QR Decomposition for Column Sub-sampled Matrices

Srijesh Sudarsanan

2018/11/10

1 Introduction

We seek to obtain the QR decomposition of a matrix formed by random column sub-sampling(selection) of a larger matrix whose QR decomposition is known. The intended application of the sub-sampled QR is to solve Least-Squares problems for the corresponding selected variables by Forward-Backward substitution.

2 Problem Statement

Let $\mathbf{D} \in \mathbb{C}^{m \times n}$ s.t $m \geq n$. QR decomposition of $\mathbf{D} = \mathbf{QR}$ s.t $\mathbf{Q} \in \mathbb{C}^{m \times m}$ is unitary and $\mathbf{R} \in \mathbb{C}^{m \times n}$ is upper triangular.

Now, let $\mathbf{D_s} \in \mathbb{C}^{m \times k}$ s.t $k \leq n$ be the matrix formed by selecting columns from $\mathbf{D}$ in ascending order of numbering (the subscript "**s**" represents column sub-sampling of $\mathbf{D}$).

We require $\mathbf{Q_s} \in \mathbb{C}^{m \times m}$ and $\mathbf{R_s} \in \mathbb{C}^{m \times k}$ s.t $\mathbf{D_s} = \mathbf{Q_s R_s}$.

Since the columns of $\mathbf{Q}$ already form the orthonormal basis for the space $\mathbb{C}^n$, we can use $k \leq n$ columns of $\mathbf{Q}$ to form the vectors of $\mathbf{D_s}$ using an appropriate post-multiplication matrix.

Moreover, given that columns of $\mathbf{R}$ when projected onto the basis vectors of $\mathbf{Q}$ form the columns of $\mathbf{D}$, $\mathbf{R_s}$, which is a selection of the appropriate columns of $\mathbf{R}$ can serve as the required post multiplication matrix to form the columns of $\mathbf{D_s}$ when similarly projected onto $\mathbf{Q}$.

$$\mathbf{D_s} = \mathbf{Q R_s} \tag{1}$$

We are interested in using the result above to solve the least-squares problem

$$\mathbf{D_s h} = \mathbf{y} \tag{2}$$

Through the QR decomposition, $\mathbf{D_s^H D_s} = \mathbf{R_s^H R_s}$. Pre-multiplying (2) by $\mathbf{D_s^H}$ we get,

$$(\mathbf{D_s^H D_s})\mathbf{h} = (\mathbf{R_s^H R_s})\mathbf{h} = \mathbf{D_s^H y} \tag{3}$$

$$\hat{\mathbf{h}} = \mathbf{D_s^\dagger y} \tag{4}$$

where,

$$\mathbf{D_s}^\dagger = (\mathbf{R_s^H R_s})^{-1}\mathbf{D_s^H y} = (\mathbf{D_s^H D_s})^{-1}\mathbf{D_s^H y} \tag{5}$$

To avoid computing the inverse of $\mathbf{D_s^H D_s}$, we would prefer if $\mathbf{R_s}$ and $\mathbf{R_s^H}$ are upper and lower triangular respectively, in order to Forward-Backward solve for $\hat{\mathbf{h}}$. When the number of columns selected $\mathbf{k = n}$, (i.e) $\mathbf{D_s = D}$, this obviously holds true ($\because \mathbf{R_s = R}$), but in other cases ($\mathbf{k < n}$), this might not be the case.

Let us begin by examining the dimensions of $\mathbf{R}$ (and $\mathbf{R^H}$), $\mathbf{R_s}$ (and $\mathbf{R_s^H}$) in the scenarios listed below:

1. $\mathbf{D} \in \mathbb{C}^{m \times n}$ and $m = n$. (i.e) $\mathbf{D}$ is square, with as many equations as unknowns
2. $\mathbf{D} \in \mathbb{C}^{m \times n}$ and $m > n$. (i.e) $\mathbf{D}$ is tall and skinny, an overdetermined system
3. $\mathbf{D} \in \mathbb{C}^{m \times n}$ and $m < n$. (i.e) $\mathbf{D}$ is short and fat, an underdetermined system

3 Dimensions of R and $\mathbf{R_s}$

The dimensions of the matrix $\mathbf{R_s}$ will decide whether or not we can directly apply Forward-Backward solving.

