QR factorization with column pivoting: a computer scientist's perspective

Edward Hutter

December 13, 2017

Citations

Summary from recent work on randomized QR factorization with column pivoting[1],[2],[3]

[1] Duersch, Gu; 2017; "Randomized QR with Column Pivoting"

[2] Martinsson, et al; 2017; "Householder QR factorization with randomization for column pivoting"

[3] Martinsson; 2015; "Blocked rank-revealing QR factorization: How randomized sampling can be used to avoid single-vector pivoting"

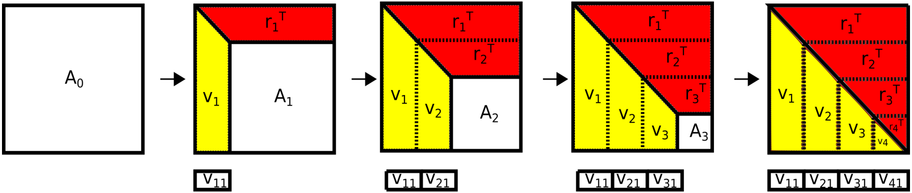
Householder QR

Householder QR - orthogonal triangularization

Goal: obtain upper-triangular $R_{n\times n}$ via $Q^T A = R$

Strategy: apply orthogonal reflectors to $A_{m\times n}$ to clear out below diagonals

- For i in range(n)
  1. Obtain column norm: $\|a_i\|$
  2. Obtain Householder reflector $v_i =$ such that $a_i - \frac{2 v_i v_i^T}{v_i^T v_i} \cdot a_i = \|a_i\| \cdot e_i$
  3. Update all trailing columns: $A_{i+1} = \begin{bmatrix} I_i & 0 \\ 0 & I_{n-i} - \frac{2 v_i v_i^T}{v_i^T v_i} \end{bmatrix} A_i$



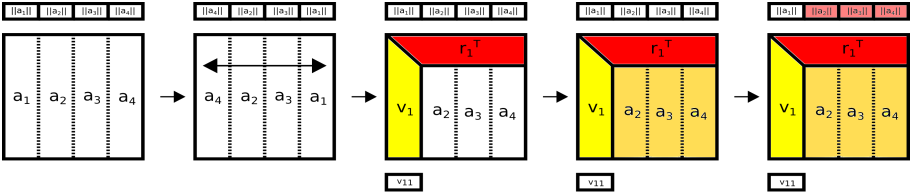
Householder QR with column pivoting

Goal: obtain upper-triangular $R_{n\times n}$ via $Q^T AP = R$ with $|R_{ii}| > |R_{jj>ii}|$

Strategy: apply orthogonal matrices Q and column swaps to $A_{m\times n}$ to clear out below diagonals

Differences from Householder QR:

- Keep array of column norms for pivoting decisions
- Swap subcolumn with greatest 2-norm to attain next reflector v_i
- Stop iterating when $\|a_j\| < \epsilon$: rank revealing capability!



Does column pivoting affect performance?

(a) Compare black, green, pink[1]

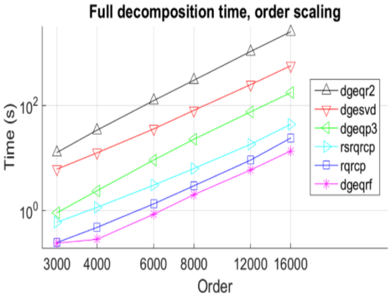


FIG. 7.1. *24 cores, $m = n$ scaled.*

(b) Compare black, blue[1]

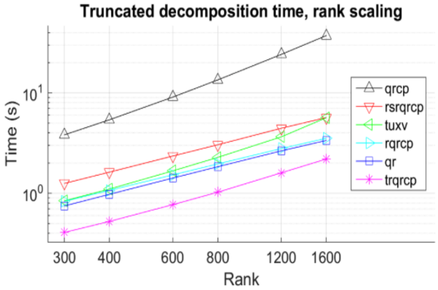


FIG. 7.2. *24 cores, $m = 12000$, $n = 12000$, k scaled.*

[1]Duersch, Gu; 2017; "Randomized QR with Column Pivoting"

