COLUMN SORTING IN PYGSVD

DOW DRAKE
UPDATED DECEMBER, 2019

1. Introduction

This document is intended to document and clarify the details of column ordering for the Python pygsvd library. For convenience, an excerpt from the LAPACK documentation is reproduced in the next section.

2. LAPACK Documentation Excerpt

The generalized singular value decomposition of an $m \times n$ matrix A and a $p \times n$ matrix B is given by the pair of factorizations

$$A = U\Sigma_1[0, R]Q^T \text{ and } B = V\Sigma_2[0, R]Q^T$$

The matrices in these factorizations have the following properties:

(1) U is $m \times m$, V is $p \times p$, Q is $n \times n$ and all three matrices are orthogonal. If A and B are complex, these matrices are unitary instead of orthogonal, and Q^T should be replaced by Q^H in the pair of factorizations.
(2) R is $r \times r$, upper triangular and nonsingular. $[0, R]$ is $r \times n$ (in other words, the 0 is an $r \times (n - r)$ zero matrix. The integer r is the rank of $\begin{pmatrix} A \\ B \end{pmatrix}$, and satisfies $r \leq n$.
(3) Σ_1 is $m \times r$, Σ_2 is $p \times r$, both are real, nonnegative and diagonal, and $\Sigma_1^T\Sigma_1 + \Sigma_2^T\Sigma_2 = I$. Write $\Sigma_1^T\Sigma_1 = \text{diag }(\alpha_1^2, \dots, \alpha_r^2)$ and $\Sigma_2^T\Sigma_2 = \text{diag }(\beta_1^2, \dots, \beta_r^2)$, where α_i and β_i lie in the interval from 0 to 1. The ratios $\alpha_1/\beta_1, \dots, \alpha_r/\beta_r$ are called the **generalized singular values of the pair A, B. If $\beta_i = 0$, then the generalized singular value α_i/β_i is infinite**

Σ_1 and Σ_2 have the following detailed structures, depending on whether $m - r \geq 0$ or $m - r < 0$. In the first case, $m - r \geq 0$, then

$$\Sigma_1 = \begin{array}{c c} & \begin{array}{cc} k & l \end{array} \\ \begin{array}{r} k \\ l \\ m-k-l \end{array} & \begin{pmatrix} I & 0 \\ 0 & C \\ 0 & 0 \end{pmatrix} \end{array} \quad \text{and} \quad \Sigma_2 = \begin{array}{c c} & \begin{array}{cc} k & l \end{array} \\ \begin{array}{r} l \\ p-l \end{array} & \begin{pmatrix} 0 & S \\ 0 & 0 \end{pmatrix} \end{array}$$

Here l is the rank of B, $k = r - l$, C and S are diagonal matrices satisfying $C^2 + S^2 = I$, and S is nonsingular. We may also identify $\alpha_1 = \cdots = \alpha_k = 1, \alpha_{k+i} = c_{ii}$ for $i = 1, \dots, l$, $\beta_1 = \cdots = \beta_k = 0$, and $\beta_{k+i} = s_{ii}$ for $i = 1, \dots, l$. Thus the first k generalized singular values $\alpha_1/\beta_1, \dots, \alpha_k/\beta_k$ are infinite, and the remaining l generalized singular values are finite.

In the second case, when $m - r < 0$,

$$\Sigma_1 = \begin{array}{c c} & \begin{array}{ccc} k & m-k & k+l-m \end{array} \\ \begin{array}{r} k \\ m-k \end{array} & \begin{pmatrix} I & 0 & 0 \\ 0 & C & 0 \end{pmatrix} \end{array} \quad \text{and} \quad \Sigma_2 = \begin{array}{c c} & \begin{array}{ccc} k & m-k & k+l-m \end{array} \\ \begin{array}{r} m-k \\ k+l-m \\ p-l \end{array} & \begin{pmatrix} 0 & S & 0 \\ 0 & 0 & I \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

Again, l is the rank of B, $k = r - l$, C and S are diagonal matrices satisfying $C^2 + S^2 = I$, S is nonsingular, and we may identify $\alpha_1 = \cdots = \alpha_k = 1, \alpha_{k+i} = c_{ii}$ for $i = 1, \ldots, m-k$, $\alpha_{m+1} = \cdots = \alpha_r = 0$, $\beta_1 = \cdots = \beta_k = 0$, and $\beta_{k+i} = s_{ii}$ for $i = 1, \ldots, m-k$, and $\beta_{m+1} = \cdots = \beta_r = 1$. Thus the first k generalized singular values $\alpha_1/\beta_1, \ldots, \alpha_k/\beta_k$ are infinite, and the remaining l generalized singular values are finite.

3. Column Sorting Performed in pygsvd

The GSVD implementation in LAPACK returns singular values in arrays of length n. But in all cases, we are only interested in the first r values. These are precisely the α_i and β_i, $i = 1, \ldots r$ values described in the two cases above.

