Sparse Matrix Library

Xiaobai Jiang
Yuwei An
Shuting Zhang

Matrix Decomposition

- LU Decomposition
- QR Decomposition
- Cholesky Decomposition

Static Public Member Functions

static void	**LU** (const **BaseMatrix** &A, **BaseMatrix** &L, **BaseMatrix** &U) Performs LU decomposition on matrix A such that A = L * U. More...
static void	**QR** (const **BaseMatrix** &A, **BaseMatrix** &Q, **BaseMatrix** &R) Performs QR decomposition using the Gram-Schmidt process. More...
static void	**Cholesky** (const **BaseMatrix** &A, **BaseMatrix** &L) Performs Cholesky decomposition on matrix A such that A = L * L^T. More...
static std::vector< double >	**solveLU** (const **BaseMatrix** &L, const **BaseMatrix** &U, const std::vector< double > &b) Solves Ax = b using LU decomposition (A = L * U). More...

What is Matrix Decomposition?

- **Definition**: the process of breaking a matrix into a product of simpler matrices, which makes certain matrix computations more efficient
- **Applications**
  - ✓ Solving linear systems
  - ✓ Eigenvalue problems
  - ✓ …

LU Decomposition: A = LU

- **Goal**: decomposes a matrix A into two matrices
  - L (*lower* triangular matrix)
  - U (*upper* triangular matrix)
- **Assumption:** A is square and non-singular
- **Steps**
  - Upper triangular matrix: $U_{ik} = A_{ik} - \sum_{j=0}^{i-1} L_{ij} U_{jk}$
  - Lower triangular matrix: $L_{ki} = \frac{1}{U_{ii}} \left(A_{ki} - \sum_{j=0}^{i-1} L_{kj} U_{ij} \right)$
  - Diagonal of L : Set $L_{ii} = 1$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

LU Decomposition

- **Pros**: Simple; easy to use in $Ax = b$ solving in $O(n^2)$ time
  - Convert into $LUx = b$
  - Forward substitution: solve $Ly = b$
  - Backward substitution: solve $Ux = y$
- **Cons**: Requires pivoting for numerical stability
- **Testcase**

```
matrix_size = 5;
BaseMatrix* lu_matrix = mg.generate_spd_matrix("COO", matrix_size);

BaseMatrix* L = mg.generate_matrix("COO", matrix_size, matrix_size, 0);
BaseMatrix* U = mg.generate_matrix("COO", matrix_size, matrix_size, 0);

Decomposition::LU(*lu_matrix, *L, *U);
```

```
[Generated Random Sparse Matrix in COO Format for LU Decomposition]
Row Indices: 0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4
Column Indices: 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4
Values: 2.47198 1.7375 1.27811 2.2062 1.64348 1.7375 1.78453 0.951362 1.832
74 1.63353 1.27811 0.951362 0.934324 1.12523 1.04285 2.2062 1.83274 1.12523
 2.40383 1.96611 1.64348 1.63353 1.04285 1.96611 1.87271
[L Matrix]
Matrix:
1 0 0 0 0
0.702878 1 0 0 0
0.517041 0.0940992 1 0 0
0.892484 0.500725 -0.156454 1 0
0.664842 0.849255 0.551538 0.985871 1
[U Matrix]
Matrix:
2.47198 1.7375 1.27811 2.2062 1.64348
0 0.563277 0.0530039 0.282047 0.478366
0 0 0.268499 -0.0420078 0.148088
0 0 0 0.287032 0.282977
0 0 0 0 0.0131537
[Frobenius Norm of Difference (L * U - Original Matrix)]
Frobenius Norm: 2.22045e-16
LU Decomposition Time: 5139 ns
```

QR Decomposition: A = QR

- **Goal**: decomposes a matrix A into two matrices
  - *Orthogonal* matrix Q $(QQ^T = Q^TQ = I)$
  - *upper triangular* matrix R
- **Steps (using Gram-Schmidt process)**
  - Orthogonalize
  - Construct $R = Q^TA$

$$\overset{\mathbf{A}}{\begin{bmatrix} | & | & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \\ | & | & | \end{bmatrix}} = \underbrace{\overset{\mathbf{Q}}{\begin{bmatrix} | & | & | \\ \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ | & | & | \end{bmatrix}}}_{\text{orthogonal unit vector}} \underbrace{\overset{\mathbf{R}}{\begin{bmatrix} \mathbf{e}_1^T\cdot\mathbf{a}_1 & \mathbf{e}_1^T\cdot\mathbf{a}_2 & \mathbf{e}_1^T\cdot\mathbf{a}_3 \\ 0 & \mathbf{e}_2^T\cdot\mathbf{a}_2 & \mathbf{e}_2^T\cdot\mathbf{a}_3 \\ 0 & 0 & \mathbf{e}_3^T\cdot\mathbf{a}_3 \end{bmatrix}}}_{\text{Upper Diagonal matrix}}$$