Performance investigation into QR with column pivoting

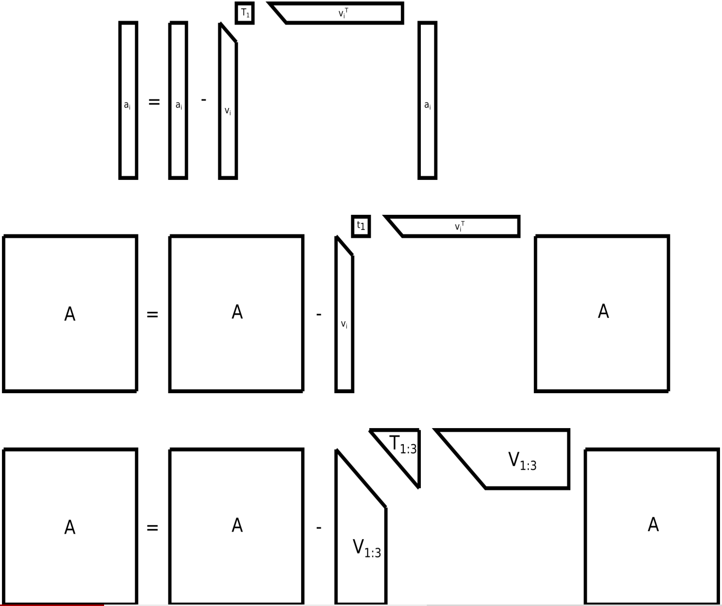
- Lets compare flop count
  - $T_{\text{HQR}}(m,n) = \sum_{i=0}^{n-1} 2(m-i) + 2(m-i)(n-i) \approx 2mn^2 - 2/3n^3$
  - $T_{\text{HQRCP}}(m,n) = mn + \sum_{i=0}^{n-1} (n-i) + 2(m-i)(n-i) \approx 2mn^2 - 2/3n^3$
  - $T_{\text{HQRCP}}(m,n,k) = mn + \sum_{i=0}^{k-1} (n-i) + 4(m-i)(n-i) \approx 2mnk$
- Same flop count for full-rank matrices!
- What is needed before we can obtain next Householder reflector v_{i+1}?
  - HQR: Reflection of a_{i+1} via $Q_i a_{i+1}$
  - HQRCP: Updating all trailing columns $a_{j>i}$ and norms, then swapping
- Big difference! Let's disect the trailing matrix update

Trailing matrix update

BLAS level 1

BLAS level 2

BLAS level 3



Preliminaries

- Assume two-level memory subsystem
  - Fast memory of size $\hat{M}$
  - Slow memory of size M
- Focus on large matrices : $\hat{M} < 2m$
- Let $\tau_i = \frac{2}{v_i^T v_i}$

Trailing matrix update with BLAS level 1

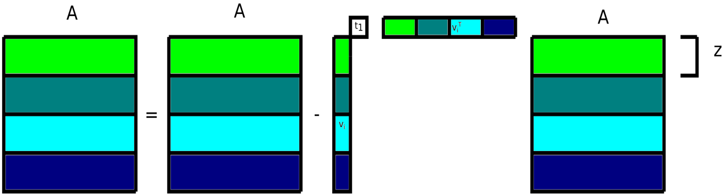
- Operations are column-centric
- For each trailing matrix column (inner) iteration:
  - reflector v_i read from M to $\hat{M}$ 2x
  - trailing column $a_{j>=i}$ read from M to $\hat{M}$ 2x
- First trailing matrix update: $2mn$ flops, $4mn$ reads
- Takeaway: more data movement than useful flops!

