CholeskyQR with Randomization and Pivoting for Tall Matrices (CQRRPT)

Abhinav Malik (MDS202401)

Abhishek Lunagariya (MDS202402)

Abhishek Singh (MDS202403)

Introduction to QR Factorization

- **QR factorization** is one of the fundamental matrix decomposition in numerical linear algebra.

- For $M \in \mathbb{R}^{m \times n}$ $(m \geq n)$, it decomposes as:

$$M = QR$$

  where:
  - Q has orthonormal columns $(Q^T Q = I)$
  - R is an upper-triangular matrix

- **Applications of QR Factorization**
  - Solving linear least squares problems
  - Block orthogonalization in iterative methods
  - Randomized low-rank approximation algorithms

QR with Column Pivoting (QRCP)

- **What is QRCP?**
  - QRCP introduces a permutation Matrix **P**. We compute $\mathbf{MP} = \mathbf{QR}$.
  - It is crucial when **M** is rank deficit as it reveals numerical rank of matrix and improves stability.

- **Why is QRCP Expensive?**
  - Classical QRCP (e.g., LAPACK's GEQP3) costs $\sim 4mn^2$ flops.
  - In contrast, unpivoted QR costs only $\sim 2mn^2$ flops.
  - QRCP involves memory-bound Level 2 BLAS operations and repeated column norm updates as it continuously tracks which column has the largest residual norm

- **CholeskyQR with Randomization and Pivoting for Tall Matrices (CQRRPT)**
  - It first compresses the input matrix using randomized sketching, selects informative columns via QRCP on the sketch, and then applies CholeskyQR to a preconditioned version of the original matrix guided by the sketch.

- **Our motivation to perform CQRRPT**
  - Achieves performance close to unpivoted QR while preserving rank-revealing properties.
  - Scalable, communication-efficient, and numerically stable.
  - Enables high-performance factorization for very large or tall matrices.

What is Sketching?

- **Definition:** Sketching is a technique to compress a large matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ into a smaller one $\mathbf{M}^{\text{sk}} = \mathbf{SM}$, where $\mathbf{S} \in \mathbb{R}^{d \times m}$ and $d \ll m$.
  It enables efficient computation by replacing large matrix operations with smaller, approximate ones.
- **Purpose:**
  - Dimensionality reduction for efficiency in downstream computations.
  - Accelerates pivot selection and enables effective preconditioning.
  - It retains properties such as column norms and subspace orientation.
- **Core Principle:** A good sketching operator $\mathbf{S}$ preserves the row space and norm relationships of $\mathbf{M}$ with high probability (e.g., subspace embedding).
  It ensures that computations on the sketch approximate the behavior of the original matrix.
- **Usage in Numerical Linear Algebra:**
  - Approximating matrix decompositions (e.g., QR, SVD).
  - Reducing computation in large-scale problems.

Random Sketching – How It Works

- Random Sketching adds a probabilistic layer. We multiply matrix $\mathbf{M}$ by a sketching matrix $\mathbf{S}$ drawn from a random distribution.
- Projects a tall matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ to a smaller matrix $\mathbf{M}^{\text{sk}} = \mathbf{SM}$, where $\mathbf{S} \in \mathbb{R}^{d \times m}$ and $d \ll m$.
- Randomness ensures that important directions in the data are preserved with high probability.
- The goal is to obtain a small matrix $\mathbf{M}^{\text{sk}}$ that maintains the rank and column space of $\mathbf{M}$.
- It enables fast QRCP and rank estimation without directly processing all m rows.
- **Outcome:**
  - The sketch $\mathbf{M}^{\text{sk}}$ is used to guide pivoting and preconditioning.
  - Guarantees both computational efficiency and theoretical stability.

CholeskyQR and How Is It Done?

- **Definition:** CholeskyQR is a fast algorithm for computing the QR factorization of a full-rank matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$, where $m \geq n$.

- **Steps:**
  - Compute the Gram matrix: $\mathbf{G} = \mathbf{M}^* \mathbf{M} \in \mathbb{R}^{n \times n}$.
  - Perform Cholesky decomposition: $\mathbf{G} = \mathbf{R}^* \mathbf{R}$.
  - Recover the orthonormal matrix: $\mathbf{Q} = \mathbf{M}\mathbf{R}^{-1}$.

- **Output:** The matrix $\mathbf{M} = \mathbf{Q}\mathbf{R}$, where:
  - $\mathbf{Q} \in \mathbb{R}^{m \times n}$ has orthonormal columns.
  - $\mathbf{R} \in \mathbb{R}^{n \times n}$ is upper triangular.

