Algorithms for Computing Strong RRQR Factorization

Shashi Satyam
Shiuli Subhra Ghosh
Shreyan Patra
Shubham Parashar

Chennai Mathematical Institute

July 1st, 2021

Overview

1. Introduction

2. Factorizations

3. Computational Algorithms

4. Algorithmic Properties

Abstract

- Finding the numerical rank of a matrix has applications in subset selection, least squares, regularization, matrix approximation etc.
- The SVD is the “best” rank-revealing factorization. However, it can be very computationally expensive and is sensitive to change in A.
- We first review the use of QR factorization as a rank-revealing alternative over SVD.
- Then we introduce a 'strong' rank revealing QR (SRRQR) factorization which apart form rank, also provides a basis for the approximate right null space.

Introduction - Covered so far...

Over the course, we looked at two ways to compute the numerical rank.

1. Using SVD
2. Using QR with column pivoting.

In this project we extend the concepts and algorithms pertaining to the QR methods.

Note - QR with column pivoting is essentially a modification of householder's procedure!

Introduction - SVD and Numerical Rank

Let $A \in \mathbb{R}^{m \times n}$ and suppose $r = \text{rank}(A) \leq \min\{m, n\}$. If $A = U\Sigma V^T$ is the SVD of A, then $A = \sum_{k=1}^{r} \sigma_k U_k {V_k}^T$ where $\sigma_i(i > r) = 0$.

- In theory $\sigma_i(i > r) = 0$. But, in reality while solving with computer it is not.
- So, we choose a $\hat{r}$ for which $\sigma_1, \sigma_2, \ldots, \sigma_{\hat{r}} \neq 0$ and rest of the σ's are stipulated to be zero.

Numerical Rank

In such a case $\hat{r}$ is called the numerical rank of A. Typically $\hat{r} \leq r$.

In this project we use a slightly different notion for Numerical Rank.

Introduction - Partial QR and Numerical Rank

Given a matrix $M \in \mathbb{R}^{m \times n}$ with $m \geq n$, we consider a partial QR factorization of the form,

$$M\Pi = QR = Q \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix}$$

where,

- $Q \in \mathbb{R}^{m \times m}$ is orthogonal and $\Pi \in \mathbb{R}^{n \times n}$ is a permutation matrix.
- $A_k \in \mathbb{R}^{k \times k}$ is upper triangular with non negative diagonal elements
- $B_k \in \mathbb{R}^{k \times n-k}$ is linearly dependent on A_k
- $C_k \in \mathbb{R}^{m-k \times n-k}$ has a sufficiently small norm

Numerical rank

k is chosen to be the smallest integer $1 \leq k \leq n$ for which $||C_k||_2$ is sufficiently small and is called the numerical rank of M.

RRQR Factorization

Rank Revealing QR Factorisation

We call a QR factorization rank-revealing if it satisfies,

$$\sigma_{min}(A_k) \geq \frac{\sigma_k(M)}{p(k,n)} \quad \text{and} \quad \sigma_{max}(C_k) \leq \sigma_{k+1}(M)p(k,n)$$

where $p(k,n)$ is a function bounded by a low degree polynomial in k and n.

- We assume that $\sigma_k(M) >> \sigma_{k+1}(M) \approx 0$ such that numerical rank of M is k.
- Our aim for RRQR algorithms is to find a Π for which $\sigma_{min}(A_k)$ is sufficiently large and $\sigma_{max}(C_k)$ is sufficiently small.

SRRQR Factorization

- Some numerical applications require a basis for the approximate right null space of M, as in-
  - Rank-deficient least-squares computations
  - Subspace tracking
- Others require us to separate the linearly independent columns of M from the linearly dependent ones, as in-
  - Subset selection
  - Linear dependency analysis

The RRQR factorization does not lead to a stable algorithm because the elements of $A^{-1}B$ can be very large. So we need stronger conditions over RRQR to attain such information.

