Sparse Matrix Library: Input and Output Report

Xiaobai Jiang, Yuwei An, Shuting Zhang

Overview

This report documents the inputs and outputs of our sparse matrix library, which supports matrix generation, decomposition (LU, QR, Cholesky), and eigenvalue solving (Power, Inverse, Lanczos, Arnoldi, QR iterations). We include both console outputs and explanation of results from `test.cpp` and `test_svd.cpp`. The outputs for Matrix Generation, EigenSolver and Matrix Decomposition are by running `make test` on `test.cpp`, and the output from SVD is by running `make test_svd` on `test_svd.cpp`.

1. Matrix Generation

Matrix generation is implemented using the `MatrixGenerator` class. It supports:

- Generating random sparse matrices in COO, CSR, or CSC formats;
- Creating symmetric positive definite (SPD) matrices for eigenvalue and decomposition testing.

The key APIs used are:

- `generate_matrix(format, m, n, density)`: Generate sparse matrix with given dimensions and density.
- `generate_spd_matrix(format, n)`: Generate an SPD matrix of size $n \times n$.

Random Sparse Matrix

A random matrix of size 10 and density 3 in COO format is generated by running `make test` on test.cpp.

```
[Generated Random Sparse Matrix]
Row Indices: 0 7 9 2 4 1 4 0 8 9 4 5
Column Indices: 6 8 4 7 0 1 8 5 8 1 1 7
Values: 0.967335 0.808437 0.591621 0.498546 0.688078 0.928566 0.144788
    ↪ 0.380714 0.739723 0.42049 0.278562 0.921385
Matrix:
0 0 0 0 0 0.380714 0.967335 0 0 0
0 0.928566 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0.498546 0 0
0 0 0 0 0 0 0 0 0 0
0.688078 0.278562 0 0 0 0 0 0 0.144788 0
0 0 0 0 0 0 0 0.921385 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0.808437 0
```

```
0 0 0 0 0 0 0 0 0.739723 0
0 0.42049 0 0 0.591621 0 0 0 0 0
```

Symmetric Positive Definite (SPD) Matrix

Used as input for eigenvalue and decomposition algorithms.

2. EigenSolver Outputs

We evaluate eigenvalue solvers implemented in the `EigenSolver` class. All methods assume square matrices; symmetric matrix is required for Lanczos Iteration and invertible matrix is required for Inverse Iteration. The SPD matrix generated is used as input for fair comparison.

The output is generated by running `make test` on `test.cpp` with SPD matrix of size 50 in COO format. The number of eigenvalues Lanczos Iteration and Arnoldi Iteration solved is set to be 10. All methods use a max iteration limit of 50 and a tolerance of 10^{-6}.

- **Power Iteration**: estimates the dominant eigenvalue using iterative multiplication and normalization.
- **Inverse Iteration**: estimates the smallest (or nearest-to-zero) eigenvalue using LU decomposition and back-substitution.
- **QR Iteration**: computes all eigenvalues via iterative QR decomposition with similarity transforms.
- **Lanczos Iteration**: efficiently approximates eigenvalues of large symmetric matrices using Krylov subspace projections.
- **Arnoldi Iteration**: generalizes Lanczos for non-symmetric matrices, using orthonormal bases and Hessenberg projections.

Power Iteration

```
[Testing Power Iteration]
Finish iteration in iter 6
Estimated dominant eigenvalue (Power Iteration): 792.356
Time taken: 0 ms
```

Inverse Iteration

```
[Testing Inverse Iteration]
Finish iteration in iter 3
Estimated smallest eigenvalue (Inverse Iteration): 5.25056e-08
Time taken: 145 ms
```

QR Iteration

```
[Testing QR Iteration]
[QR finished]
Time taken: 19478 ms
[QR eigenvalues (first 10)]
792.356 12.8577 11.9656 10.7768 9.66794 8.73318 8.41568 7.88438 7.15266
    ↪ 7.14658
```

Lanczos Iteration

```
[Testing Lanczos Iteration]
[Lanczos finished]
Time taken: 13 ms
[Lanczos eigenvalues (first 10)]
792.356 792.355 12.778 11.4522 9.91066 7.30989 4.48347 2.7544 1.04209
    ↪ 0.133806
```

Arnoldi Iteration

```
[Testing Arnoldi Iteration]
[Arnoldi finished]
Time taken: 13 ms
[Arnoldi eigenvalues (first 10)]
792.356 12.8503 11.6539 10.3237 8.81115 6.48019 3.72511 2.11372 0.924672
    ↪ 0.123859
```

3. Matrix Decomposition Outputs

Decompositions are handled by the `Decomposition` class, which supports LU, QR, and Cholesky factorizations. Each method assumes a square matrix as input. The correctness is verified by reconstructing the original matrix and computing Frobenius norm of the difference.

All decomposition tests are generated by running `make test` (in `test.cpp`) on a 10x10 SPD matrix generated in COO, CSR, or CSC format depending on algorithm:

- LU uses a COO-formatted matrix
- QR uses a CSR-formatted matrix
- Cholesky uses a CSC-formatted matrix
- **LU Decomposition**: Factorizes $A = LU$ where L is lower triangular and U is upper triangular.
- **QR Decomposition**: Factorizes $A = QR$ using Gram-Schmidt process; Q is orthogonal, R is upper triangular.
- **Cholesky Decomposition**: For SPD matrices, decomposes $A = LL^T$, where L is lower triangular.