- **Flop Count:**
  - Forming $\mathbf{G} = \mathbf{M}^* \mathbf{M}$: mn^2 flops.
  - Cholesky factorization: $\frac{1}{3}n^3$ flops.
  - Solving $\mathbf{Q} = \mathbf{M}\mathbf{R}^{-1}$: mn^2 flops.
  - **Total:** $\approx 2mn^2 + \frac{1}{3}n^3$, efficient for $m \gg n$.

CholeskyQR: Compute $\mathbf{G} = \mathbf{M}^*\mathbf{M}$ and $\mathbf{R}$

- Original matrix $\mathbf{M} \in \mathbb{R}^{3\times 2}$:

$$\mathbf{M} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

- Transpose $\mathbf{M}^* \in \mathbb{R}^{2\times 3}$:

$$\mathbf{M}^* = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

- Compute Gram matrix:

$$\mathbf{G} = \mathbf{M}^*\mathbf{M} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} 35 & 44 \\ 44 & 56 \end{bmatrix}$$

- Assume Cholesky factorization $\mathbf{G} = \mathbf{R}^*\mathbf{R}$ where $\mathbf{R}$ is upper triangular:

$$\mathbf{R} = \begin{bmatrix} a & b \\ 0 & c \end{bmatrix} \quad \text{(example structure)}$$

- Actual Cholesky factor:

$$\mathbf{R} \approx \begin{bmatrix} 5.916 & 7.437 \\ 0 & 0.775 \end{bmatrix}$$

CholeskyQR: Compute $\mathbf{R}^{-1}$ and $\mathbf{Q} = \mathbf{M} \cdot \mathbf{R}^{-1}$

- From Cholesky decomposition, we had:

$$\mathbf{R} = \begin{bmatrix} 5.916 & 7.437 \\ 0 & 0.775 \end{bmatrix}$$

- Compute $\mathbf{R}^{-1}$ (inverse of upper-triangular matrix):

$$\mathbf{R}^{-1} = \begin{bmatrix} \frac{1}{5.916} & -\frac{7.437}{5.916 \cdot 0.775} \\ 0 & \frac{1}{0.775} \end{bmatrix} \approx \begin{bmatrix} 0.169 & -1.621 \\ 0 & 1.290 \end{bmatrix}$$

- Now compute:

$$\mathbf{Q} = \mathbf{M} \cdot \mathbf{R}^{-1}$$

where

$$\mathbf{M} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \quad \Rightarrow \quad \mathbf{Q} \approx \begin{bmatrix} 0.169 & 0.290 \\ 0.507 & -0.041 \\ 0.845 & -0.372 \end{bmatrix}$$

- The columns of $\mathbf{Q}$ are orthonormal, satisfying the QR factorization.

Limitations of CholeskyQR and its Variants

- **Limitations of CholeskyQR:**
  - Requires the Gram matrix $\mathbf{G} = \mathbf{M}^*\mathbf{M}$ to be full rank and well-conditioned.
  - If $\mathbf{M}$ is ill-conditioned, numerical errors can accumulate during Cholesky factorization.
  - It may lead in $\mathbf{Q}$ losing orthogonality:
  - This means $\mathbf{Q}^*\mathbf{Q}$ will deviate from $\mathbf{I}$ by a factor of $\mathcal{O}(u\,\kappa(\mathbf{M})^2)$, where u is the machine precision and $\kappa(\mathbf{M})$ is the condition number of $\mathbf{M}$.

- **Variants of CholeskyQR:**
  - **CholeskyQR2:** Applies CholeskyQR twice to reduce loss of orthogonality.
  - **Jacobi Preconditioning:** It uses diagonal scaling to normalize column norms of M before applying CholeskyQR. It brings the condition number of $\mathbf{M}$ within a constant factor of the best possible diagonal preconditioner.
  - **LU Preconditioning:** Uses the upper triangular factor $\mathbf{U}$ from the LU decomposition of $\mathbf{M}$ to precondition CholeskyQR. This improves numerical stability by reducing the condition number of the matrix being factored.

- These enhancements are used to improve numerical stability and orthogonality in practice, especially for ill-conditioned matrices.

