Lecture 12

Matrix Factorizations: LU and Cholesky

Owen L. Lewis

Department of Mathematics and Statistics
University of New Mexico

Sept. 26, 2024

Goals for today...

- Motivate matrix factorizations.
- Re-visit Forward Elimination
- LU factorization
- LDT factorization
- Cholesky factorization

Multiple Right Hand Sides

- Solve $Ax = b$ for many different b vectors
- For k different b vectors, Gaussian Elimination costs $\mathcal{O}(kn^3)$
- We can do better: factor the matrix beforehand

Multiple Right Hand Sides

- Solve $Ax = b$ for many different b vectors
- For k different b vectors, Gaussian Elimination costs $\mathcal{O}(kn^3)$
- We can do better: factor the matrix beforehand
  - Requires up front cost, but makes each individual solve cheap!

Motivation: Why factor at all?

- Suppose that we knew $A = LU$.
- What would it take to solve $Ax = b$?

- Triangular solves are cheaper than Gaussian elimination $(\mathcal{O}(n^2) vs \mathcal{O}(n^3))$!
- Cost of calculating L & U vs. cost of solving $Ax = b$: trade-offs.

Motivation: Why factor at all?

- Suppose that we knew $A = LU$.
- What would it take to solve $Ax = b$?
  - Replace $LUx = b$.

- Triangular solves are cheaper than Gaussian elimination $(\mathcal{O}(n^2) vs \mathcal{O}(n^3))$!
- Cost of calculating L & U vs. cost of solving $Ax = b$: trade-offs.

Motivation: Why factor at all?

- Suppose that we knew $A = LU$.
- What would it take to solve $Ax = b$?
  - Replace $LUx = b$.
  - "Multiply" by L^{-1} to get $Ux = L^{-1}b = c$.

- Triangular solves are cheaper than Gaussian elimination $(\mathcal{O}(n^2) vs \mathcal{O}(n^3))$!
- Cost of calculating L & U vs. cost of solving $Ax = b$: trade-offs.

Motivation: Why factor at all?

- Suppose that we knew $A = LU$.
- What would it take to solve $Ax = b$?
  - Replace $LUx = b$.
  - "Multiply" by L^{-1} to get $Ux = L^{-1}b = c$.
  - "Multiply" by U^{-1} to get $x = U^{-1}c$.

- Triangular solves are cheaper than Gaussian elimination $(\mathcal{O}(n^2) vs \mathcal{O}(n^3))$!
- Cost of calculating L & U vs. cost of solving $Ax = b$: trade-offs.

Motivation: Why factor at all?

- Suppose that we knew $A = LU$.
- What would it take to solve $Ax = b$?
  - Replace $LUx = b$.
  - “Multiply” by L^{-1} to get $Ux = L^{-1}b = c$.
  - “Multiply” by U^{-1} to get $x = U^{-1}c$.
- We don’t actually do any multiplication. Just do one lower triangular solve via forward substitution & one upper triangular solve via back substitution.
- Triangular solves are cheaper than Gaussian elimination ($\mathcal{O}(n^2) vs \mathcal{O}(n^3)$)!
- Cost of calculating L & U vs. cost of solving $Ax = b$: trade-offs.

Motivation: Can things get better?

- A is symmetric, if $A = A^T$
- If $A = LU$ and A is symmetric, then could $L = U^T$?
- If so, this could save 50% of the computation of LU by only calculating L
- Save 50% of the FLOPS!
- This is achievable: LDL^T and Cholesky ($L^T L$) factorization

Factorization Methods

Factorizations are the common approach to solving $Ax = b$: simply organized Gaussian elimination.

Goals for today:

- *LU* factorization
- Cholesky factorization

LU Factorization

Find L and U such that

$$A = LU$$

and L is lower triangular, and U is upper triangular.

$$L = \begin{bmatrix} 1 & 0 & \cdots & & 0 \\ \ell_{2,1} & 1 & 0 & & 0 \\ \ell_{3,1} & \ell_{3,2} & 1 & & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ \ell_{n,1} & \ell_{n,2} & \cdots & \ell_{n-1,n} & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ 0 & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & & & u_{n-1,n} \\ 0 & 0 & & & u_{n,n} \end{bmatrix}$$

Why?

