Homework 1

Zaiwen Wen
Beijing International Center for Mathematical Research
Peking University

Sep 17, 2025

1 Submission Requirement

1. Prepare a report including
   - detailed answers to each question
   - numerical results and their interpretation
2. The programming language can be either Python or C/C++.
3. Pack all of your codes named as `hw1-studentID-name.zip` and send it to TA: `TBD`.

   The homework should be packaged into a compressed file with the naming format `hw1-studentID-name`. The file type is arbitrary. Do not include spaces in the file name, and it is preferable not to use Chinese characters.
4. Do not paste large amounts of code directly into the report. For actual results, use tables or plots instead of screenshots or direct copies from the command line.
5. If you submit Word documents, please provide the original Word file and also convert it to a PDF file.
6. If you get significant help from others on one routine, write down the source of references at the beginning of this routine.

2 Problems

Notice:

- Complete at least four problems in total: choose any two from A1, A2, and A3; and any two from A4, A5, and A6.
- For sparse matrices, select test matrices from Matrix Market. Examples include the *Cylshell* series (eg. `S1RMQ4M1`), `BCSSTK13`, and `BCSSTM13`.

A1. Dense Matrix-Vector Multiplication

- **Matrix Generation:** Generate an 1024×1024 dense matrix A with $A_{ij} \sim \mathcal{N}(0,1)$ and a vector x of length 1024 with $x_i \sim \mathcal{N}(0,1)$.
- **Requirements:**
  - Implement $y = Ax$ using both NumPy (CPU) and CuPy/ PyTorch (GPU)
  - Compare execution times for different matrix sizes ($n = 128, 1024, 8192$). Plot the GPU speedup relative to the CPU of matrix size.
    Hint: Compute the average over multiple runs.

A2. Sparse Matrix-Vector Multiplication

- **Matrix Generation:** Select a test sparse matrix A and a sparse vector x from Matrix Market.

- **Requirements:**
  - Implement $y = Ax$ using both SciPy (CPU) and CuPy/PyTorch (GPU) sparse libraries
  - Compare performance across different sparse storage formats (such as CSR, CSC).
  - Analyze how sparsity affects speedup ratio.

A3. QR Decomposition (Orthogonalization)

- **Matrix Generation:** Generate a "tall and skinny" matrix A of size 512×128, with elements from $\mathcal{N}(0, 1)$.
- **Requirements:**
  - Compute thin QR decomposition using `numpy.linalg.qr`.
  - Test the numerical stability.
    Hint: Test ill-conditioned matrices.

A4. Eigenvalue Decomposition of Dense Matrices

- **Matrix Generation:** Generate a random symmetric matrix

$$A = \frac{B + B^\top}{2}, \quad B \sim \mathcal{N}(0, I_{256}).$$

- **Requirements:**
  - Compute eigenvalues and eigenvectors using `numpy.linalg.eigh` (CPU) and `torch.linalg.eigh` (GPU). Verify the decomposition results.
  - Compare the numerical stability of the two methods.
    Hint: One approach is to generate a low-rank positive definite matrix and check whether the computed eigenvalues include any significantly negative values.
  - Compare the time usage of the CPU and GPU implementations for different matrix sizes.

A5. Partial Eigenvalue Decomposition of Sparse Matrices

- **Matrix Generation:** Select a test sparse matrix A from Matrix Market.
- **Requirements:**
  - Compute top 50 largest eigenvalues using `scipy.sparse.linalg.eigsh` (CPU) and `cupyx.scipy.sparse.linalg.eigsh` (GPU).
  - Compare the time usage of the CPU and GPU implementations for different matrix sizes.
  - Compare results with dense eigenvalue decomposition methods. Analyze differences in computation time and memory usage.

A6. Solving Linear Systems

- **Matrix Generation:** Generate an 512×512 dense matrix A with $A_{ij} \sim \mathcal{N}(0, 1)$ and a vector x of length 512 with $x_i \sim \mathcal{N}(0, 1)$.
- **Requirements:**
  - Solve the system $Ax = b$ using (1) direct linear algebra functions (e.g., `numpy.linalg.solve`), and (2) LU/QR/Cholesky decomposition, with support for execution on either CPU or GPU.
  - Compare the numerical stability of the above methods.
    Hint: Generate A by creating a low-rank matrix plus a small identity matrix.
  - Test with sparse A and b, and compare the computational time with corresponding dense matrix methods.