Randomized Preconditioning for CholeskyQR

- **Step 1: Random Sketching**
  - Generate a sketch $\mathbf{M}^{\text{sk}} = \mathbf{S} \cdot \mathbf{M}$, where $\mathbf{S} \in \mathbb{R}^{d \times m}$, $d \ll m$.
  - This reduces the matrix size while retaining key structural information.
- **Step 2: QR on Sketch**
  - Compute QR factorization:

$$\mathbf{Q}^{\text{sk}}, \mathbf{R}^{\text{sk}} \leftarrow \mathtt{qr}(\mathbf{M}^{\text{sk}})$$

  - Use any numerically stable QR method (e.g., Householder QR).
  - The triangular factor $\mathbf{R}^{\text{sk}}$ is used for preconditioning.
- **Step 3: Preconditioning**
  - Compute $\mathbf{M}^{\text{pre}} = \mathbf{M} \cdot \left(\mathbf{R}^{\text{sk}}\right)^{-1}$
  - The resulting matrix $\mathbf{M}^{\text{pre}}$ is better conditioned for CholeskyQR.
- **Step 4: CholeskyQR**
  - Apply CholeskyQR to $\mathbf{M}^{\text{pre}}$ to obtain a numerically stable and efficient QR factorization.
- **Step 5: Undo Preconditioning**
  - Form the final triangular factor:

$$\mathbf{R} = \mathbf{R}^{\text{pre}} \cdot \mathbf{R}^{\text{sk}}$$

  - This restores the original scaling and completes the QR factorization.

Randomized Preconditioned CholeskyQR

Given Matrix:

$$\mathbf{M} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 9 \end{bmatrix}$$

Step 1: Sketching Operation

$$\mathbf{S} = \begin{bmatrix} 1 & 0 & -1 & 2 \\ 0 & 1 & 1 & -1 \\ -1 & 2 & 0 & 1 \end{bmatrix} \quad \Rightarrow \quad \mathbf{M}_{\text{sk}} = \mathbf{S} \cdot \mathbf{M} = \begin{bmatrix} 10 & 14 \\ 1 & 1 \\ 12 & 15 \end{bmatrix}$$

Step 2: QR Decomposition of the Sketch (First Column)

$$\|\mathbf{m}_1\| = \sqrt{10^2 + 1^2 + 12^2} = \sqrt{245} \approx 15.6525$$

$$\mathbf{q}_1 = \frac{1}{15.6525} \begin{bmatrix} 10 \\ 1 \\ 12 \end{bmatrix} \approx \begin{bmatrix} 0.6390 \\ 0.0639 \\ 0.7670 \end{bmatrix}$$

Randomized Preconditioned CholeskyQR

Step 2: Continue QR Decomposition (Second Column)

$$\langle \mathbf{m}_2, \mathbf{q}_1 \rangle = 14 \cdot 0.6390 + 1 \cdot 0.0639 + 15 \cdot 0.7670 \approx 20.5149$$

$$\mathbf{v}_2 = \begin{bmatrix} 14 \\ 1 \\ 15 \end{bmatrix} - 20.5149 \cdot \begin{bmatrix} 0.6390 \\ 0.0639 \\ 0.7670 \end{bmatrix} \approx \begin{bmatrix} 0.890 \\ -0.311 \\ -0.741 \end{bmatrix}$$

$$\|\mathbf{v}_2\| \approx \sqrt{0.890^2 + (-0.311)^2 + (-0.741)^2} \approx 1.193 \quad \Rightarrow \quad \mathbf{q}_2 \approx \begin{bmatrix} 0.746 \\ -0.261 \\ -0.621 \end{bmatrix}$$

$$\mathbf{R}_{\text{sk}} \approx \begin{bmatrix} 15.6525 & 20.5149 \\ 0 & 1.193 \end{bmatrix}$$

Step 3: Preconditioning

$$\mathbf{R}_{\text{sk}}^{-1} = \begin{bmatrix} \frac{1}{15.6525} & \frac{-20.5149}{15.6525 \cdot 1.193} \\ 0 & \frac{1}{1.193} \end{bmatrix} \approx \begin{bmatrix} 0.0639 & -1.100 \\ 0 & 0.838 \end{bmatrix}$$

$$\mathbf{M}_{\text{pre}} = \mathbf{M} \cdot \mathbf{R}_{\text{sk}}^{-1} \approx \begin{bmatrix} 0.0639 & 0.576 \\ 0.192 & 0.052 \\ 0.319 & -0.472 \\ 0.447 & -0.158 \end{bmatrix}$$

Randomized Preconditioned CholeskyQR

Step 4: CholeskyQR on Preconditioned Matrix

Compute the Gram matrix:

$$\mathbf{G} = \mathbf{M}_{\text{pre}}^{*}\mathbf{M}_{\text{pre}} \approx \begin{bmatrix} 0.343 & -0.175 \\ -0.175 & 0.581 \end{bmatrix}$$