3.1 Case 1: $\mathbf{m = n}$

When $m = n$,

There is exactly **one** column in $\mathbf{R}$, column n where $\mathbf{r_i} = [\mathbf{R_{data}}]$, (i.e) column with no zeros.

Thus, $\mathbf{R_s} = \{\mathbf{rs_1}, \mathbf{rs_2}, ...\mathbf{rs_k}\}$ s.t $\mathbf{rs_i} \in \mathbb{C}^{m \times 1}$, $\mathbf{i} \in [1, \mathbf{k}]$

$\mathbf{rs_i} = \begin{bmatrix} \mathbf{R_{data}} \\ \mathbf{0} \end{bmatrix}$ s.t $\mathbf{R_{data}} \in \mathbb{C}^{i \times 1}$ and $\{\mathbf{0}\} \in \mathbb{C}^{n-i \times 1}$. If $k = n^{th}$ column is selected from $\mathbf{R}$, $\mathbf{rs_k} = [\mathbf{R_{data}}]$.

3.2 Case 2: $\mathbf{m > n}$

When $m > n$,

There is **no** column in $\mathbf{R}$, where $\mathbf{r_i} = [\mathbf{R_{data}}]$, (i.e) all columns have trailing zeros.

$\mathbf{R}$ is of the form $\begin{bmatrix} \mathbf{R_1} \\ \mathbf{0} \end{bmatrix}$ where $\mathbf{R_1}$ is square and upper triangular.

Thus, $\mathbf{R_s} = \{\mathbf{rs_1}, \mathbf{rs_2}, ...\mathbf{rs_k}\}$ s.t $\mathbf{rs_i} \in \mathbb{C}^{m \times 1}$, $\mathbf{i} \in [1, \mathbf{k}]$

$\mathbf{rs_i} = \begin{bmatrix} \mathbf{R_{data}} \\ \mathbf{0} \end{bmatrix}$ s.t $\mathbf{R_{data}} \in \mathbb{C}^{i \times 1}$ and $\{\mathbf{0}\} \in \mathbb{C}^{n-i \times 1}$.

3.3 Case 3: $\mathbf{m < n}$

When $m < n$,

There are $n - m$ columns in $\mathbf{R}$, where $\mathbf{r_i} = [\mathbf{R_{data}}]$, (i.e) $n - m$ columns with no zeros.

$\mathbf{R}$ is of the form $\begin{bmatrix} \mathbf{R_1} & \mathbf{R_2} \end{bmatrix}$ where $\mathbf{R_1}$ is square $\in \mathbb{C}^{m \times m}$ and upper triangular, while $\mathbf{R_2}$ is rectangular $\in \mathbb{C}^{m \times n-m}$.

Thus, $\mathbf{R_s} = \{\mathbf{rs_1}, \mathbf{rs_2}, ...\mathbf{rs_k}\}$ s.t $\mathbf{rs_i} \in \mathbb{C}^{m \times 1}$, $\mathbf{i} \in [1, \mathbf{k}]$

$\mathbf{rs_i} = \begin{bmatrix} \mathbf{R_{data}} \\ \mathbf{0} \end{bmatrix}$ s.t $\mathbf{R_{data}} \in \mathbb{C}^{i \times 1}$ and $\{\mathbf{0}\} \in \mathbb{C}^{n-i \times 1}$.

4 Reformating of $\mathbf{R_s}$(and $\mathbf{R_s^H}$) matrices

From the information in the preceding section, in all 3 cases, we are not guaranteed triangular $\mathbf{R_s}$ matrices. They are "Upper-Hessenberg" matrices.

- In Case 1 (**3.1**) and Case 2 (**3.2**) , we are guaranteed that no row in $\mathbf{R_s}$ will have its leading non-zero entry in the same column as the one preceding it. (i.e) It can be row reduced to an echelon form for Gauss-elimination solving. This is to be expected, as it is equivalent to picking a subset of unknown variables to solve with the same number of equations.

  However, the same is not true for $\mathbf{R_s^H}$, which cannot be row-reduced (It would have to be column reduced, which is not legal for the way in which the system is defined).

  Thus, we would have to triangularize $\mathbf{R_s}$ first, to ensure we can Forward-Backward solve as intended.

- In Case 3 (**3.3**), we might end up having an $\mathbf{R}$ that is certainly not reducible to an echelon form.

