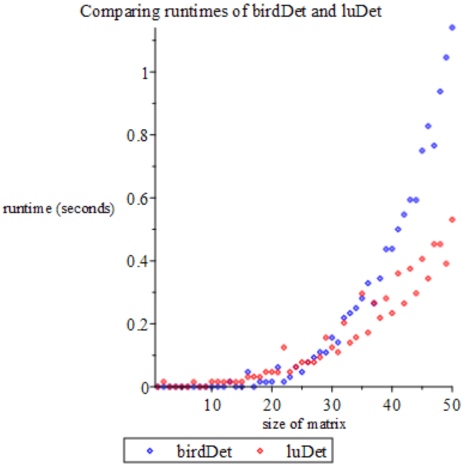
Bird's algorithm (cont.)

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 the *Strassen algorithm* for matrix multiplication.

Bird's algorithm vs LU decomposition

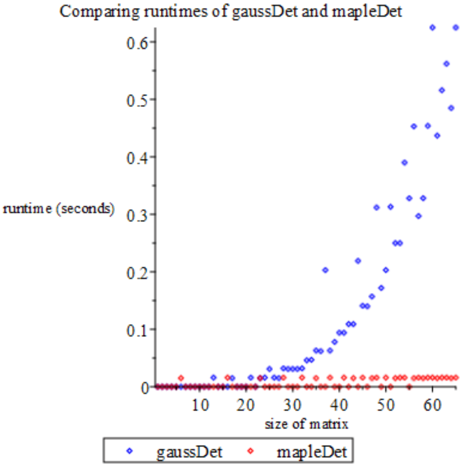


Bird's runtimes increase noticeably more rapidly than LU decomposition, but it's still polynomial.

Summary of determinant algorithms

Algorithm	*Runtime*	*Exact*
Leibniz formula	$\mathcal{O}((N+1)!)$	Yes
Laplace expansion	$\mathcal{O}(N!)$	Yes
LU decomposition	$\mathcal{O}(N^3)$	No
Gaussian elimination	$\mathcal{O}(N^3)$	No
Bareiss algorithm	$\mathcal{O}(N^3)$	Yes
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!