Cholesky factorization:

$$\mathbf{R}_{\text{pre}} = \begin{bmatrix} \sqrt{0.343} & \frac{-0.175}{\sqrt{0.343}} \\ 0 & \sqrt{0.581 - \frac{(-0.175)^2}{0.343}} \end{bmatrix} \approx \begin{bmatrix} 0.586 & -0.298 \\ 0 & 0.702 \end{bmatrix}$$

Now compute:

$$\mathbf{Q}_{\text{pre}} = \mathbf{M}_{\text{pre}} \cdot \mathbf{R}_{\text{pre}}^{-1} \approx \begin{bmatrix} 0.109 & 0.867 \\ 0.327 & 0.213 \\ 0.545 & -0.440 \\ 0.763 & 0.099 \end{bmatrix}$$

Step 5: Final Reconstruction of R

$$\mathbf{R} = \mathbf{R}_{\text{pre}} \cdot \mathbf{R}_{\text{sk}} \approx \begin{bmatrix} 9.17 & 11.68 \\ 0 & 0.838 \end{bmatrix}$$

Verification:

$$\mathbf{Q}_{\text{pre}} \cdot \mathbf{R} \approx \begin{bmatrix} 1.00 & 2.00 \\ 3.00 & 4.00 \\ 5.00 & 6.00 \\ 7.00 & 8.98 \end{bmatrix} \quad \text{(matches original } \mathbf{M} \text{ up to rounding)}$$

Understanding RQRCP (Randomized QR with Column Pivoting)

Problem: We are tasked with computing the QR decomposition of a large matrix A with dimensions $m \times n$, where m and n are large. Traditional QR decompositions may be computationally expensive for such matrices.

Key Goal: We aim to approximate A as a product of two matrices Q (orthogonal) and R (upper triangular), i.e.,

$$A \approx QR$$

using the randomized approach, which is faster for large matrices.

Randomized Approach:
- Instead of computing the full QR decomposition directly, we use a **randomized sketching** method to approximate A.
- This technique involves using random matrices to reduce the size of A, making the computation of Q and R more efficient.

Main Components:
- **Block Size** b: The number of columns processed in each iteration. Typically, b is a small number (e.g., 100).
- **Oversampling** s: A small number added to the block size to improve the stability of the approximation (e.g., 50).

Randomized QRCP (RQRCP) Pseudocode

Algorithm 1 *

Randomized QR with Column Pivoting

Require: Matrix $A \in \mathbb{R}^{m \times n}$, block size b, oversampling s

Initialize $k = 0$, permutation $\Pi = I_n$
Initialize $Q = \emptyset$, $R = \emptyset$
while $k < n$ **do**
    Generate random matrix $S \in \mathbb{R}^{(b+s) \times m}$
    Compute sketch $Y = SA$
    Perform QRCP on $Y \rightarrow$ pivot indices $P_{\text{block}}[1{:}b]$
    Update permutation Π with P_{block}
    $[Q_k, R_k] = \text{QR}(A[:, P_{\text{block}}])$
    Append Q_k to Q (accumulate orthogonal factors)
    Append R_k to R (accumulate upper triangular factors)
    Update trailing matrix: $A \leftarrow Q_k^T A$
    $k \leftarrow k + b$
end while
return Q, R, Π

RQRCP Example

Problem Setup: We have a matrix $A \in \mathbb{R}^{m \times n}$, where $m = 10,000$ and $n = 1,000$. We wish to compute a pivoted QR decomposition using the RQRCP method.

Step 1: Initialization
- **Matrix A**: The matrix is initialized with dimensions $m \times n$, where m is large, and n is the number of columns.
- **Permutation Matrix Π**: Initialize $\Pi = I_n$ (the identity matrix). This matrix will store the pivoting information during the process.
- **Block Size b**: Choose a block size $b = 10$. This is the number of columns to be processed in each iteration.
- **Oversampling s**: We set $s = 5$ for the oversampling factor, which helps ensure stability during the sketching process.

Step 2: Generate Random Matrix S
- **Random Matrix S**: We generate a random matrix $S \in \mathbb{R}^{(b+s) \times m}$. Here, $b + s = 15$, so S is a $15 \times 10,000$ matrix.
- **Purpose**: The matrix S serves as a sketch that helps in approximating the column space of A. It will be used to compute the sketch matrix $Y = SA$, which will then be processed for pivoting.

