~~Naive bfbcg test~~

~~ inside orth(\*) ~~

Set up memory and handler 0.000724 s

Part(a): mtxS <- mtxZ' \* mtxZ 0.003251 s

Part(b): SVD Decomp for mtxS 0.004908 s

Part(c): Transpose mtxV <- mtxVT' 0.000026 s

Part(d): Calculate current rank 0.000010 s

Part(e): Truncate matrix V 0.000016 s

Part(f): mtxY <- mtxZ \* mtxV Truncated 0.002136 s

Part(g): Normalize mtxY 0.001812 s

~~Exit orth(\*)~~

💫💫💫 Iteration 1 💫💫💫

Q <- AP: 0.001363 s

Part(a): (P'Q) 0.000844 s

~~ inside (P'Q)^{-1} ~~

Part(b.i): Create Identity Matrix 0.000025 s

Part(b.ii): Perform QR decompostion 0.000982 s

Part(b.iii): Obtainig Q' implicitly 0.000026 s

Part(b.iv): Solve RX = Q', then result is (P'Q) 0.001565 s

~~Exit (P'Q)^{-1}~~

Part(b): (P'Q)^{-1} 0.002819 s

Part(C): (P'R) 0.000842 s

Part(d): Alpha <- (P'Q)^{-1} \* (P'R): 0.000023 s

X\_{i+1} <- x\_{i} + P \* alpha: 0.000074 s

R\_{i+1} <- R\_{i} - Q \* alpha: 0.000075 s

Z\_{i+1} <- MR\_{i+1}: 0.000758 s

beta <- -(P'Q)^{-1} \* (Q'Z\_{i+1}): 0.000861 s

(\*) <- Z\_{i+1} + p \* beta: 0.000073 s

~~ inside orth(\*) ~~

Set up memory and handler 0.000674 s

Part(a): mtxS <- mtxZ' \* mtxZ 0.000930 s

Part(b): SVD Decomp for mtxS 0.000359 s

Part(c): Transpose mtxV <- mtxVT' 0.000015 s

Part(d): Calculate current rank 0.000009 s

Part(e): Truncate matrix V 0.000014 s

Part(f): mtxY <- mtxZ \* mtxV Truncated 0.000170 s

Part(g): Normalize mtxY 0.000589 s

~~Exit orth(\*)~~

P\_{i+1} = orth(\*): 0.004413 s

= = current Rank: 5 = =

💫💫💫 Iteration 2 💫💫💫

Q <- AP: 0.001197 s

Part(a): (P'Q) 0.000843 s

~~ inside (P'Q)^{-1} ~~

Part(b.i): Create Identity Matrix 0.000009 s

Part(b.ii): Perform QR decompostion 0.000056 s

Part(b.iii): Obtainig Q' implicitly 0.000013 s

Part(b.iv): Solve RX = Q', then result is (P'Q) 0.000039 s

~~Exit (P'Q)^{-1}~~

Part(b): (P'Q)^{-1} 0.000324 s

Part(C): (P'R) 0.000841 s

Part(d): Alpha <- (P'Q)^{-1} \* (P'R): 0.000019 s

X\_{i+1} <- x\_{i} + P \* alpha: 0.000074 s

R\_{i+1} <- R\_{i} - Q \* alpha: 0.000075 s

Z\_{i+1} <- MR\_{i+1}: 0.000689 s

beta <- -(P'Q)^{-1} \* (Q'Z\_{i+1}): 0.000860 s

(\*) <- Z\_{i+1} + p \* beta: 0.000073 s

~~ inside orth(\*) ~~

Set up memory and handler 0.000664 s

Part(a): mtxS <- mtxZ' \* mtxZ 0.000928 s

Part(b): SVD Decomp for mtxS 0.000343 s

Part(c): Transpose mtxV <- mtxVT' 0.000014 s

Part(d): Calculate current rank 0.000009 s

Part(e): Truncate matrix V 0.000014 s

Part(f): mtxY <- mtxZ \* mtxV Truncated 0.000186 s

Part(g): Normalize mtxY 0.000491 s

~~Exit orth(\*)~~

P\_{i+1} = orth(\*): 0.004300 s

= = current Rank: 4 = =

💫💫💫 Iteration 3 💫💫💫

Q <- AP: 0.001076 s

Part(a): (P'Q) 0.000852 s

~~ inside (P'Q)^{-1} ~~

Part(b.i): Create Identity Matrix 0.000009 s

Part(b.ii): Perform QR decompostion 0.000048 s

Part(b.iii): Obtainig Q' implicitly 0.000013 s

Part(b.iv): Solve RX = Q', then result is (P'Q) 0.000034 s

~~Exit (P'Q)^{-1}~~

Part(b): (P'Q)^{-1} 0.000316 s

Part(C): (P'R) 0.000851 s

Part(d): Alpha <- (P'Q)^{-1} \* (P'R): 0.000022 s

X\_{i+1} <- x\_{i} + P \* alpha: 0.000075 s

R\_{i+1} <- R\_{i} - Q \* alpha: 0.000072 s

Z\_{i+1} <- MR\_{i+1}: 0.000680 s

beta <- -(P'Q)^{-1} \* (Q'Z\_{i+1}): 0.000861 s

(\*) <- Z\_{i+1} + p \* beta: 0.000068 s

~~ inside orth(\*) ~~

Set up memory and handler 0.000660 s

Part(a): mtxS <- mtxZ' \* mtxZ 0.000928 s

Part(b): SVD Decomp for mtxS 0.000369 s

Part(c): Transpose mtxV <- mtxVT' 0.000015 s

Part(d): Calculate current rank 0.000009 s

Part(e): Truncate matrix V 0.000003 s

Part(f): mtxY <- mtxZ \* mtxV Truncated 0.000001 s

Part(g): Normalize mtxY 0.000001 s

~~Exit orth(\*)~~

P\_{i+1} = orth(\*): 0.003633 s

!!!Current Rank became 0!!!

🔸Exit iteration🔸

🫥Relative Residue: 0.000000🫥

For SVD

1. Setup: Reuse handler before SCD
2. Part(a): Replcae sgeDmm with cublasDsyrk(upper triangular and miller) in Part (a) Part(a): mtxS <- mtxZ' \* mtxZ
3. Part(g): Normalization

**Kernel: Optimizing the Normalization Function**

**Implementation Steps:**

1. **Write a Custom Kernel for Normalization: See the code example**
2. **Call the Kernel: See the Example code**

**Alt: Optimizing the Normalization Function**

For the normalization step, since mtxY\text{mtxY}mtxY is also a large matrix (N×currentRankN \times \text{currentRank}N×currentRank), using shared memory may still not be practical.

**Alternative Approach**

* **Compute Norms Using cuBLAS cublasDnrm2:**
  + Compute the norms of each column without transferring data to the host.
* **Scale Columns Using cublasDscal:**
  + Scale each column vector on the device.

**Implementation Steps**

1. **Compute Norms and Inverses:**
   * Allocate a device array norms\_d of size currentRank.
2. **Compute Norms: See sample code**
3. **Compute Inverses: See sample code**
   * Use a kernel to compute the inverses of the norms.
4. Scale Columns: See Sample Code

**Benefits:**

* **Device-Only Computation:**
  + All computations are performed on the device, avoiding host-device transfers.
* **Utilizes cuBLAS Optimizations:**
  + Efficient vector operations.

**Considerations:**

* **Kernel Launch Overhead:**
  + The inverses computation introduces an additional kernel launch but is minimal compared to data transfer overhead.