- Since L and U are triangular, it is easy, $\mathcal{O}(n^2)$, to apply their inverses
- Decompose once, solve k right-hand sides quickly:
  - $\mathcal{O}(kn^3)$ with GE
  - $\mathcal{O}(n^3 + kn^2)$ with LU

LU Factorization

Listing 1: LU Solve

```
Factor A into L and U
Solve Ly = b for y              use forward substitution
Solve Ux = y for x              use backward substitution
```

Recall: Gaussian Elimination

- Eliminate elements under the pivot element in the first column.
- x' indicates a value that has been changed once.

$$\begin{bmatrix} \boxed{x} & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \longrightarrow \begin{bmatrix} \boxed{x} & x & x & x & x \\ 0 & x' & x' & x' & x' \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix}$$

$$\longrightarrow \begin{bmatrix} \boxed{x} & x & x & x & x \\ 0 & x' & x' & x' & x' \\ 0 & x' & x' & x' & x' \\ x & x & x & x & x \end{bmatrix}$$

$$\longrightarrow \begin{bmatrix} \boxed{x} & x & x & x & x \\ 0 & x' & x' & x' & x' \\ 0 & x' & x' & x' & x' \\ 0 & x' & x' & x' & x' \end{bmatrix}$$

Recall: Naive Gaussian Elimination

Listing 2: Forward Elimination

```
given A, b

for k = 1...n-1
   for i = k+1...n
      xmult = a_ik / a_kk
      a_ik = 0
      for j = k+1...n
         a_ij = a_ij - (xmult) a_kj
      end
      b_i = b_i - (xmult) b_k
   end
end
```

Notes

Elimination Matrices

- Another way to zero out entries in a column of A
- Annihilate entries below k^{th} element in a with matrix, M_k:

$$M_k a = \begin{bmatrix} 1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 1 & 0 & \dots & 0 \\ 0 & \dots & -m_{k+1} & 1 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & -m_n & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ a_{k+1} \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

where $m_i = a_i/a_k$, $i = k+1, \dots, n$.

- The divisor a_k is the "pivot" (and needs to be nonzero)
- Note, that m_i is the multiplier from GE

Elimination Matrices

- Matrix M_k is an “elementary elimination matrix”
  - Adds a multiple of row k to each subsequent row, with “multipliers” m_i
  - Result is zeros in the k^{th} column for rows $i > k$.
- M_k is unit lower triangular and nonsingular
- $M_k = I - m_k e_k^T$ where $m_k = [0, \ldots, 0, m_{k+1}, \ldots, m_n]^T$ and e_k is the k^{th} column of the identity matrix I.
- $M_k^{-1} = I + m_k e_k^T$, which means M_k^{-1} is also lower triangular, and we will denote $M_k^{-1} = L_k$.

Can you prove $M_k^{-1} = I + m_k e_k^T$?

Elimination Matrices

- Consider L_j and L_k, which are the inverses of the M matrices with $j > k$, then

$$\begin{aligned} L_k L_j &= I + m_k e_k^T + m_j e_j^T + m_k e_k^T m_j e_j^T \\ &= I + m_k e_k^T + m_j e_j^T + m_k (e_k^T m_j) e_j^T \\ &= I + m_k e_k^T + m_j e_j^T \end{aligned}$$

  because the k^{th} entry of vector m_j is zero (since $j > k$)
- Thus $L_k L_j$ is essentially a union of their columns.
- We don't need to do any multiplication, its just a record of all the multiples we've been calculating during GE.

Gaussian Elimination

- To reduce $Ax = b$ to upper triangular form, first construct M_1 with a_{11} as the pivot (eliminating the first column of A below the diagonal.)
- Then $M_1 Ax = M_1 b$ still has the same solution.
- Next construct M_2 with pivot a_{22} to eliminate the second column below the diagonal.
- Then $M_2 M_1 Ax = M_2 M_1 b$ still has the same solution
- $M_{n-1} \dots M_1 Ax = M_{n-1} \dots M_1 b$
- Let $M = M_n M_{n-1} \dots M_1$. Then $MAx = Mb$, with MA upper triangular.
- Do back substitution on $MAx = Mb$.