Step 3: Compute the Sketch Matrix Y
- The matrix Y is computed as:

$$Y = SA$$

This sketch matrix approximates the important structure of the matrix A, retaining the main information while reducing its size.

Step 4: Perform QRCP on Y
- We now perform the **QR with Column Pivoting (QRCP)** on Y. This step finds the best column pivots in Y that will help approximate the leading singular vectors of A.
- The **pivot indices** P_{block} are identified. For example, we might select the first 10 columns of Y, corresponding to the most important columns in the original matrix A.

RQRCP Example

Step 5: Update the Permutation Matrix Π
- The permutation matrix Π is updated based on the pivot indices P_{block}.
- The matrix A is reordered by multiplying it by Π, ensuring that the important columns come first.

Step 6: Compute Q_k and R_k
- The QR decomposition is performed on the selected columns of A:

$$[Q_k, R_k] = \text{QR}(A[:, P_{\text{block}}])$$

- This gives us the block orthogonal matrix Q_k and the upper triangular matrix R_k.

Step 7: Accumulate Q_k

- After computing Q_k in each iteration, we accumulate the results as follows:

$$Q \leftarrow \begin{bmatrix} Q & Q_k \end{bmatrix}$$

  where Q is built by horizontal concatenation of Q_k blocks.
- Initially, Q is empty ($Q = \emptyset$).
- In each iteration, Q grows by appending Q_k columns, building the full orthogonal matrix Q.

RQRCP Example

Step 8: Structure of R

- The matrix R is constructed iteratively with the following block structure:

$$R = \begin{bmatrix} R_1 & R_{12} & R_{13} & \cdots & R_{1p} \\ 0 & R_2 & R_{23} & \cdots & R_{2p} \\ 0 & 0 & R_3 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & R_p \end{bmatrix}$$

- **Diagonal blocks** $R_1, R_2, \ldots, R_p$:
  Upper triangular matrices from successive QR decompositions (size $b \times b$ each, except possibly the last block).
- **Off-diagonal blocks** R_{ij}:
  Cross terms generated from projecting the trailing matrix onto previous pivoted columns.

Step 9: Trailing Matrix Projection

- After processing the k-th block, update the trailing matrix to prevent redundant processing:

$$A_{\text{working}}[:, k:] \leftarrow (I - Q_k Q_k^\top) A_{\text{working}}[:, k:]$$

- **Numerical Effect**:
  - Helps the next pivoting step focus on new, unexplored directions

RQRCP: Pros and Cons

Advantages:

- **Speed:** It is much faster than the regular QRCP, up to 100 times quicker.
- **Memory:** It only stores a smaller sketch of the matrix, saving memory.
- **Rank-Revealing:** It can find the most important features, close to full SVD accuracy.

Limitations:

- **Sketch Size:** Needs extra data ($s \geq 5$) to work well.
- **Approximation:** The result only gives an approximation, not the exact values.
- **Blocking:** Choosing the right block size (b) is important for good performance.
- **Randomness:** It is based on probability, so the results may vary.

Algorithm 1: CQRRPT (Wrapper)

- **Inputs:**
  - $\mathbf{M} \in \mathbb{R}^{m \times n}$: Input matrix
  - Optional scalar $\gamma \geq 1$: Sketch size multiplier (default: 1.25)
  - Optional sketching distribution family $\mathcal{F}$ (default: sparse distributions)
- **Outputs:**
  - Orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{m \times k}$
  - Upper-triangular matrix $\mathbf{R} \in \mathbb{R}^{k \times n}$
  - Pivot vector $J \in \{1, \ldots, n\}^n$

Steps (Pseudocode)

1: **Function:** `cqrrpt(M, , F)`

2: If $\mathcal{F}$ not provided, set $\mathcal{F} \leftarrow$ Sparse Distribution Family

3: If γ not provided, set $\gamma \leftarrow 1.25$

4: Set $d \leftarrow \lceil \gamma n \rceil$

5: Draw sketching matrix $\mathbf{S} \sim \mathcal{F}_{d,m}$

6: Return: `cqrrpt_core(M, S)`

Algorithm 2: `cqrrpt_core`

- **Inputs:** Matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$, Sketch operator $\mathbf{S} \in \mathbb{R}^{d \times m}$, where $n \leq d \ll m$
- **Outputs:** Orthogonal matrix $\mathbf{Q}_k$, upper triangular matrix $\mathbf{R}_k$, permutation vector J