  In this case, we might have to segment our initial matrix $\mathbf{D}$ into $\begin{bmatrix} \mathbf{D_1} & \mathbf{D_2} \end{bmatrix}$ s.t $\mathbf{D_1} \in \mathbb{C}^{m \times c_1}$ and $\mathbf{D_2} \in \mathbb{C}^{m \times c_2}$, $c_1 + c_2 = n$ and $c_1, c_2 \leq m$. Computing the **QR** decomposition on these two matrices separately will give us results in the cases 1 and(or) 2 as discussed above.

5 Rank-1 updates to QR decomposition

A method to "triangularize" the upper-hessenberg $\mathbf{R_s}$ matrices is through rank-1 updates to the existing **QR** decomposition by deleting and(or) adding columns as necessary.

Column updates to the **QR** can be done as follows:

- **Deleting a column** can be done by first removing the unnecessary column from $\mathbf{R}$ to get $\mathbf{R_s}$, and then applying a sequence of Givens rotations to the vectors of $\mathbf{R_s}$ to remove the unwanted sub-diagonal elements. This approach has a complexity of $\mathbf{O(n^2)}$ per update.
- **Appending a column** is similarly done by first inserting the necessary column from $\mathbf{R}$ into $\mathbf{R_s(old)}$ to get $\mathbf{R_s(new)}$. The new matrix is upper triangular, except for the new column that was added in. Applying a sequence of Givens rotations to the vectors of $\mathbf{R_s(new)}$ can remove the unwanted sub-diagonal elements. This approach has a complexity of $\mathbf{O(mn)}$ per update.

5.1 Algorithmic Considerations:

A decision is to be made on whether we want to perform deletions on $\mathbf{R}$ or modifications on the current $\mathbf{R_s}$ to produce the next one.

- An advantage of the former is that the operations can be performed in parallel at the cost of providing access to $\mathbf{R}$ in memory.

  The computational cost could be fixed at **No. of sub-matrices** $\times$ $\mathbf{n-k}$ deletions $\times$ $\mathbf{O(n^2)}$ (assuming no repetitions of sub-matrices) to select $\mathbf{k}$ columns of $\mathbf{R} \in \mathbb{C}^{m \times n}$. [1]
- Efficient computation of the latter depends on ordering the generation of the sub-matrices in a manner that ensures minimal differences between them.

This could at best do **No. of sub-matrices** $\times$ $\mathbf{O(n^2)}$, provided $m \geq n$ (assuming no repetitions of sub-matrices and only one deletion between sub-matrices).

- The choice of algorithm to "triangularize" is also crucial. If the rotations can be performed by the Householder algorithm, it might be a contender to the Givens method.

5.2 Givens Rotations for Rank-1 updates to R

The use of Givens matrices to introduce zeros into a vector by rotating it in space can aid us in "triangularizing" our sub-sampled $\mathbf{R_s}$ matrix if it is not already so. While the popular Householder reflections are useful to zero out vectors on a "grand-scale", (i.e) all but the first element, Givens matrices can be used to selectively add zeros since they operate only on two rows at a time.

Moreover, since different rows of the matrix can be operated on independently by this method, computations can be parallelized on compatible computing platforms.

5.2.1 Algorithm for generation of the rotation matrices

The standard form of the Givens matrix in 2 dimensions is

$$\mathbf{G} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \tag{6}$$

where $c^2 + s^2 = 1$. Equivalently, this corresponds to $c = Cos(\theta)$ and $s = Sin(\theta)$. Pre-multiplying a vector $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ with $\mathbf{G^H}$ has the effect of rotating $\mathbf{X}$ anti-clockwise by the angle θ. The utility of Givens matrices in the computation of the QR decomposition is to introduce zeros necessary to form the triangular $\mathbf{R}$ matrix [1]. The application is illustrated as:

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^H \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} r_1 \\ 0 \end{bmatrix} \tag{7}$$

where, if $|x_2| < |x_1|$

$$\tau = -x_1/x_2$$

$$c = x_1/\sqrt{1+\tau^2}$$

$$s = 1/\sqrt{1+\tau^2}$$

else,

$$\tau = -x_2/x_1$$

$$c = 1/\sqrt{1+\tau^2}$$

$$s = x_2/\sqrt{1+\tau^2}$$

and,

$$r_1 = \|\mathbf{X}\|_2$$

The above computation requires, in total, a minimum of:

