Gram-Schmidt and the QR algorithm

Arleth Salinas
University of Chicago, Chicago, Illinois 60637

I. INTRODUCTION

In this expository, we will explore the QR algorithm and the linear algebra ideas behind it. The QR algorithm was developed by John Francis and published in *The Computer Journal* in 1961.[1] This algorithm is powerful as it can be used to calculate the eigenvalue and eigenvectors of a matrix. We will begin by establishing the linear algebra ideas, namely the Gram-Schmidt process and QR decomposition, needed to execute the QR algorithm. We will give examples of these linear algebra ideas, then discuss an actual implementation of the QR algorithm as well as some results from experiments with the algorithm.

II. MATHEMATICAL DERIVATIONS

A. Gram-Schmidt Algorithm

Background

The Gram-Schmidt algorithm is named after Erhard Schmidt and J.P. Gram. Gram published his paper on the algorithm in 1883, and Schmidt based his version of the procedure on Schmidt's original paper. This algorithm was also independently found by P.S. Laplace, appearing in a paper he published in 1820. We will proceed to describe the Gram-Schmidt process.

The Algorithm

Let $\mathbf{X} = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ be a linearly independent system.
Gram-Schmidt constructs an orthonormal basis $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ such that

$$\text{span}\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} = \text{span}\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}.$$

For the first step of the process, initialize

$$\mathbf{v}_1 = \mathbf{x}_1, \mathbf{e}_1 = \frac{\mathbf{v}_1}{||\mathbf{v}_1||}.$$

The next step, set

$$\mathbf{v}_2 = \mathbf{x}_2 - \text{proj}_{\mathbf{v}_1}(\mathbf{x}_2), \mathbf{e}_2 = \frac{\mathbf{v}_2}{||\mathbf{v}_2||}.$$

Note, we define $\text{proj}_{\mathbf{x}}(\mathbf{v})$ as

$$\text{proj}_{\mathbf{x}}(\mathbf{v}) = \frac{\mathbf{x} \cdot \mathbf{v}}{||\mathbf{v}||^2}\mathbf{v}$$

The 3rd step, set

$$\mathbf{v}_3 = \mathbf{x}_3 - \text{proj}_{\mathbf{v}_1}(\mathbf{x}_3) - \text{proj}_{\mathbf{v}_2}(\mathbf{x}_3), \mathbf{e}_2 = \frac{\mathbf{v}_3}{||\mathbf{v}_3||}.$$

We can continue this process n times, setting,

$$\mathbf{v}_n = \mathbf{x}_n - \text{proj}_{\mathbf{v}_1}(\mathbf{x}_n) - \text{proj}_{\mathbf{v}_2}(\mathbf{x}_n) - \ldots - \text{proj}_{\mathbf{v}_n-1}(\mathbf{x}_n), \mathbf{e}_n = \frac{\mathbf{v}_n}{||\mathbf{v}_n||}$$

.

We end with vectors $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$ which are the vectors of our orthonormal basis for $\mathbf{X}$.
We can reformulate the above and see that to find $\mathbf{v}_k$, we can do the following,

$$\mathbf{v}_k = \mathbf{x}_k - \sum_{j=1}^{n-1} \text{proj}_{\mathbf{v}_j}(\mathbf{x}_k).$$

Example

Let

$$\mathbf{X} = \begin{bmatrix} 1 & 1 \\ 2 & -1 \\ -2 & 4 \end{bmatrix}.$$

We would set,

$$\mathbf{v}_1 = \begin{bmatrix} 1 & 2 & -2 \end{bmatrix}^\top$$

[FINISH EXAMPLE]

B. QR Decomposition

Background

To use the QR algorithm, we must first introduce the QR decomposition. This decomposition was developed by John Francis while making modifications to Rutishauser's algorithm for finding eigenvalues, which involved using the LR decomposition. The QR decomposition relies on knowing that a matrix can be reduced to a triangular matrix, thanks to Schur.

QR Decomposition Statement

For any matrix $\mathbf{A}$ (of order n, say) there exists a unitary matrix $\mathbf{Q}$ such that $\mathbf{A} = \mathbf{QR}$ where $\mathbf{R}$ is an upper (right) triangular matrix which has real, non-negative, diagonal elements. Moreover, $\mathbf{Q}$ is unique if $\mathbf{A}$ is non-singular.