Pseudocode

1: function `cqrrpt_core`$(\mathbf{M}, \mathbf{S})$
2: $\quad \mathbf{M}^{\text{sk}} \leftarrow \mathbf{S} \cdot \mathbf{M}$
3: $\quad [\mathbf{Q}^{\text{sk}}, \mathbf{R}^{\text{sk}}, J] \leftarrow \texttt{qrcp}(\mathbf{M}^{\text{sk}})$
4: $\quad k \leftarrow \texttt{rank}(\mathbf{R}^{\text{sk}})$
5: $\quad \mathbf{M}^{\text{pre}} \leftarrow \mathbf{M}_k \cdot (\mathbf{A}_k^{\text{sk}})^{-1}$,
$\quad$ where $\mathbf{M}_k = \mathbf{M}[:, J[1:k]]$, $\mathbf{A}_k^{\text{sk}} = \mathbf{R}^{\text{sk}}[1:k, 1:k]$
6: $\quad [\mathbf{Q}_k, \mathbf{R}^{\text{pre}}] \leftarrow \texttt{cholqr}(\mathbf{M}^{\text{pre}})$
7: $\quad \mathbf{R}_k \leftarrow \mathbf{R}^{\text{pre}} \cdot \mathbf{R}^{\text{sk}}[1:k, :]$
8: $\quad$ return $\mathbf{Q}_k, \mathbf{R}_k, J$

Arithmetic Complexity of CQRRPT

- The arithmetic cost of Algorithm 2 depends on the cost of forming the sketch $\mathbf{M}^{\text{sk}} = \mathbf{SM}$ and on the QRCP decomposition of $\mathbf{M}^{\text{sk}}$
- **Total flop count:** Using LAPACK's `GEQP3` routine for QRCP on the sketch, the total flop count becomes:

$$2mk^2 + mk(k+1) + 4dnk - 2k^2(d+n) + \frac{5}{3}k^3 + C_{\text{sk}} + (\text{lower-order terms})$$

- This implies that the asymptotic flop count of CQRRPT has a leading term of $3mn^2$ when rank $= n$.
- **Implicit Q-Factor Optimization:**
  - When only an implicit representation of $\mathbf{Q}$ is required (e.g., via $\mathbf{M}^{\text{pre}} \cdot \mathbf{R}_{\text{pre}}^{-1}$),
  - The leading term can drop to: $\mathbf{2mn^2}$
- **Comparison with Householder QR:**
  - Explicit orthogonalization in Householder QR requires LAPACK's `ORGQR`, adding another $2mn^2$
  - CQRRPT can be up to $\frac{4}{3}\times$ cheaper than Householder QR when computing explicit $\mathbf{Q}$

How Algorithm 2 inherits Rank-Revealing properties

- Given $X \in \mathbb{R}^{m \times n}$ with $\mathrm{rank}(X) = k$, run pivoted QR: $[Q, R, J] = \mathrm{qrcp}(X)$.
- Partition R into a $k \times n$ upper-triangular block form for each $\ell \leq k$:

$$R = \begin{bmatrix} A_\ell & B_\ell \\ & C_\ell \end{bmatrix},$$

  where $A_\ell \in \mathbb{R}^{\ell \times \ell}$, $B_\ell \in \mathbb{R}^{\ell \times (n-\ell)}$, $0 \in \mathbb{R}^{(k-\ell) \times \ell}$, and $C_\ell \in \mathbb{R}^{(k-\ell) \times (n-\ell)}$.
- **RRQR property**: $\exists\, f_\ell \geq 1$ such that for all $j \leq \ell$ and $j \leq k - \ell$:

$$\sigma_j(A_\ell) \geq \frac{\sigma_j(X)}{f_\ell},$$
$$\sigma_j(C_\ell) \leq f_\ell\, \sigma_{\ell+j}(X).$$

- **Inheritance in CQRRPT**: If $\mathrm{qrcp}(SM)$ satisfies RRQR up to f_ℓ and κ is the restricted condition number of S, then CQRRPT on M satisfies RRQR up to factors κf_ℓ.
- **Strong RRQR**: additionally enforce $\|A_\ell^{-1} B_\ell\| \leq g_\ell$. CQRRPT inherits this with factors $(g_\ell + \kappa f_\ell^2)$.

Probabilistic Aspects of CQRRPT

- CQRRPT incorporates randomness through sketching matrix $\mathbf{S}$, which is sampled from a distribution family $\mathcal{F}$.
- Any quantity that $\mathbf{S}$ affects in Algorithm 2 can be regarded as a random variable — most notably $\mathbf{M}^{\text{pre}}$ and its condition number $\kappa(\mathbf{M}^{\text{pre}})$.
- The numerical performance of CQRRPT is closely tied to how well-conditioned $\mathbf{M}^{\text{pre}}$ is.
- **Question:** What is the probability that $\kappa(\mathbf{M}^{\text{pre}})$ remains within acceptable bounds, regardless of the input matrix?
- **Answer:** The probability depends on choice of **sketching distribution** $\mathcal{F}$ and the **oversampling factor** γ.