The problem addressed by column sorting is that although α_i corresponds to β_i for each i and the columns of U, V and X are ordered correspondingly, no absolute ordering is guaranteed by the LAPACK algorithm. The authors of pygsvd have chosen to provide the α values in descending order and the β values in ascending order, resulting in generalized singular values in descending order, which corresponds to the conventional ordering for the standard SVD algorithm.

As can be seen in the structures of Σ_1 and Σ_2 above, the non-trivial α and β values that require sorting begin at index $i = k+1$ and continue to either $i = k+l$ in the first case or $i = k+m-k = m$ in the second case. When the sort operation is limited to this minimal set, we can guarantee that no indexing error will occur during sorting of columns of U, V or X.

We next develop a strategy for re-ordering columns to accomplish the desired sorting of singular values. We note that if X is defined by

$$X = Q \begin{pmatrix} I & 0 \\ 0 & R^{-1} \end{pmatrix}$$

That is, if X is the inverse transpose of the default X returned by pygsvd, then

$$U^T A X = U^T U \Sigma_1 \begin{pmatrix} 0 & R \end{pmatrix} Q^T Q \begin{pmatrix} I & 0 \\ 0 & R^{-1} \end{pmatrix} = \Sigma_1 \begin{pmatrix} 0 & R \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & R^{-1} \end{pmatrix} = \Sigma_1 \begin{pmatrix} 0 & I \end{pmatrix}$$

Similarly

$$V^T B X = \Sigma_2 \begin{pmatrix} 0 & I \end{pmatrix}$$

Thus we see that if $r < n$, we have $n-r$ columns of zeros in these products preceding the structures of Σ_1 and Σ_2. In the first case, $m - r \geq 0$, we get

$$U^T A X = \begin{array}{c|ccc|} & n-r & k & l \\ k & 0 & I & 0 \\ l & 0 & 0 & C \\ m-k-l & 0 & 0 & 0 \end{array} \quad \text{and} \quad V^T B X = \begin{array}{c|ccc|} & n-r & k & l \\ l & 0 & 0 & S \\ p-l & 0 & 0 & 0 \end{array}$$

In the second case, when $m - r < 0$,

$$U^T A X = \begin{array}{c|cccc|} & n-r & k & m-k & k+l-m \\ k & 0 & I & 0 & 0 \\ m-k & 0 & 0 & C & 0 \end{array} \quad \text{and}$$

$$V^T B X = \begin{array}{c|cccc|} & n-r & k & m-k & k+l-m \\ m-k & 0 & 0 & S & 0 \\ k+l-m & 0 & 0 & 0 & I \\ p-l & 0 & 0 & 0 & 0 \end{array}$$

To see clearly how columns must be reordered, it is useful to write U^TAX as

$$\begin{pmatrix} u_1^T \\ u_1^T \\ \vdots \\ u_m^T \end{pmatrix} A \begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix} = \begin{pmatrix} u_1^TAx_1 & u_1^TAx_2 & \cdots & u_1^TAx_n \\ u_2^TAx_1 & u_2^TAx_2 & \cdots & u_2^TAx_n \\ \vdots & \vdots & \vdots & \vdots \\ u_m^TAx_1 & u_m^TAx_2 & \cdots & u_m^TAx_n \end{pmatrix}$$

and V^TBX as

$$\begin{pmatrix} v_1^T \\ v_1^T \\ \vdots \\ v_m^T \end{pmatrix} B \begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix} = \begin{pmatrix} v_1^TBx_1 & v_1^TBx_2 & \cdots & v_1^TBx_n \\ v_2^TBx_1 & v_2^TBx_2 & \cdots & v_2^TBx_n \\ \vdots & \vdots & \vdots & \vdots \\ v_p^TBx_1 & v_p^TBx_2 & \cdots & v_p^TBx_n \end{pmatrix}$$

Pre-multiplication of a square matrix A by a compatible permutation matrix P^T reorders the rows of A and post-multiplication of A by P reorders the corresponding columns. So in particular, if D is a diagonal matrix, P^TDP will re-order the diagonal elements of D. In the case, for example where U, A and X are $n \times n$, and $r = l = n$ we could let $\tilde{U} = UP$, $\tilde{X} = XP$ and get

$$\tilde{U}^TA\tilde{X} = (UP)^TAXP = P^TU^TAXP = P^T\Sigma_1P$$

So we see that the desired ordering of the diagonal entries can be achieved by an appropriate ordering of the columns of U and A. Now consider the two representations of U^TAX above. In the case $m - r > 0$. We wish to permute columns of U and columns of X in such a way that the diagonal entries of C are in descending order. The top left element of C corresponds to $u_{k+1}^TAx_{n-r+k+1}$. The desired entries of C can be sorted performing the desired sort on the correct ranges of columns of U and X. For U, we see that we must skip the first k columns and sort the next l columns. For X, we must skip the first $n-r+k$ columns and sort the next l columns. Since X always has n columns, this is equivalent to sorting its last l columns. To sort S, we must sort the first l values of V and the last l columns of X, which agrees with the sort already performed on X. These operations correspond to the Python code:

```
if m - r >= 0:
    ix = np.argsort(C[k:r])[::-1]  # sort l values
    X[:, -l:] = X[:, -l:][:, ix]
    if compute_uv[0]:
        U[:, k:k+l] = U[:, k:k+l][:, ix]
    if compute_uv[1]:
        V[:, :l] = V[:, :l][:, ix]
```

