Applications and Analysis of LU Factorization

Contents

Applications of PLU Factorization

Solving a Square System

Multiplying $A\mathbf{x} = \mathbf{b}$ on the left by P we obtain

$$\underbrace{PA}_{=LU}\mathbf{x} = \underbrace{P\mathbf{b}}_{=:\boldsymbol{\beta}} \quad \longrightarrow \quad LU\mathbf{x} = \boldsymbol{\beta}\,,$$

which can be solved in two steps:

- Define $U\mathbf{x} = \mathbf{y}$ and solve for $\mathbf{y}$ in the equation

$$L\mathbf{y} = \boldsymbol{\beta}\,. \qquad \text{(forward elimination)}$$

- Having calculated $\mathbf{y}$, solve for $\mathbf{x}$ in the equation

$$U\mathbf{x} = \mathbf{y}\,. \qquad \text{(backward substitution)}$$

Solving a Square System (cont')

- Using the instructional codes (`backsub`, `forelim`, `myplu`):

```
[L,U,P] = myplu(A);
x = backsub( U, forelim(L, P*b) );
```

- Using MATLAB's built-in functions:

```
[L,U,P] = lu(A);
x = U \ (L \ (P*b));
```

  - The backslash is designed so that triangular systems are solved with the appropriate substitution.

- The most compact way:

```
x = A \ b;
```

  - The backslash does partial pivoting and triangular substitutions silently and automatically.

Computing Inverses

Observe that

$$(PA)^{-1} = (LU)^{-1} \longrightarrow A^{-1}P^{-1} = U^{-1}L^{-1} \longrightarrow LUA^{-1} = P$$

So solve $LU\mathbf{a}_i = \mathbf{p}_i$ with forward and backward substitution for each column $\mathbf{p}_i$ of P. Then

$$A^{-1} = \left[\begin{array}{c|c|c|c} & & & \\ \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \\ & & & \end{array}\right].$$

Computing Determinants

Observe that

$$\det(A) = \det(P^{-1}LU) = \det(P^{-1})\det(L)\det(U) = \frac{\det(L)\det(U)}{\det(P)}.$$

Useful facts.

- The determinant of a triangular matrix is the product of its diagonal entries. (What are diagonal entries of L?)
- P is a row permutation of the identity matrix (which has determinant 1), and each row swap negates the determinant. So if s is the number of row swaps, then $\det(P) = (-1)^s$.

It follows that

$$\det(A) = (-1)^s \prod_{i=1}^{n} u_{ii}.$$

Efficiency of PLU Factorization

Algorithm

Notation: Big-O and Asymptotic

Let f, g be positive functions defined on $\mathbb{N}$.

- $f(n) = O\left(g(n)\right)$ ("f is *big-O* of g") as $n \to \infty$ if

$$\frac{f(n)}{g(n)} \leqslant C, \quad \text{for all sufficiently large } n.$$

- $f(n) \sim g(n)$ ("f is *asymptotic* to g") as $n \to \infty$ if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1.$$

Timing Vector/Matrix Operations – FLOPS

- One way to measure the "efficiency" of a numerical algorithm is to count the number of floating-point arithmetic operations (FLOPS) necessary for its execution.
- The number is usually represented by $\sim cn^p$ where c and p are given explicitly.
- We are interested in this formula when n is large.

FLOPS for Major Operations

Vector/Matrix Operations

Let $x, y \in \mathbb{R}^n$ and $A, B \in \mathbb{R}^{n \times n}$. Then

- (vector-vector) $x^\mathrm{T} y$ requires $\sim 2n$ *flops*.
- (matrix-vector) Ax requires $\sim 2n^2$ *flops*.
- (matrix-matrix) AB requires $\sim 2n^3$ *flops*.

Cost of PLU Factorization

Note that we only need to count the number of *flops* required to zero out elements below the diagonal of each column.

- For each $i > j$, we replace R_i by $R_i + cR_j$ where $c = -a_{i,j}/a_{j,j}$. This requires approximately $2(n - j + 1)$ *flops*:
  - 1 division to form c
  - $n - j + 1$ multiplications to form cR_j
  - $n - j + 1$ additions to form $R_i + cR_j$
- Since $i \in \mathbb{N}[j + 1, n]$, the total number of *flops* needed to zero out all elements below the diagonal in the jth column is approximately $2(n - j + 1)(n - j)$.
- Summing up over $j \in \mathbb{N}[1, n - 1]$, we need about $(2/3)n^3$ *flops*:

$$\sum_{j=1}^{n-1} 2(n - j + 1)(n - j) \sim 2\sum_{j=1}^{n-1} (n - j)^2 = 2\sum_{j=1}^{n-1} j^2 \sim \frac{2}{3}n^3$$

Cost of Forward Elimination and Backward Substitution

Forward Elimination

- The calculation of $y_i = \beta_i - \sum_{j=1}^{i-1} \ell_{ij} y_j$ for $i > 1$ requires approximately $2i$ *flops*:
  - 1 subtraction
  - $i - 1$ multiplications
  - $i - 2$ additions
- Summing over all $i \in \mathbb{N}[2, n]$, we need about n^2 *flops*:

$$\sum_{i=2}^{n} 2i \sim 2\frac{n^2}{2} = n^2 \, .$$

Backward Substitution

- The cost of backward substitution is also approximately n^2 *flops*, which can be shown in the same manner.

Cost of G.E. with Partial Pivoting

Gaussian elimination with partial pivoting involves three steps:

- PLU factorization: $\sim (2/3)n^3$ *flops*
- Forward elimination: $\sim n^2$ *flops*
- Backward substitution: $\sim n^2$ *flops*

Summary

The total cost of Gaussian elimination with partial pivoting is approximately

$$\frac{2}{3}n^3 + n^2 + n^2 \sim \frac{2}{3}n^3$$

flops for large n.

Application: Solving Multiple Square Systems Simultaneously

To solve two systems $A\mathbf{x}_1 = \mathbf{b}_1$ and $A\mathbf{x}_2 = \mathbf{b}_2$.

Method 1.

- Use G.E. for both.
- It takes $\sim (4/3)n^3$ *flops*.

```
%% method 1
x1 = A \ b1;
x2 = A \ b2;
```

Method 2.

- Do it in two steps:
  1. Do PLU factorization $PA = LU$.
  2. Then solve $LU\mathbf{x}_1 = P\mathbf{b}_1$ and $LU\mathbf{x}_2 = P\mathbf{b}_2$.
- It takes $\sim (2/3)n^3$ *flops*.

```
%% method 2
[L, U, P] = lu(A);
x1 = U \ (L \ (P*b1));
x2 = U \ (L \ (P*b2));
```

```
%% compact implementation
X = A \ [b1, b2];
x1 = X(:, 1);
x2 = X(:, 2);
```