Structure of Random Sketching Matrices S

Let $\mathcal{F}$ denote a family of distributions for sketching matrices. For fixed dimensions $d \ll m$, we draw $\mathbf{S} \sim \mathcal{F}_{d,m}$.

- **Gaussian Matrices**
- **Short-Axis Sparse Operators (SASO)**
- **Subsampled Randomized Fast Trigonometric Transforms (SRFT)**

Gaussian Matrices

Definition:

- A Gaussian matrix $S \in \mathbb{R}^{m \times d}$ has entries drawn independently from:

$$S_{ij} \sim \mathcal{N}\left(0, \frac{1}{\sqrt{d}}\right)$$

Structure:

- Entries: i.i.d. $\mathcal{N}(0, 1/\sqrt{d})$
- Example (for $d = 2$, $m = 3$):

$$S = \frac{1}{\sqrt{2}} \begin{bmatrix} 0.8 & -0.5 \\ -1.2 & 0.7 \\ 0.3 & 1.1 \end{bmatrix}$$

Key Properties:

- Isotropy: $\mathbb{E}[S^* S] = I_m$ (columns are unbiased)
- Concentration: For orthonormal U, $SU \approx$ orthonormal

Intuition:

- "Smears" data uniformly in all directions — preserving geometric structure on average

Short-Axis-Sparse Operators (SASO)

Definition:

- A SASO matrix $S \in \mathbb{R}^{m \times d}$ has exactly ℓ nonzero entries per column, each being $\pm\frac{1}{\sqrt{d}}$, placed uniformly at random.

Structure:

- Each column has exactly ℓ nonzeros (e.g., $\ell = 2$)
- Nonzero values: $\pm\frac{1}{\sqrt{d}}$
- Example (for $d = 3$, $m = 4$):

$$S = \frac{1}{\sqrt{3}} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Key Properties:

- **Sparsity** $\rightarrow$ Fast matrix-vector multiplies
- $\mathbb{E}[S^*S] \approx I_m$ with proper scaling

Intuition:

- "Lights up" random coordinates like sparse spotlights

Subsampled Randomized Fast Trigonometric Transforms (SRFT)

Definition:

- (SRFT) is a matrix that combines subsampling, fast transforms, and random signs.
- It is defined as:

$$S = \sqrt{\frac{m}{d}} \cdot \underbrace{C}_{\text{subsample}} \cdot \underbrace{F}_{\text{fast transform}} \cdot \underbrace{D}_{\text{random signs}}$$

Key Properties:

- **Speed:** The fast Fourier transform makes it computationally efficient.
- **Randomness:** Introduces randomness to provide good approximation guarantees.
- **Near-Optimal:** Yields a sketch that is close to the true data representation, with significantly reduced dimensionality.

Intuition:

- SRFTs can be viewed as a way to perform dimensionality reduction while preserving key features of the data.
- *"Randomize → Transform → Subsample"* — this sequence captures the essence of SRFTs' efficiency.

Visual:

- A 3-step diagram: **Randomize → Transform → Subsample**

Step 1 : Randomization(Breaking Structure with Sign Flips)

Definition:

- The matrix D is a diagonal matrix where each entry is randomly chosen from ± 1 (Rademacher variables).

Example: For $m = 4$, the matrix D is:

$$D = \begin{bmatrix} +1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & +1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Purpose:

- The random signs are applied to input data columns to break any "bad alignments" and prevent undesirable correlations in the data.

Intuition:

- Think of this process like shuffling a deck of cards before dealing — each card (data element) gets randomized to avoid unfavorable structures.

Visual:

- A vector $[x_1, x_2, x_3, x_4]$ where random ± 1 signs are applied to each element.

$$\text{Before: } [x_1, x_2, x_3, x_4] \quad \Rightarrow \quad \text{After: } [+x_1, -x_2, +x_3, -x_4]$$

Step 2 : Fast Trigonometric Transforms (Mixing Information)

Definition:

- The matrix F represents an orthogonal/unitary transform used to mix information across different coordinates, ensuring a more efficient representation of data.
- These transforms redistribute the components of the input data, making it possible to represent them in terms of different bases or "coordinates."
- Examples of transforms:
  - **Fourier Transform (FFT):** Converts data to "frequency" space, allowing us to analyze the frequency components of a signal or dataset. This is especially useful in signal processing and time-series analysis.
  - **Walsh-Hadamard Transform (WHT):** Uses binary ± 1 patterns to transform data. It is known for its simplicity and efficiency in computations, especially in binary coding and fast algorithms.