We also note that the given structures of U^TAX and V^TBX do not place the singular values on the diagonal in cases when the rank r of the stacked AB matrix is less than n and/or in cases when $k = r - l > 0$, the difference between the rank of AB and the rank of B. In most cases, this can be corrected by rotating the columns of X to the left by $n - r$ and rotating the columns of V down by k. This rotation is impossible only in the case when $k > 0$ and $p < r$, i.e. the number of rows of B is less then the rank of AB. These operations correspond to the Python code:

```
if n-r > 0:
    X = np.roll(X, r-n, axis=1)
if k > 0 and p >= r:
    V = np.roll(V, k, axis=1)
```

4. Edge case discussion and comparison with Matlab

As mentioned above, it is convenient to place the singular on the diagonals of U^TAX and V^TBX. In most cases, this is performed by the roll operations above. But there is an edge case,

$k > 0$ and $p < r$ such that this cannot be done for V^TBX. Similarly, the Matlab implementation also tries to place the singular values on the diagonal, but for Matlab also, it is not always possible.

The implementation in Matlab uses the opposite convention and sorts the α values in ascending order and the β values in descending order, resulting in generalized singular values in ascending order. Because of this different convention, in Matlab, the singular values in the product V^TBX can always be placed on the diagonal. However, in the case $m < n$, the singular values in the product U^TAX cannot be placed on the diagonal by Matlab. These edge cases are illustrated by the following examples.

Using pygsvd for a case with $k = 1 > 0$ and $p = 2 < n = 3$:

```
A = array([[4, 1, 8],
           [7, 1, 0],
           [3, 0, 5]])

B = array([[0, 5, 6],
           [0, 6, 5]])

C,S,X,U,V = gsvd(A,B,X1=True)
C
array([1.        , 0.98318738, 0.07632218])
S
array([0.        , 0.1825995 , 0.99708321])
U.T@A@X
array([[ 1.        , -0.        , -0.        ],
       [ 0.        ,  0.98318738, -0.        ],
       [ 0.        , -0.        ,  0.07632218]])
V.T@B@X
array([[ 0.        ,  0.1825995 , -0.        ],
       [ 0.        ,  0.        ,  0.99708321]])
```

We note that this off-diagonal result for V^TBX cannot be corrected by rotating the columns of X to the left because that would also have the unintended side affect of moving the singular values of U^TAX off the diagonal.

Using Matlab for the same A and B, we do get the singular values on the diagonal:

```
  >> [U,V,X,C,S] = gsvd(A,B)
C =

    0.0763         0         0
         0    0.9832         0
         0         0    1.0000



S =

    0.9971         0         0
         0    0.1826         0
```

However, if we reverse the roles of A and B, we find in Matlab

```
  B =

     4     1     8
     7     1     0
     3     0     5

A =

     0     5     6
     0     6     5

>> [U,V,X,C,S] = gsvd(A,B)

C =

         0    0.1826         0
         0         0    0.9971


S =

    1.0000         0         0
         0    0.9832         0
         0         0    0.0763
```

We see that the singular values do not lie on the diagonal of C. We can check that the C matrix returned by Matlab's gsvd is the same as $U^T A X_1$, where $X_1 = X^{-T}$

```
>> X1=inv(X')

X1 =

   -0.1162   -0.0634    0.0183
         0   -0.1275   -0.1291
         0    0.1296    0.0015

>> U'*A*X1

ans =

         0    0.1826   -0.0000
         0   -0.0000    0.9971
```

Finally we show that for this A and B, the pygsvd algorithm *does* place the singular values on the diagonals.

```
In [4]: A
Out[4]:
array([[0, 5, 6],
       [0, 6, 5]])

In [5]: B
Out[5]:
array([[4, 1, 8],
       [7, 1, 0],
       [3, 0, 5]])
```

```
In [6]: C,S,X,U,V = gsvd(A,B,X1=True)

In [7]: U.T@A@X
Out[7]:
array([[ 0.99708321, -0.        , -0.        ],
       [-0.        ,  0.1825995 , -0.        ]])

In [8]: V.T@B@X
Out[8]:
array([[ 0.07632218,  0.        , -0.        ],
       [-0.        ,  0.98318738, -0.        ],
       [-0.        ,  0.        ,  1.        ]])
```

In conclusion, we see that the pygsvd implementation is correct, given its sorting convention, in the same way that the Matlab implementation is correct.