Obtaining the QR Decomposition

Via Gram-Schmidt

One method of obtaining the QR decomposition is by using the Gram-Schmidt algorithm.
Let $\mathbf{X} = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ be a linearly independent system.
For the first step of the process, initialize

$$\mathbf{v}_1 = \mathbf{x}_1, \mathbf{e}_1 = \frac{\mathbf{v}_1}{||\mathbf{v}_1||}.$$

The next step, set

$$\mathbf{v}_2 = \mathbf{x}_2 - \mathrm{proj}_{\mathbf{v}_1}(\mathbf{x}_2), \mathbf{e}_2 = \frac{\mathbf{v}_2}{||\mathbf{v}_2||}.$$

The 3rd step, set

$$\mathbf{v}_3 = \mathbf{x}_3 - \mathrm{proj}_{\mathbf{v}_1}(\mathbf{x}_3) - \mathrm{proj}_{\mathbf{v}_2}(\mathbf{x}_3), \mathbf{e}_2 = \frac{\mathbf{v}_3}{||\mathbf{v}_3||}.$$

We can continue this process n times, setting,

$$\mathbf{v}_n = \mathbf{x}_n - \mathrm{proj}_{\mathbf{v}_1}(\mathbf{x}_n) - \mathrm{proj}_{\mathbf{v}_2}(\mathbf{x}_n) - \ldots - \mathrm{proj}_{\mathbf{v}_n - 1}(\mathbf{x}_n), \mathbf{e}_n = \frac{\mathbf{v}_n}{||\mathbf{v}_n||}$$

.

To obtain the QR decomposition, we define the matrices $\mathbf{Q}, \mathbf{R}$ as,

$$\mathbf{Q} = [\mathbf{e}_1|\mathbf{e}_2|\ldots|\mathbf{e}_n], \mathbf{R} = \begin{bmatrix} \mathbf{e}_1 \cdot \mathbf{x}_1 & \mathbf{e}_1 \cdot \mathbf{x}_2 & \mathbf{e}_1 \cdot \mathbf{x}_3 & \cdots \\ 0 & \mathbf{e}_2 \cdot \mathbf{x}_2 & \mathbf{e}_2 \cdot \mathbf{x}_3 & \cdots \\ 0 & 0 & \mathbf{e}_3 \cdot \mathbf{x}_3 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

Other methods

Other methods that can be used to obtain the QR decomposition include using Householder reflections., which involve taking a vector and rotating it about a plane or hyperplane. A benefit of using Householder reflections over the Gram-Schmidt process is, Gram-Schmidt experiences numerical instability. This is because round-off error can occur as vectors are projected onto each orthonormal basis vector.

One can also use Givens rotations. This involves rotating the matrix and zero-ing each element below the diagonal of the matrix, which forms the **R** matrix. Concatenating all of the Givens rotations forms the **Q** matrix.

C. QR Algorithm

The Algorithm

Now that we have established the QR decomposition, we can proceed to how the QR decomposition can be used in the QR algorithm.

Let $\mathbf{X} = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ be a linearly independent system.
Initialize $\mathbf{X}_1 = \mathbf{X}$.

$$\mathbf{X}_1 = \mathbf{Q}_1\mathbf{R}_1$$

$$\mathbf{R}_1\mathbf{Q}_1 = \mathbf{X}_2 = \mathbf{Q}_2\mathbf{R}_2$$

$$\mathbf{R}_2\mathbf{Q}_2 = \mathbf{X}_3 = \mathbf{Q}_3\mathbf{R}_3$$

$$\vdots$$

$$\mathbf{R}_{k-1}\mathbf{Q}_{k-1} = \mathbf{X}_k = \mathbf{Q}_k\mathbf{R}_k$$

As we repeat this process, if **A** is non-singular (has a determinant of 0) then the elements below the diagonal of $\mathbf{X}_k$ tend to 0, the elements above the diagonal tend to fixed values, and the elements on the principal diagonal then to the eigenvalues of **A**.

Example

Recall the matrix,

$$\mathbf{X} = \begin{bmatrix} 1 & 1 \\ 2 & -1 \\ -2 & 4 \end{bmatrix}.$$

[FINISH EXAMPLE, WHICH'LL BE FROM GRAM-SCHMIDT SECTION]

III. IMPLEMENTATION

A. Background

The implementation was done in the Julia programming language. We will proceed to show the code and discuss important implementation decisions made in the implementation of the QR algorithm and its precursory components.