SRRQR Factorization

Strong Rank Revealing QR Factorisation

A QR factorization strong rank revealing if, for all $1 \leq i \leq k$ and $1 \leq j \leq n-k$

- $\sigma_i(A_k) \geq \dfrac{\sigma_i(M)}{q_1(k,n)}$ and $\sigma_j(C_k) \leq \sigma_{k+j}(M) q_1(k,n)$
- $|(A_k^{-1}B_k)_{i,j}| \leq q_2(k,n)$

Where, q_1 and q_2 are functions bounded by a low degree polynomial in k and n.

Columns of $N = \Pi \begin{pmatrix} -A_k^{-1}B_k \\ I_{n-k} \end{pmatrix}$ form an approximate basis for the right null space of M.

Algorithms - Notation

By convention,

- $A_k, \bar{A}_k \in \mathbb{R}^{k \times k}$ denote upper triangular matrices with non negative diagonal elements
- $B_k, \bar{B}_k \in \mathbb{R}^{k \times n-k}$ and $C_k, \bar{C}_k \in \mathbb{R}^{m-k \times n-k}$ denote general matrices

In the partial QR factorization $X = Q \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix}$ of a matrix $X \in \mathbb{R}^{m \times n}$, where the diagonal elements of A_k are non negative, we write

$$\mathcal{A}_k(X) = A_k, \;\; \mathcal{C}_k(X) = C_k, \;\; \mathcal{R}_k(X) = \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix}$$

Further define the following-

- $1/\omega_i$ denotes the 2-norm of the i^{th} row of A^{-1} (where $A \in \mathbb{R}^{l \times l}$ is non singular)
- γ_i denotes the 2-norm of the i^{th} column of C (where C has l columns)
- $\omega_*(A) = (\omega_1(A), \omega_2(A), \cdots, \omega_l(A))^T$ and $\gamma_*(A) = (\gamma_1(A), \gamma_2(A), \cdots, \gamma_l(A))$

Algorithm 1: QR with column pivoting

ALGORITHM 1. QR with column pivoting.

$k := 0;\ \ R := M;\ \ \Pi := I;$

while $\max_{1 \le j \le n-k} \gamma_j\left(\mathcal{C}_k(R)\right) \ge \delta$ **do**

$\quad j_{\max} := \operatorname{argmax}_{1 \le j \le n-k}\ \gamma_j\left(\mathcal{C}_k(R)\right);$

$\quad k := k + 1;$

$\quad$ Compute $R := \mathcal{R}_k(R\,\Pi_{k,k+j_{\max}-1})$ and $\Pi := \Pi\ \Pi_{k,k+j_{\max}-1};$

endfor;

- This algorithm uses a greedy strategy for finding well-conditioned columns.
- Having determined the first k columns it picks a column from the remaining $n - k$ columns that maximizes $\det(\mathcal{A}_{k+1}(R))$
- When the algorithm halts,

$$\sigma_{max}(C_k(M\Pi)) \le \sqrt{n-k}$$

$$\max_{1 \le j \le n-k} \gamma_j(C_k(M\Pi)) \le \sqrt{n-k}\delta$$

Failing of Algorithm 1

- When there are only a few well-conditioned columns, this strategy is guaranteed to find a strong RRQR factorization.
- It works well in general but fails for the following example.
  Let $M = S_n K_n$ where,

$$S_n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & s & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & s^{n-1} \end{pmatrix} \quad \text{and} \quad K_n = \begin{pmatrix} 1 & -c & \cdots & -c \\ 0 & 1 & \cdots & -c \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 \end{pmatrix}$$

  where $c^2 + s^2 = 1$ and $c, s > 0$
- Here all the columns of M are unit norms so no permutation of columns take place.