1. 1 comparison
2. 4 multiplies (2 divisions, 2 multiplications)
3. 1 addition
4. 1 square-root

Listing 1: **MATLAB** script to generate the Givens rotation matrix

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% @file   givens_rotate.m                                                  %
% @brief  Givens rotation on a 2x1 vector by a 2x2 rotation matrix        %
%         Result rotated vector is a 2x1 vector with rotated(2) = 0       %
% @author srijeshs                                                         %
% @date   11/12/2018                                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Givens_Mat = givens_rotate(vector)
% Let vector = | A |
%              | B | 2x1
A = vector(1);
B = vector(2);

% Obtain the Givens rotation matrix = [Cos , Sin ; -Sin , Cos]
if(B ~= 0)
    if(abs(B) > abs(A))
        A_by_B = -A/B;
        hypot  = 1/sqrt(1 + (A_by_B * A_by_B));
        Cos_theta = hypot * A_by_B;
        Sin_theta = hypot;
    else
        B_by_A = -B/A;
        hypot  = 1/sqrt(1 + (B_by_A * B_by_A));
        Sin_theta = hypot * B_by_A;
        Cos_theta = hypot;
    end
else
%      No need to rotate since B == 0. Thus, theta = 0
%      Cos_theta = 1; % Cos(0)
%      Sin_theta = 0; % Sin(0)
    Givens_Mat = eye(2);
    return;
end