LU Decomposition

```
[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 3 3 3 3 4 4 4 4
Column Indices: 0 1 2 3 4 0 1 2 3 4 0 1 2 1 0 2 3 4 0 1 2 3 4
Values: 2.47198 1.7375 1.27811 2.2062 1.64348 1.7375 0.934324 1.78453
   ↪ 0.951362 1.83274 1.63353 1.27811 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 - A): 2.23217e-10
Time: 3290800 ns
```

QR Decomposition

```
[Generated Random Sparse Matrix in COO Format for QR Decomposition]
Row Indices: 0 0 0 0 0 1 1 1 1 1 2 2 2 3 3 3 3 4 4 4 4
Column Indices: 0 1 2 3 4 0 1 2 3 4 0 1 2 1 0 2 3 4 0 1 2 3 4
Values: 2.47198 1.7375 1.27811 2.2062 1.64348 1.7375 0.934324 1.78453
   ↪ 0.951362 1.83274 1.63353 1.27811 1.12523 1.04285 2.2062 1.83274
   ↪ 1.12523 2.40383 1.96611 1.64348 1.63353 1.04285 1.96611 1.87271
[Q Matrix]
Matrix:
-0.494796 -0.550489 0.183783 -0.277943 -0.573517
-0.347158 0.0248482 0.777477 0.497223 -0.198363
-0.255381 0.133379 -0.569472 0.751688 -0.166946
-0.443919 0.558356 0.117178 -0.246871 0.640228
-0.329864 0.606771 -0.139634 -0.187391 -0.682347
[R Matrix]
Matrix:
-4.99616 -4.27827 -3.98774 -3.91689 -3.89679
0 -1.67963 -0.068301 0.490712 -0.222728
0 0 -1.04938 0.469145 -0.0163359
0 0 0 -0.207865 -0.54845
0 0 0 0 0.128795
Frobenius Norm of Difference (Q * R - A): 1.02123e-9
Time: 4781300 ns
```

Cholesky Decomposition

```
[Generated Random Sparse Matrix in COO Format for Cholesky Decomposition]
Row Indices: 0 0 0 0 0 1 1 1 1 1 2 2 2 3 3 3 3 4 4 4 4
Column Indices: 0 1 2 3 4 0 1 2 3 4 0 1 2 1 0 2 3 4 0 1 2 3 4
Values: 2.47198 1.7375 1.27811 2.2062 1.64348 1.7375 0.934324 1.78453
   ↪ 0.951362 1.83274 1.63353 1.27811 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:
1.57229 0 0 0 0
1.10512 0.697368 0 0 0
0.813589 0.124317 0.789182 0 0
1.40318 0.798369 -0.249767 0.455921 0 0
1.04528 1.2187 0.868742 1.90044 0.25595
Frobenius Norm of Difference (L * L^T - A): 8.10232e-11
Time: 2119000 ns
```

4. SVD Output

SVD computation uses `SVD::lanczos_bidiag`, which approximates the largest singular value using Lanczos Bidiagonalization. This is useful for applications like PCA and connectivity estimation in networks.

The evaluation is conducted by running `make test_svd`, which invokes `test_svd.cpp` with a 100x100 COO matrix (default density = 32). Lanczos uses $k = 3$ bidiagonal steps. For comparison, LAPACK's `dgesvd_()` is also applied to extract full SVD.

Given matrix A, the method computes bidiagonal matrix B such that:

$$A \approx UBV^T$$

The top singular value is approximated from B's spectrum.

test_svd.cpp

```
============================
Sparse Matrix SVD Testing
============================
============================
Lanczos Bidiagonalization
============================

Estimated Largest singular value: 16.6819
Time taken: 0.089 ms
Time taken: 2.852 ms
Largest singular value: 16.4561
```

5. Conclusion

The library achieves both functional correctness and performance across all components. Outputs from decomposition and eigenvalue methods were validated by reconstructing the input matrix or comparing numerical results.

Summary of Observations:

- **Power Iteration** quickly converges (6 iterations) to the dominant eigenvalue, with negligible time cost (0 ms).
- **Inverse Iteration** estimates the smallest eigenvalue with high precision, though it incurs higher runtime (145 ms) due to LU solving.
- **Lanczos and Arnoldi** both yield top-k eigenvalues efficiently (13 ms), with results closely matching those from full QR iteration.
- **QR Iteration** gives complete spectrum but is slowest (19,478 ms), due to full matrix transformation at each step.
- **LU, QR, and Cholesky Decompositions** all achieve Frobenius norm reconstruction errors below 10^{-9}, verifying numerical stability. Cholesky was the fastest, benefitting from SPD input.
- **SVD** estimated the largest singular value accurately using Lanczos bidiagonalization, matching LAPACK reference within margin.

The matrix generation module consistently produced structurally valid inputs across COO, CSR, and CSC formats, supporting reliable benchmarking of all algorithms.