Trailing matrix update with BLAS level 2

- Operations are matrix-centric
- Smart chunking along rows of A allows re-use of v_i
- For each trailing matrix update (all columns)
  - reflector v_i read from M to $\hat{M}$ 2x
  - trailing A read from M to $\hat{M}$ 2x
- First trailing matrix update: $2mn$ flops, $2mn + 2m$ reads

Figure: Assume $2 \cdot z <= \hat{M}$

BLAS level 2



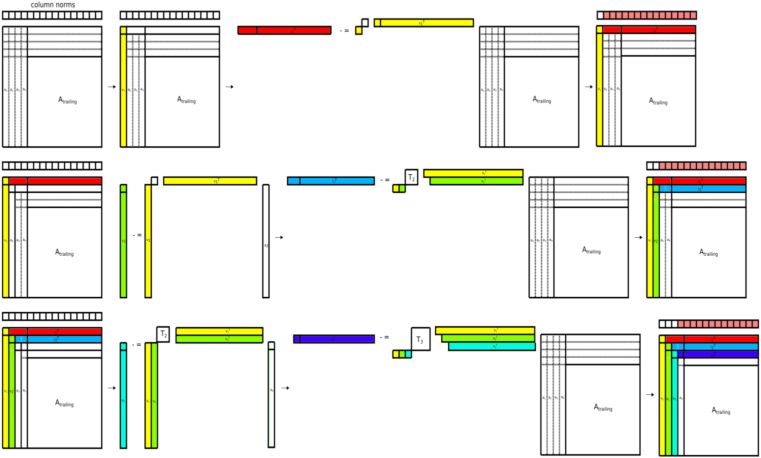
Analysis

- BLAS level 2 barely improves upon level 1.
- In both levels, trailing A must be read from memory 2x per update
- New goal: reduce the need for trailing matrix updates at each iteration.
  - Non-pivoted HQR can delay updates every b iterations, for a total of n/b block reflector updates
  - In addition, the reflectors are no longer vectors, so we can perform rank-b update instead of rank-1 update.
    - Block reflectors allow usage of BLAS Level 3, with $\mathcal{O}(b)$ useful work per memory access
  - Can HQRCP do the same kind of delaying? Remember the dependency difference from before!

Aggregation with BLAS level 3

- Key insight: only reflection of current row is needed for norm updates
- Delayed updates will need to modify current row and pivot column at each iteration
- Proceed in $\frac{n}{b}$ block iterations of size b
- After inner loop, blocked rank-b trailing matrix update is applied
- First block iteration: $\sim 2mnb$ flops, $\sim mnb + 2mb$ reads
- Lets see how each block-iteration works before trailing matrix update...

QRCP BLAS level 3 block iteration



Performance comparisons against BLAS Level 3 QRCP

(a) Compare green, blue[1]

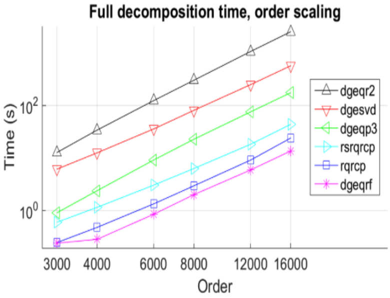


FIG. 7.1. *24 cores, $m = n$ scaled.*

(b) Compare all[1]

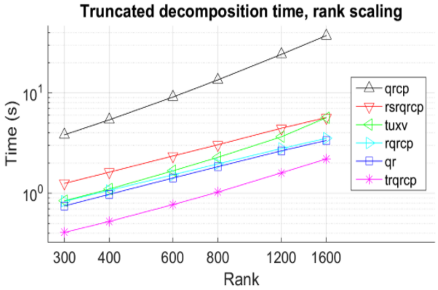


FIG. 7.2. *24 cores, $m = 12000$, $n = 12000$, k scaled.*

[1] Duersch, Gu; 2017; "Randomized QR with Column Pivoting"

Randomization to the rescue

- New motivation: find pivot columns without knowledge of $A_{m \times n}$
  - BLAS-3 variant didn't help: entire A had to be read from slow to fast memory per iteration
- dimensional reduction via random sampling: $B_{l \times n} = \Omega_{l \times m} A_{m \times n}$
  - $\Omega_{l \times n}$ has unit-variance Gaussian independent identically distributed elements
  - preserves linear dependencies among columns
- QRCP on B with tunable blocksize $l = b + k$ for oversampling parameter k
- QRCP now a building block in a new algorithm

Optimizations to randomized QRCP

- Ideas:
  - Don't want to reform $B_{l\times n} = \Omega_{l\times m} A_{m\times n}$ at each block iteration
  - Don't want a trailing matrix update
  - Want to exploit BLAS level-3 reflector blocking

Is this even possible?