Givens_Mat = [Cos_theta -Sin_theta; Sin_theta Cos_theta];
end
```

The script returns the transpose of the Givens matrix so that it can be directly multiplied to the vector input. [1]

Through rank-1 updates, the number of columns to"update" changes based on whether we are deleting or appending columns. The choice depends on the scenario motivating the updates. Consider the following scenarios:

5.2.2 Case 1: n_1 common and $k - n_1$ unique columns

Let the matrix $\mathbf{D_s}$ be constructed by sampling $\mathbf{D}$ such that $\mathbf{D_s} = [\mathbf{d_1}, \mathbf{d_2}, ...\mathbf{d_{n_1}}, \mathbf{d_{n_2}}, \mathbf{d_{n_3}}...\mathbf{d_k}]$, where $\mathbf{d_i}$ are columns of $\mathbf{D} \in \mathbb{C}^{m\times 1}$. Vectors $\mathbf{d_i}$ for $i \in [1, n_1]$ are contiguous columns of $\mathbf{D}$ and $\mathbf{d_i}$ for $i \in [n_2, k]$ are chosen at random from the remaining $n - n_1$ columns of $\mathbf{D}$.

This is a scenario where different variations of $\mathbf{D_s}$ are created by keeping n_1 columns common between

[1]Since $\begin{bmatrix} r_1 \\ 0 \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^H \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

variations and then appending $k - n_1$ random columns to it.

The solution to this scenario is proposed as follows:

1. Start by computing the standard QR of a sub-sampled matrix $\mathbf{D_{s_{common}}} \in \mathbb{C}^{m \times n_1}$ (i.e) the matrix formed with the n_1 columns of $\mathbf{D}$ that are common to all variations required.

$$\mathbf{D_{s_{common}}} = \mathbf{Q_{common}} \mathbf{R_{common}} \tag{8}$$

   where, $\mathbf{Q_{common}} \in \mathbb{C}^{m \times m}$, $\mathbf{R_{common}} \in \mathbb{C}^{m \times n_1}$ and $\mathbf{Q_{common}}$ is unitary, $\mathbf{R_{common}}$ is upper triangular. Since n_1 is typically $< n$, $\mathbf{R_{common}} = \begin{bmatrix} \mathbf{R_{n_1}} \\ \mathbf{0} \end{bmatrix}$ where $\mathbf{R_{n_1}} \in \mathbb{C}^{n_1 \times n_1}$ and upper triangular, and $\{\mathbf{0}\} \in \mathbb{C}^{m-n_1 \times n_1}$

2. For each variation, augment $\mathbf{R_{Common}}$ by appending the randomly selected columns of $\mathbf{D}$, (i.e) $\mathbf{R_{augmented}} = \begin{bmatrix} \mathbf{R_{common}} & \mathbf{D_{new}} \end{bmatrix}$, where $\mathbf{R_{augmented}}$ is an upper-Hessenberg matrix, with the first n_1 columns being triangular and the remaining $k - n_1$ columns being possibly full row-dimensioned.

3. To solve Least-Squares problems by forward-backward substitution, we require that each variation's $\mathbf{R}$ matrix, (i.e) $\mathbf{R_{augmented}}$ is upper-triangular. This can be done by $k - n_1$ rank-1 updates to the augmented matrix to "triangularize" it. This can be done by sequentially pre-multiplying $\mathbf{R_{augmented}}$ by appropriate Givens rotation matrices.

   The $\mathbf{Q}$ matrix for the sub-sampled matrix can be obtained by,

$$\mathbf{Q_{augmented}} = \mathbf{Q_{common}} \mathbf{G_1} \mathbf{G_2} \mathbf{G_3} ... \mathbf{G_z} \tag{9}$$

   where z is the total number of zeros we are required to introduce. Each Givens rotation introduces a single zero, so we need z Givens matrices in total.

   Thus, our QR decomposition for the subsampled matrix is completed as:

$$\mathbf{D_s} = \mathbf{Q_{augmented}} \mathbf{R_{augmented}} \tag{10}$$

A necessary cost of any strategy to obtain a suitable $\mathbf{R_s}$ is to "triangularize" the matrix. This can be seen as a multiple of the cost of introducing a single zero, since the Givens rotation method introduces a single zero at every iteration.

This cost is comprised of:

1. Cost of computing the required Givens matrix $\mathbf{G_{2 \times 2}}$ as in eq.(**6**)
2. Cost of the $\mathbf{G_{2 \times 2}}$ $\mathbf{X_{2 \times 1}}$ matrix-vector product where $\mathbf{X_{2 \times 1}}$ is a vector comprised of the appropriate indices of the columns that are yet to be "triangularized". (i.e) the remaining columns $[n_1, k]$ of a matrix $\mathbf{R_s}$ such that columns $[1, n_1]$ already form a triangular matrix.

Let the matrix $\mathbf{R_s} \in \mathbb{C}^{m \times n}$,

$$\text{No. of zeros in a column "j"} (\leq n) = m - j \tag{11}$$

for triangularity. Thus, the number of zeros to add per column is the sum of a series decreasing by 1 for each successive column.

For eg. if we start introducing zeros from the 5th column of a matrix $\in \mathbb{C}^{10 \times 6}$, we would require:

10 - 5 = 5 zeros for column 5
10 - 6 = 4 zeros for column 6
.
.
.
10 - 10 = 0 zeros for column 10

For the general case, starting from the $n+1^{th}$ column of a matrix $\in \mathbb{C}^{m\times n}$, to add k columns, the sum of the arithmetic series is as below:

$$\text{No. of required zeros} = \frac{k}{2} \times [2(m-n+1) - (k-1)] \tag{12}$$

The details discussed above are illustrated in the following MATLAB script.

Listing 2: **MATLAB** script to compute the complexity of Rank 1 updates for Case 5.2.2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% @file   Complexity_QR_R1Update.m                                         %
% @brief  Complexity computation of the Rank 1 QR update                   %
%                                                                          %
% I/O PARAMETERS:                                                          %
% @param[in] colStart  The column no. starting from which,                 %
%                      the new columns were added                          %
%                                                                          %
% @param[in] nRows   Number of rows in the R matrix                        %
% @param[in] nNew    Number of new columns added                           %
% @return    COMPLEXITY structure with fields:                             %
%            COMPS No. of comparisons                                      %
%            MULTS No. of multiplies                                       %
%            ADDS  No. of additions                                        %
%            SQRT  No. of square roots                                     %
%                                                                          %
% @author srijeshs                                                         %
% @date   11/12/2018                                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function COMPLEXITY = General_R1Update_Complexity(colStart,nRows,nNew)

% Print interpretation of inputs
fprintf('\nExisting columns in the R matrix =  Columns 1:%d\n',colStart-1);

% Initialization
COMPLEXITY.COMPS = 0;   % No. of comparisions
COMPLEXITY.MULTS = 0;   % No. of multiplies
COMPLEXITY.ADDS  = 0;   % No. of additions
COMPLEXITY.SQRT  = 0;   % No. of square roots
totalNewZeros    = 0;   % No. of new zeros we are going to introduce

% For each new column
for iCol = colStart:colStart+nNew-1
   ZerosToAdd = nRows - iCol;   % No. of zeros we need to introduce in this column

   % For each new zero required in this column
   for nZeros = 1:ZerosToAdd

    % Givens Matrix generation start
    COMPLEXITY.COMPS = COMPLEXITY.COMPS+1;
    COMPLEXITY.MULTS = COMPLEXITY.MULTS+4;
    COMPLEXITY.ADDS  = COMPLEXITY.ADDS+1;
    COMPLEXITY.SQRT  = COMPLEXITY.SQRT+1;
    % Givens Matrix generation end

    % MAT_VECT_MULT 2x2 Givens * 2x1 Vectors start
    nMULTS_MATVEC = (colStart +  nNew - iCol) * 4;
    nADDS_MATVEC  = (colStart +  nNew - iCol) * 2;

    COMPLEXITY.MULTS = COMPLEXITY.MULTS + nMULTS_MATVEC;
    COMPLEXITY.ADDS  = COMPLEXITY.ADDS + nADDS_MATVEC;
    % MAT_VECT_MULT 2x2 Givens * 2x1 Vectors end

    totalNewZeros = totalNewZeros+1;

   end % iCol
end % nZeros

% Print results
fprintf('A total of %d new columns were added\n%d new zero(s) introduced\nNew R''s
    dimensions = %dx%d',nNew,totalNewZeros,nRows,colStart+nNew-1);
end % function
```