Another Way to Look at A

L and U?
Consider this

$$
\begin{aligned}
A &= A \\
A &= (M^{-1}M)A \\
A &= (M_1^{-1}M_2^{-1}\dots M_n^{-1})(M_nM_{n-1}\dots M_1)A \\
A &= (M_1^{-1}M_2^{-1}\dots M_n^{-1})((M_nM_{n-1}\dots M_1)A) \\
A &= \qquad\qquad L \qquad\qquad\qquad\qquad U
\end{aligned}
$$

But MA is upper triangular, and we've seen that $M_1^{-1}\dots M_n^{-1}$ is lower triangular. Thus, we have an algorithm that factors A into two matrices L and U.

LU Factorization

As an example take one column step of GE, A becomes

$$\begin{bmatrix} 6 & -2 & 2 \\ 12 & -8 & 6 \\ 3 & -13 & 9 \end{bmatrix} \to \begin{bmatrix} 6 & -2 & 2 \\ 0 & -4 & 2 \\ 0 & -12 & 8 \end{bmatrix}$$

using the elimination matrix

$$M_1 = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{bmatrix}$$

So we have performed

$$M_1 A x = M_1 b$$

LU Factorization

Summary:

- Inverting M_i is easy: just flip the sign of the lower triangular entries

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ -\frac{1}{2} & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \quad \Rightarrow \quad M_1^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}$$

- M_i^{-1} is just the multipliers used in Gaussian Elimination!
- $M_i^{-1} M_j^{-1}$ is still lower triangular, for $i < j$,
  and is the union of the columns
- $M_1^{-1} M_2^{-1} \ldots M_j^{-1}$ is lower triangular, with the lower triangle the multipliers from Gaussian Elimination

LU Factorization

Summary:

- Zeroing each column yields sequence of elimination matrix operations:

$$M_3 M_2 M_1 A x = M_3 M_2 M_1 b$$

- $M = M_3 M_2 M_1$. Thus
- $L = M_1^{-1} M_2^{-1} M_3^{-1}$ is lower triangular

$$\begin{aligned} MA &= U \\ M_3 M_2 M_1 A &= U \\ A &= M_1^{-1} M_2^{-1} M_3^{-1} U \\ A &= LU \end{aligned}$$

LU (forward elimination) Algorithm

Listing 3: *LU*

```
given A
initialize L and U
for k = 1...n-1
   ℓ_kk = 1
   for i = k+1...n
      xmult = a_ik / a_kk
      a_ik = 0
      ℓ_ik = xmult
      for j = k+1...n
         a_ij = a_ij - (xmult) a_kj
      end
   end
end
U = A
```

- There is a lot of wasted work here.
- L only has information below the pivot, A is set to zero below the pivot

LU (forward elimination) Algorithm

Listing 4: *LU*

```
given A
for k = 1...n-1
   for i = k+1...n
      xmult = a_ik / a_kk
      a_ik = xmult
      for j = k+1...n
         a_ij = a_ij - (xmult) a_kj
      end
   end
end
```

- *U* is stored in the upper triangular portion of *A*
- *L* (without the diagonal) is stored in the lower triangular

What About Pivoting?

- Pivoting (that is row exhanges) can be expressed in terms of matrix multiplication
- Do pivoting during elimination, but track row exchanges in order to express pivoting with matrix P
- Let P be all zeros
  - Place a 1 in column j of row 1 to exchange row 1 and row j
  - If no row exchanged needed, place a 1 in column 1 of row 1
  - *Repeat for all rows of P*
- P is a permutation matrix
- Now using pivoting,

$$LU = PA$$

MATLAB *LU*

Like GE, *LU* needs pivoting. With pivoting the *LU* factorization always exists, even if A is singular. With pivoting, we get