$$q_1 = \frac{a_1}{\|a_1\|}$$

1. First vector

$$r_{ij} = q_i^\top a_j \quad \text{for } i < j$$

$$\tilde{a}_j = a_j - \sum_{i=1}^{j-1} r_{ij} q_i$$

2. Remove projections

$$q_j = \frac{\tilde{a}_j}{\|\tilde{a}_j\|}$$

3. Normalize

QR Decomposition

- **Pros**: Numerically more stable than LU for least squares
- **Cons**: Produces dense matrices even from sparse input
- **Testcase**

```
BaseMatrix* qr_matrix = mg.generate_spd_matrix("CSR", matrix_size);

BaseMatrix* Q_mat = mg.generate_matrix("CSR", matrix_size, matrix_size, 0);
BaseMatrix* R = mg.generate_matrix("CSR", matrix_size, matrix_size, 0);  //

Decomposition::QR(*qr_matrix, *Q_mat, *R);
```

```
[Generated Random Sparse Matrix in CSR Format for QR Decomposition]
Row Pointers: 0 5 10 15 20 25
Column Indices: 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4
Values: 2.47198 1.7375 1.27811 2.2062 1.64348 1.7375 1.78453 0.95136
2 1.83274 1.63353 1.27811 0.951362 0.934324 1.12523 1.04285 2.2062 1
.83274 1.12523 2.40383 1.96611 1.64348 1.63353 1.04285 1.96611 1.872
71
[Q Matrix]
Matrix:
0.577359 -0.604661 -0.20485 -0.255721 0.440097
0.405813 0.603861 -0.1904 -0.638756 -0.162499
0.298518 -0.215819 0.82557 -0.159569 -0.396586
0.515284 -0.0244722 -0.352484 0.550231 -0.553971
0.383852 0.471763 0.340548 0.445399 0.561911
[R Matrix]
Matrix:
4.28153 3.58276 2.38303 4.34677 3.65504
0 0.547475 0.0644601 0.398583 0.602976
0 0 0.286903 -0.0496943 0.157978
0 0 0 0.283972 0.285818
0 0 0 0 0.00739119
Frobenius Norm of Difference (Q * R - Original Matrix)
Frobenius Norm: 3.14018e-16
QR Decomposition Time: 4707 ns
```

Cholesky Decomposition: $A = LL^T$

- **Goal**: decomposes a matrix A into the product $A = LL^T$
  - where L is a lower triangular matrix
- **Assumption**: A is a symmetric, positive-definite matrix
- **Steps** $x^T Ax > 0$ for all nonzero $x \in \mathbb{R}^n$
  - Diagonal entries (for each row)

$$L_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} L_{ik}^2}$$

  - Off-diagonal entries (compute elements below L_{ii})

$$L_{ij} = \frac{1}{L_{jj}}\left(A_{ij} - \sum_{k=1}^{j-1} L_{ik}L_{jk}\right), \text{ for } i > j$$

$$\begin{bmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{00} & 0 & 0 \\ L_{10} & L_{11} & 0 \\ L_{20} & L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} L_{00} & L_{10} & L_{20} \\ 0 & L_{11} & L_{21} \\ 0 & 0 & L_{22} \end{bmatrix}$$

Cholesky Decomposition

- **Pros**: Fast and memory efficient, numerically stable without pivoting
- **Cons**: Only applies to symmetric, positive-definite matrices
- **Testcase**

```
BaseMatrix* cholesky_matrix = mg.generate_spd_matrix("CSC", matrix_size);
BaseMatrix* chol_L = mg.generate_matrix("CSC", matrix_size, matrix_size, 0);
Decomposition::Cholesky(*cholesky_matrix, *chol_L);
```

```
[Generated Random Sparse Matrix in CSC Format for Cholesky Decomposition]
Column Pointers: 0 5 10 15 20 25
Row Indices: 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4
Values: 2.47198 1.7375 1.27811 2.2062 1.64348 1.7375 1.78453 0.951362 1.8327
4 1.63353 1.27811 0.951362 0.934324 1.12523 1.04285 2.2062 1.83274 1.12523 2
.40383 1.96611 1.64348 1.63353 1.04285 1.96611 1.87271
[Cholesky L Matrix]
Matrix:
1.57225 0 0 0 0
1.1051 0.750518 0 0 0
0.812919 0.0706231 0.518169 0 0
1.40321 0.375803 -0.0810695 0.535754 0
1.0453 0.637381 0.28579 0.528184 0.114689
[Frobenius Norm of Difference (L * L^T - Original Matrix)]
Frobenius Norm: 0
Cholesky Decomposition Time: 2399 ns
```