The script returns only a rough theoretical estimate. Actual implementation may exploit platform-dependent computational paradigms.

5.2.3 Case 2: All columns of $\mathbf{D_s}$ chosen randomly from $\mathbf{D}$

Let the matrix $\mathbf{D_s}$ is constructed by sampling $\mathbf{D}$ such that $\mathbf{D_s} = [\mathbf{d_1}, \mathbf{d_2}, ...\mathbf{d_k}]$, where $\mathbf{d_i}$ are columns of $\mathbf{D} \in \mathbb{C}^{m\times 1}$, $\mathbf{d_i}$ for $i \in [1, k]$ are chosen at random from the n columns of $\mathbf{D}$.

In this case, we do not have the luxury of pre-computing a partial QR decomposition. Our choice of algorithm depends thus, on analysis of the similarities between each required variation of $\mathbf{D_s}$. Through careful ordering of each QR computation, we can produce all the required variants through say $\mathbf{k}$ updates to a variant to obtain the next one.

The proposed solution assumes:

1. The selected column variants are known ahead of time
2. Computation of overlaps between the variants is a permissible cost

The cost of successive updates can be represented as a traversal-cost matrix

$$\mathbf{C} = \begin{bmatrix} 0 & c_{1,2} & \dots & c_{1,nSites} \\ c_{2,1} & 0 & \dots & c_{2,nSites} \\ \vdots & \vdots & \ddots & \vdots \\ c_{nSites,1} & c_{nSites,2} & \cdots & 0 \end{bmatrix}$$

where $c_{i,j}$ represents the number of differences, or lack of overlap (in terms of columns) between the variant i and the variant j. The cost of updating at each step is given by $c_{i,j}$. We would like to traverse in a manner that minimizes this cost. The algorithm is as follows:

1. Begin with a traversal that has the minimum non-zero cost. (i.e) i to j for i,j = index(min($\mathbf{C_{i,j}}$))
2. $i_{new} = j$, and j_{new} = index(min($\mathbf{C_{i_{new},[1,nSites]}}$)) such that $j_{new} \neq$ any previous i
3. Repeat 1 and 2 till all variants have been successfully computed

In the worst case, we would have to update all columns of the matrix, and in the best case, we could get away by just deleting and/or appending one column. Based on the sequences, we could also re-order the columns of the matrix to minimize the cost of “triangularization" at the expense of data-movement.

In summary, this method's cost would be:

1. Start by performing a QR on the initial variant of the matrix with a cost of a standard QR decomposition of a matrix $\in \mathbb{C}^{m\times n}$
2. For each variant, for each new column, perform "triangularization" with a cost of

$$\text{Cost of one zero} \times \text{No. of zeros to add}$$

   as described by **eq.** 11 and 12

References

[1] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, third edition, 1996.