$$LU = PA$$

```
>> A=rand(4,4);
>> b=rand(4,1);
>> [L,U,P]=lu(A)
L = 1.0000         0         0         0
    0.9013    1.0000         0         0
    0.0298   -0.8982    1.0000         0
    0.7233    0.5813   -0.2670    1.0000
U = 0.7809    0.9890    0.4613    0.2971
         0   -0.8838   -0.0548    0.1857
         0         0    0.7183    0.6403
         0         0         0    0.2065
P =  0     1     0     0
     1     0     0     0
     0     0     0     1
     0     0     1     0
>> x=U\(L\(P*b))
x = 0.5326 0.5416 -1.2765 1.1315
>> A\b
    0.5326 0.5416 -1.2765 1.1315
```

Use SYMMETRY ! YRTEMMYS esU

- Suppose

$$A = LU, \quad \text{and} \quad A = A^T$$

- Then

$$LU = A = A^T = (LU)^T = U^T L^T$$

- Thus

$$U = L^{-1} U^T L^T$$

and

$$U(L^T)^{-1} = L^{-1} U^T = D$$

- We can conclude that

$$U = DL^T$$

and

$$A = LU = LDL^T$$

Notes

LDL^T Factorization

Listing 5: LDL^T Factorization

```
given A
output L, D

for j = 1 ... n
    ℓ_jj = 1

    d_j = a_jj − Σ_{ν=1}^{j−1} d_ν ℓ_{jν}^2

    for i = j+1 ... n
        ℓ_ji = 0
        ℓ_ij = (a_ij − Σ_{ν=1}^{j−1} ℓ_{iν} d_ν ℓ_{jν}) / d_j
    end
end
```

- Special form of the LU factorization

LL^T: Cholesky Factorization

- A must be symmetric and positive definite (SPD)
- A is Positive Definite (PD) if for all $x \neq 0$ the following holds

$$x^T A x > 0$$

- Positive definite gives us an all positive D in $A = LDL^T$
  - Let $x = L^{-T} e_i$, where e_i is the i-th column of I
  - Then, $x^T A x = d_i > 0$
- L becomes $LD^{1/2}$
- $A = LL^T$, i.e. $L = U^T$
  - Half as many flops as LU!
  - Only calculate L not U

Cholesky Factorization

Listing 6: Cholesky

```
given $A$
output $L$

for $k = 1 \dots n$
  $\ell_{kk} = \left(a_{kk} - \sum_{s=1}^{k-1} \ell_{ks}^2\right)^{1/2}$

  for $i = k+1 \dots n$
    $\ell_{ik} = \left(a_{ik} - \sum_{s=1}^{k-1} \ell_{is}\ell_{ks}\right) / \ell_{kk}$
  end
end
```

Why SPD?

In general, SPD gives us

- non singular
  - If $x^T A x > 0$, for all nonzero x
  - Then $Ax \neq 0$ for all nonzero x
  - Hence, the columns of A are linearly independent
- No pivoting
  - From algorithm, can derive that $|l_{kj}| \leqslant \sqrt{a_{kk}}$
  - Elements of L do not grow with respect to A
  - *For short proof see book*
- Cholesky faster than LU
  - No pivoting
  - Only calculate L, not U

Why SPD?

A matrix is Positive Definite (PD) if for all $x \neq 0$ the following holds

$$x^T Ax > 0$$

- For SPD matrices, use the Cholesky factorization, $A = LL^T$
- Cholesky Factorization
  - Requires no pivoting
  - Requires one half as many flops as LU factorization, that is only calculate L not L and U.
  - Cholesky will be more than *twice* as fast as LU because no pivoting means no data movement
- Use MATLAB's built-in `chol` function for routine work

Motivation Revisted

Multiple right hand sides

- Solve $Ax = b$ for k different b vectors
- Using LU factorization, the cost is $\mathcal{O}(n^3) + \mathcal{O}(kn^2)$
- Using Gaussian Elimination, the cost is $\mathcal{O}(kn^3)$

If A is symmetric

- Save 50% of the flops with LDL^T factorization
- Save 50% of the flops and many memory operations with Cholesky (L^TL) factorization

See `time_LU_vs_Cholesky.m`