Example matrices for $m = 4$:

WHT:

$$F_{\text{WHT}} = \frac{1}{2}\begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix}$$

FFT:

$$F_{\text{FFT}} = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}$$

Purpose:

- This ensures no coordinate retains isolated or uncorrelated information after transformation.
- The FFT mixes using complex exponentials (phases), while the WHT uses simple binary flips.

Step 3 : Subsampling (C) – Reducing Dimension

Definition:

- Subsampling refers to selecting a smaller subset of coordinates (or features) from a larger set.
- The matrix C is used to select d coordinates out of m total dimensions, typically chosen uniformly at random.
- This operation reduces the dimensionality of the data while aiming to retain essential information.

Example:

- Let $m = 4$, $d = 2$
- Suppose we randomly select the 2nd and 4th columns. The matrix C becomes:

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- When C multiplies a vector of length 4, it extracts the 2nd and 4th components.

Purpose:

- Reduces dimension from $m \to d$, helping with faster computations and storage savings.
- This step is crucial in randomized linear algebra, compressed sensing, and fast data sketching algorithms.
- Although some information is discarded, if the original data is already mixed well (via earlier steps), key patterns are likely preserved.

SRFT Construction – Step 1 & 2: Random Sign Flip and Fast Transform

Goal: Construct a Subsampled Randomized Fourier Transform (SRFT) matrix S to reduce input from dimension $m = 4$ to $d = 2$.

Step 1 – Random Sign Flipping (D):

- Flip signs of input vector randomly using a diagonal matrix D.
- Example: $D = \text{diag}(+1, -1, +1, -1)$

Step 2 – Fast Transform (F):

- Apply Walsh-Hadamard Transform to spread the information.
- WHT matrix (for $m = 4$):

$$F = \frac{1}{2} \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix}$$

- Apply the transform: $F \cdot D$

$$F \cdot D = \frac{1}{2} \begin{bmatrix} +1 & -1 & +1 & -1 \\ +1 & +1 & +1 & +1 \\ +1 & -1 & -1 & +1 \\ -1 & +1 & +1 & -1 \end{bmatrix}$$

SRFT Construction – Step 3 & 4: Subsampling and Scaling

Step 3 – Subsampling (C):

- Select $d = 2$ rows from the transformed matrix $F \cdot D$.
- Let's pick rows 2 and 4:

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(a } 2 \times 4 \text{ matrix)}$$

- Apply subsampling: $C \cdot (F \cdot D)$

$$C \cdot (F \cdot D) = \frac{1}{2} \begin{bmatrix} +1 & +1 & +1 & +1 \\ -1 & +1 & +1 & -1 \end{bmatrix}$$

Step 4 – Scaling:

- Multiply result by scaling factor $\sqrt{m/d} = \sqrt{4/2} = \sqrt{2}$

$$S = \sqrt{2} \cdot \frac{1}{2} \begin{bmatrix} +1 & +1 & +1 & +1 \\ -1 & +1 & +1 & -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} +1 & +1 & +1 & +1 \\ -1 & +1 & +1 & -1 \end{bmatrix}$$

Conclusion: The final SRFT matrix $S \in \mathbb{R}^{2\times 4}$ projects from 4D to 2D while preserving structure.

Why SRFTs Work – Key Properties

Isotropy:

- $\mathbb{E}[S^*S] = I_m$: Unbiased in expectation.
- No directional preference — energy spreads across dimensions.

Speed:

- FFT/WHT transforms make SRFT fast: $O(m \log m)$.
- Much faster than Gaussian ($O(md)$).

Concentration:

- SU is nearly orthonormal with high probability when applied to orthonormal U.
- Ensures preservation of important geometric structure.

Property	**Gaussian**	**SASO**	**SRFT**
Structure	Dense	Sparse	Structured + Random
Speed	Slow ($O(md)$)	Fast ($O(\ell d)$)	Fastest ($O(m \log m)$)
Guarantees	Optimal	Moderate	Near-optimal
Use Case	Small-scale theory	Sparse data	Large-scale practice

Insight: SRFT strikes a balance, offering both speed and good theoretical guarantees for large-scale applications.

Thank You