Truncated randomized QRCP algorithm without trailing update[1]

- Form $B_{b+k \times n} = \Omega_{b+k \times m} A_{m \times n}$
- For i in range$(0, \lceil \frac{n}{b} \rceil, b)$
  - Find b pivot indices via QRCP(B)
  - Swap b pivots into current b columns of A
  - Permute current elements in completed rows of R, Y
  - Accumulate blocked reflector updates to current b pivot columns
  - Attain blocked reflectors via QR on b current columns
  - Aggregate reflectors to collection of existing reflectors
  - Accumulate blocked reflector updates to curent b rows
  - Downsample B

[1] Duersch, Gu; 2017; "Randomized QR with Column Pivoting"

Randomization details

- $\Omega_{l \times n}$ has unit-variance Gaussian independent identically distributed elements
- Chi-squared distribution with l degrees of freedom gives E and Var
  - $E(\|b_j\|_2^2) = l \cdot \|a_j\|_2^2$
  - $Var(\|b_j\|_2^2) = 2l \cdot \|a_j\|_2^4$
- Biases can be introduced
  - Post-hoc selection
  - Compression matrix no longer GIID after multiple orthogonal transformations
- Error bounds and analysis on potential problems given in paper[1], still working on understanding these

[1] Duersch, Gu; 2017; "Randomized QR with Column Pivoting"

Does new randomized scheme improve performance?

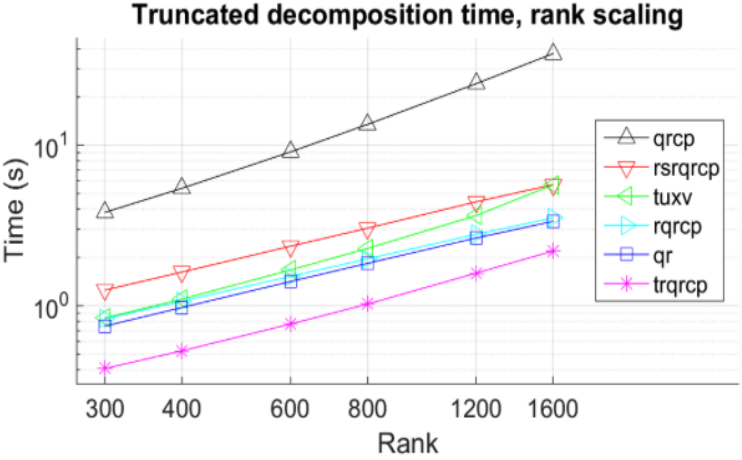


Figure: Performance comparisons: randomized vs. classical[1]

Numerical comparison

(a) Dataset 1[1]

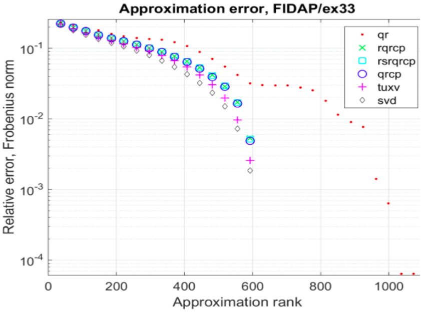


Fig. 7.3. *Matrix: FIDAP/ex33.* 1733 × 1733.

(b) Dataset 2[1]

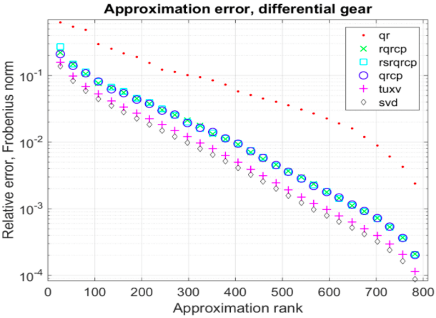


Fig. 7.4. *Matrix: Differential Gear [14].* 1280 × 804.

[1] Duersch, Gu; 2017; "Randomized QR with Column Pivoting"

Progress

- Working on implementing all of these different variants in Python
- Performing numerical tests for deviation from orthogonality and residual for matrices of different conditioning and rank
- Trying to get better understanding of randomization effects
- Developing a distributed-memory algorithm that minimizes communication and synchronization