Failing of Algorithm 1

- Let $k = n - 1$, then it can be shown that,

$$\frac{\sigma_k(M)}{\sigma_{min}(A_k)} \geq \frac{c^3(1+c)^{n-4}}{2s}$$

  and the right hand side grows faster than any polynomial in k and n.
- For example, let $n = 100, k = 99, c = 0.2$. Though M is not in rank-revealed form, the singular values are,

$$\sigma_{100}(M) \approx 3 \times 10^{-9}, \quad \sigma_{99}(M) \approx 0.1482, \quad \overline{\sigma}_{99} \approx 4 \times 10^{-9}$$

  Although the 99^{th} and 100^{th} singular values are well separated, the smallest singular value of the first 99 columns ($\overline{\sigma}_{99}$) is exponentially smaller than $\sigma_{99}(M)$.

Algorithms 2: Hybrid-III(k)

ALGORITHM 2. Hybrid-III(k).

$R := M$; $\Pi := I$;
repeat
$i_{\min} := \operatorname{argmin}_{1\le i\le k}\ \omega_i\,(\mathcal{A}_k(R))$;
if there exists a j such that $\det\left[\mathcal{A}_k(R\,\Pi_{i_{\min},j+k})\right] / \det\left[\mathcal{A}_k(R)\right] > 1$ **then**
Find such a j;
Compute $R := \mathcal{R}_k(R\,\Pi_{i_{\min},j+k})$ and $\Pi := \Pi\,\Pi_{i_{\min},j+k}$;
endif;
$j_{\max} := \operatorname{argmax}_{1\le j\le n-k}\ \gamma_j\,(\mathcal{C}_k(R))$;
if there exists an i such that $\det\left[\mathcal{A}_k(R\,\Pi_{i,j_{\max}+k})\right] / \det\left[\mathcal{A}_k(R)\right] > 1$ **then**
Find such an i;
Compute $R := \mathcal{R}_k(R\,\Pi_{i,j_{\max}+k})$ and $\Pi := \Pi\,\Pi_{i,j_{\max}+k}$;
endif;
until no interchange occurs;

- Algorithm 2 keeps interchanging the most "dependent" of the first k columns (column i_{min}) with one of the last $n-k$ columns, and interchanging the most "independent" of the last $n-k$ columns (column j_{max}) with one of the first k columns, as long as det $[\mathcal{A}_k(R)]$ strictly increases.

Failing of Algorithm 2

- Considering the previous example where $M = S_n K_n$, let $k = n - 2$.

$$M = \begin{pmatrix} S_{k-1}K_{k-1} & 0 & 0 & -cS_{k-1}d_{k-1} \\ & \mu & 0 & 0 \\ & & \mu & 0 \\ & & & \mu \end{pmatrix}$$

where $d_{k-1} = (1, \ldots, 1)^T \in \mathbb{R}^{k-1}$ and $\mu = \frac{1}{\sqrt{k}} \min_{1 \le i \le k-1} \omega_i(S_{k-1}K_{k-1})$

- Then the algorithm does not permute the columns of M, yet it can be shown that,

$$\frac{\sigma_{k-1}(M)}{\sigma_{k-1}(A_k)} \ge \frac{c^3(1+c)^{n-4}}{2s} \quad \text{and} \quad ||A_k^{-1}B_k||_\infty = c(1+c)^{k-2}$$

and the right hand side grows faster than any polynomial in k and n.

Algorithm 3: Compute strong RRQR, given k

ALGORITHM 3. Compute a strong RRQR factorization, given k.

$R := \mathcal{R}_k(M)$; $\Pi := I$;

while there exist i and j such that $\det(\bar{A}_k))/\det(A_k) > f$,

where $R = \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix}$ and $\mathcal{R}_k(R\,\Pi_{i,j+k}) = \begin{pmatrix} \bar{A}_k & \bar{B}_k \\ & \bar{C}_k \end{pmatrix}$, **do**

Find such an i and j;

Compute $R := \mathcal{R}_k(R\,\Pi_{i,j+k})$ and $\Pi := \Pi\,\Pi_{i,j+k}$;

endwhile;

- This algorithm constructs a SRRQR factorization by using column interchanges to try to maximize $\det(A_k)$.