B. Gram-Schmidt and QR Decomposition

```
"""
Compute an orthonormal basis for a full-rank, real,
square matrix of any size
Inputs:
    x - matrix to find orthonormal basis and QR decomposition of
Outputs:
    q - Q matrix in QR decomposition
    r - R matrix in QR decomposition
"""
function qr_gs_(x)
    #check if matrix meets pre-requisites
    if fullrank_square(x) == false
        throw("Matrix is not square or it is not full-rank!")
        return
    end

    #get size of matrix
    rows, cols = size(x)

    #initialize orthogonal matrix and upper triangular matrix
    q = zeros(rows, cols)
    r = zeros(rows, cols)

    #define v1 & e1, this is the first step
    # before we can begin iterating
    v1 = x[:, 1]
    e1 = v1 ./ norm(v1)

    #set the first columns of q and r
    q[:, 1] = e1
    #variable to hold last sum a_i, we begin with a_1
    a_i = dot(e1, x[:, 1])
    r[1, 1] = a_i

    #continue to iterate through gram-schmidt
    # through all columns, fill q & r
    for i = 2:cols
        #get the ith column from the x matrix and e_i
        x_i = x[:, i]
        #initialize weighted sum of projections
        projection_sum = zeros(rows)

        #calculate the v_i-th column of the orthonormal basis
        for j = 1:i-1
            curr_q = q[:, j]
            projection_sum += (dot(curr_q, x_i)/norm(curr_q)) .* curr_q
        end
        #subtract projections and normalize
        e_i = (x_i - projection_sum)
        e_i ./= norm(e_i)

        q[:, i] = e_i

        #fill in the r matrix with r_entry
        for k = 1:rows
            r[k, i] = dot(x_i, q[:, k])
        end
    end

    #return the orthonormal basis, q, and r
    return q, r
end
```

You will note that we only work with square, real, matrices. The rationale for this decisions is we know that the QR decomposition exists for any real square matrices and we can depend on the Gram-Schmidt algorithm to give us the decomposition of square matrices. Additionally, we accept only real matrices because the QR algorithm requires adjustment to converge to complex eigenvalues.John Francis introduces this adjustment as a complex shift in his second paper on the QR algorithm.

QR Algorithm

```
"""
Run the QR algorithm on a full-rank, real, square matrix x and iterate until
 eigenvalues within tolerance level are found or iteration limit has been reached
Inputs:
    x - matrix to find eigenvalues of using QR
    tol - tolerance level
    max_iters - iteration limit
Outputs:
    q - the Q matrix of x's QR decomposition
    r - the R matrix of x's QR decomposition
    true_lambdas - the true eigenvalues of x according to Julia
    qr_lambdas - the eigenvalues determined by QR algorithm
    iters - the number of iterations x ran through the QR algorithm
    errors - error between qr_lambdas and true_lambdas at each iteration
"""
function qr_eigenvals_(x, tol::Float64=default_tol, max_iters::Int64=default_maxit)
    #check if matrix meets pre-requisites
    if fullrank_square(x) == false
        throw("Matrix is not square or it is not full-rank!")
        return
    end

    #get size of matrix
    rows, cols = size(x)

    #use Julia's eigenvalue solver to get what we will consider the "true"
    # eigenvalues of x
    true_lambdas = eigvals(x)

    #initialize vector to hold lambda estimates
    qr_lambdas = zeros(cols)

    #initialize matrices q and r
    q = zeros(rows, cols)
    r = zeros(rows, cols)

    #current error, initialize iteration counter, initialize vector to hold errors
    errors = zeros(0)
    err = 1
    iters = 0

    #initialize matrix to find q r decomposition of to be original matrix x
    curr_A = x

    #find QR decomposition of x and iterate until max_iters have occured
    # or tolerance level has been satisfied
    while (err > tol) && (iters < max_iters)
        q, r = qr_gs_(curr_A)
        curr_a = r * q

        #get values along diagonal of curr_A
        qr_lambdas = diag(curr_A)
        #calculate error, using L2 norm
        err = norm(true_lambdas) - norm(qr_lambdas)
        #increment iteration counter & update error list
        iters += 1
        append!(errors, err)
    end

    #return the final qr decomposition, "true" eigenvalues,
    # estimated eigenvalues and iteration count
    return q, r, true_lambdas, qr_lambdas, iters, errors
end
```

[INSERT DISCUSSION OF IMPLEMENTATION DECISIONS HERE]

Experiments

Identity Matrix
[INSERT CODE, VISUALIZATIONS, DISCUSSION HERE]
Nice Diagonal Matrices and Noise
[INSERT CODE, VISUALIZATIONS, DISCUSSION HERE]
Matrices with complex eigenvalues
[INSERT CODE, VISUALIZATIONS, DISCUSSION HERE]

IV. REFERENCES

[1] van der Maaten, Laurens, Postma, Eric, and Herik, H., Dimensionality Reduction: A Comparative Review, Journal of Machine Learning Research, **10**