Why $\det(A_k)$?

Since, $\det(A_k) = \Pi_{i=1}^{k}\sigma_i(A_k) = \sqrt{\det(M^T M)}/\Pi_{j=1}^{n-k}\sigma_j(C_k)$ a strong RRQR factorization also results in a large $\det(A_k)$.

Lemma 3.1

To prove that algorithm 3 computes a SRRQR factorization, we first express $\det(\bar{A}_k)/\det A_k$ in terms of $\omega_i(A_k), \gamma_j(C_k)$ and $({A_k}^{-1}B_k)_{i,j}$.

Let, $R = \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix}$ and $\mathcal{R}_k(R\ \Pi_{i,j+k}) = \begin{pmatrix} \bar{A}_k & \bar{B}_k \\ & \bar{C}_k \end{pmatrix}$. Where A_k has positive diagonal elements. Then,

$$\frac{det(\overline{A}_k)}{det(A_k)} = \sqrt{(A_k^{-1}B_k)_{i,j}^2 + (\gamma_j(C_k)/\omega_i(A_k))^2}$$

Some Additional Notation

We will be using Lemma 3.1 to modify Algorithm 3 in two ways. Define $\rho(R,k)$ and $\hat{\rho}(R,k)$ for the same, where

$$\rho(R,k) = \max_{1\leq i\leq k, 1\leq j\leq n-k} \sqrt{(A_k^{-1}B_k)_{i,j}^2 + (\gamma_j(C_k)/\omega_i(A_k))^2}$$

$$\hat{\rho}(R,k) = \max_{1\leq i\leq k, 1\leq j\leq n-k} \max\left\{|(A_k^{-1}B_k)_{i,j}|, \gamma_j(C_k)/\omega_i(A_k)\right\}$$

Algorithm 4: Compute strong RRQR given k

ALGORITHM 4. Compute a strong RRQR factorization, given k.

Compute $R \equiv \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} := \mathcal{R}_k(M)$ and $\Pi = I$;

while $\rho(R, k) > f$ **do**

Find i and j such that $\sqrt{(A_k^{-1}B_k)_{i,j}^2 + (\gamma_j(C_k)/\omega_i(A_k))^2} > f$;

Compute $R \equiv \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} := \mathcal{R}_k(R\,\Pi_{i,j+k})$ and $\Pi := \Pi\,\Pi_{i,j+k}$;

endwhile;

- Algorithm 4 is equivalent to Algorithm-3. $\det(\overline{A}_k)/\det(A_k)$ is replaced with $\rho(R, k)$.
- It eventually halts and finds a permutation Π for which $\rho(\mathcal{R}_k(M\Pi), k) \le f$.
- This implies $|A_k^{-1}B_k|_{i,j} \le q_2(k,n)$ with $q_2(k,n) = f$ and $\sigma_i(A_k) \ge \dfrac{\sigma_i(M)}{q_1(k,n)}$, $\sigma_j(C_k) \le \sigma_{k+j}(M) q_1(k,n)$ with $q_1(k,n) = \sqrt{1 + f^2 k(n-k)}$

Algorithm 5: Compute k and a SRRQR factorization

ALGORITHM 5. Compute k and a strong RRQR factorization.

$k := 0$; $R \equiv C_k := M$; $\Pi := I$;
Initialize $\omega_*(A_k)$, $\gamma_*(C_k)$, and $A_k^{-1} B_k$;
while $\max_{1 \le j \le n-k} \gamma_j(C_k) \ge \delta$ **do**
  $j_{\max} := \operatorname{argmax}_{1 \le j \le n-k} \ \gamma_j(C_k)$;
  $k := k + 1$;
  Compute $R \equiv \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} := \mathcal{R}_k(R\, \Pi_{k,k+j_{\max}-1})$ and $\Pi := \Pi\, \Pi_{k,k+j_{\max}-1}$;
  Update $\omega_*(A_k)$, $\gamma_*(C_k)$, and $A_k^{-1} B_k$;
  while $\hat{\rho}(R, k) > f$ **do**
    Find i and j such that $\left|\left(A_k^{-1} B_k\right)_{i,j}\right| > f$ or $\gamma_j(C_k)/\omega_i(A_k) > f$;
    Compute $R \equiv \begin{pmatrix} A_k & B_k \\ & C_k \end{pmatrix} := \mathcal{R}_k(R\, \Pi_{i,j+k})$ and $\Pi := \Pi\, \Pi_{i,j+k}$;
    Modify $\omega_*(A_k)$, $\gamma_*(C_k)$, and $A_k^{-1} B_k$;
  endwhile;
endwhile;

- This algorithm is combination of Algo-1 and Algo-4.
- Instead of $\rho(R, k)$ from Algo-4 here $\hat{\rho}(R, k)$ is used.
- It halts having found k and a permutation Π for which $\rho(R, \hat{k}) \le f$ which implies $\rho(\mathcal{R}_k(M\Pi), k) \le \sqrt{2} f$
- This satisfies $q_1(k, n) = \sqrt{1 + 2f^2 k(n-k)}$ and $q_2(k, n) = \sqrt{2} f$

Efficiency of Algorithm 5

- Algorithm 1 has a flop count of $4mnk - 2k^2(m+n) + 4k^3/3$.

Item	Flops
Updating procedure	$2(2m-k)(n-k)$
Reduction procedure	$3k(2n-k)$
Modifying procedure	$4m(n-k)+k^2$
Finding $\hat{\rho}(R,k)$	$2k(n-k)$

Table: Flop Count for Algorithm 5

- Let, k_f be the final value of k when Algorithm 5 terminates and the total number of interchanges is denoted by t_{k_f}. t_{k_f} is bounded by $k_f log_f \sqrt{n}$. Then the total cost is about $2mk_f(2n-k_f) + 4t_{k_f}n(m+n)$. When f is taken to be small power of n the total cost is $\mathcal{O}(mnk_f)$.

Efficiency of Algorithm 5

Comparing Algorithms 1 and 5

- When $m >> n$, Algorithm 5 is almost as fast as Algorithm 1.
- When $m \approx n$ Algorithm 5 is about 50% more expensive.

Final Remark

Despite the fact that Algorithm 5 is providing more information while covering a broader class of matrices, the efficiency is almost the same to Algorithm 1.

Numerical Stability of Algorithm 5

- Since we updated and modified $\omega_*(A_k), \gamma_* C_k$ and ${A_k}^{-1}B_k$,rather than recompute them, we might expect some loss of accuracy.
- But Since we use these quantities for deciding which pairs of columns to interchange, Algorithm 5 could be only unstable if they are extremely inaccurate.
- We give an upper bound of $\rho(R, k)$ during interchanges. Since the bound grows slowly with k, A_k can never be extremely ill conditioned provided that $\sigma_k(M)$ is not very much smaller than $||M||_2$. This implies that ${A_k}^{-1}B_k$ cannot be too inaccurate.

References

Minh Gu and Stanley C. Eisenstat (1996)
Efficient Algorithms for Computing A Strong Rank-Revealing OR Factorization
SIAM J. SCI. COMPUT Vol. 17, No. 4, pp. 848-869, July 1996

Shivkumar Chandrasekaran and Ilse C.F. Ipsen (1994)
On Rank Revealing Factorisations
SIAM J. MATRIX ANAL. APPL. Vol. 15, No. 2, pp. 592-622, April 1994

Xin Xing (2019)
Strong Rank Revealing QR decomposition
(https://www.mathworks.com/matlabcentral/fileexchange/69139-strong-rank-revealing-qr-decomposition), MATLAB Central File Exchange. Retrieved July 1, 2021.

Thank You