Math 2P40 Final Project
A Demonstration of the QR Algorithm
Megan Fisher and Daniel Teeuwsen

Project Type & Audience

Our project is an application which uses techniques found in real-world mathematical software to find the eigenvalues and eigenvectors of a 5x5 matrix and then show the general solution of a corresponding system of 5 differential equations. Our audience consists of linear algebra students and anyone else who may need numerical linear algebra software in order to find eigenvalues and eigenvectors of a 5x5 matrix.

Purpose

The overall purpose of our project was to generate the eigenvalues and eigenvectors of a 5x5 matrix by using the QR algorithm. We chose a 5x5 matrix because there exist relatively simple formulas for calculating the eigenvalues and eigenvectors of 2x2, 3x3, and 4x4 matrices. It should be noted that our program only works for symmetric matrices as this ensures that all eigenvalues and eigenvectors are real numbers. Another important note is that although professional software programs such as Maple use the concepts that are found in our program their algorithms are much more involved using shifts, householder transforms, and even combinations of other eigenvalue algorithms in order to create the most effecient possible path to the desired eigenvalues and eigenvectors of any matrix.

Background: a Brief history and Summary

The QR algorithm was first published in 1959 by English mathematician and computer scientist John Francis and then was introduced independently by Russian mathematician Vera Nikolaevna Kublanovskaya. In the year 2000, Jack Dongarra and Francis Sullivan included it as number 6 in their list of top 10 most influential algorithms of the 20th Century.

The way this algorithm works is that it takes a matrix A then factors it into two matrices Q and R. Q is a matrix which consists of the orthonormalized column vectors of A, and R is the matrix whose entries consist of the inner product of the A column vectors with the Q column vectors. R is always an upper diagonal matrix. The next step of the algorithm is to now multiply R*Q to obtain A2. A2 is then factored into Q2 and R2 and A3=R2*Q2. This process is repeated until An converges to a Schur form in which the eigenvalues of A are listed on the diagonal of the matrix in order of decreasing magnitude. In order to find the corresponding eigenvectors the algorithm generates the matrix M=Q*Q2...*Qn. The column vectors of the matrix M are the eigenvectors of A with the first column corresponding to the first eigenvalue and the nth column corresponding to the nth eigenvalue.

An example of this process with a 3x3 matrix can be seen below

```
> with(LinearAlgebra):
```

```
> A:=Matrix([[ 2  , 1  ,0   ],
           [ 1, 3, -1],
           [ 0, -1, 6]], datatype=float[8]);
```

$$A := \begin{bmatrix} 2. & 1. & 0. \\ 1. & 3. & -1. \\ 0. & -1. & 6. \end{bmatrix} \quad (1)$$

```
> evalf(Eigenvalues(A));
```

$$\begin{bmatrix} 1.31866935639502 + 0.\,I \\ 3.35792636751850 + 0.\,I \\ 6.32340427608648 + 0.\,I \end{bmatrix} \quad (2)$$

```
> evalf(Eigenvectors(A, output = list));
```

$$\left[\left[1.31866935639502 + 0.\,I,\ 1.,\ \left\{\begin{bmatrix} -0.820501114447383 + 0.\,I \\ 0.559032552385037 + 0.\,I \\ 0.119417446650284 + 0.\,I \end{bmatrix}\right\}\right], \left[3.35792636751850 + 0.\,I,\ 1.,\ \left\{\begin{bmatrix} -0.567219325612607 + 0.\,I \\ -0.770242078415420 + 0.\,I \\ -0.291529376375476 + 0.\,I \end{bmatrix}\right\}\right], \left[6.32340427608648 + 0.\,I,\ 1.,\ \left\{\begin{bmatrix} -0.0709940690634231 + 0.\,I \\ -0.306936061765582 + 0.\,I \\ 0.949078551093456 + 0.\,I \end{bmatrix}\right\}\right]\right] \quad (3)$$

```
> Q, R := QRDecomposition(A, fullspan) :
> A2 := R.Q :
> S1:=Q:
> Q, R := QRDecomposition(A2, fullspan):
> S2:=evalf(S1.Q):
> A3:=R.Q:
> Q, R := QRDecomposition(A3, fullspan):
> S3:=evalf(S2.Q):
> A4:=R.Q:
> Q, R := QRDecomposition(A4, fullspan):
> S4:=evalf(S3.Q):
> A5:=R.Q:
> Q, R := QRDecomposition(A5, fullspan):
> S5:=evalf(S4.Q):
> A6:=R.Q:
> Q, R := QRDecomposition(A6, fullspan):
> S6:=evalf(S5.Q):
> A7:=R.Q:
> Q, R := QRDecomposition(A7, fullspan):
> S7:=evalf(S6.Q):
> A8:=R.Q:
> Q, R := QRDecomposition(A8, fullspan):
> S8:=evalf(S7.Q):
```

```
> A9:=R.Q:
> Q, R := QRDecomposition(A9, fullspan):
> S9:=evalf(S8.Q):
> A10:=R.Q:
> Q, R := QRDecomposition(A10, fullspan):
> S10:=evalf(S9.Q):
> A11:=R.Q:
> Q, R := QRDecomposition(A11, fullspan):
> S11:=evalf(S10.Q):
> A12:=R.Q:
> Q, R := QRDecomposition(A12, fullspan):
> S12:=evalf(S11.Q):
> A13:=R.Q:
> Q, R := QRDecomposition(A13, fullspan):
> S13:=evalf(S12.Q):
> A14:=R.Q:
> Q, R := QRDecomposition(A14, fullspan):
> S14:=evalf(S13.Q):
> A15:=R.Q:
> Q, R := QRDecomposition(A15, fullspan):
> S15:=evalf(S14.Q):
> A16:=R.Q:
> Q, R := QRDecomposition(A16, fullspan):
> S16:=evalf(S15.Q):
> A17:=R.Q:
> Q, R := QRDecomposition(A17, fullspan):
> S17:=evalf(S16.Q):
> A18:=R.Q:
> Q, R := QRDecomposition(A18, fullspan):
> S18:=evalf(S17.Q):
> A19:=R.Q:
> Q, R := QRDecomposition(A19, fullspan):
> S19:=evalf(S18.Q):
> A20:=evalf(R.Q):
> Q, R := QRDecomposition(A20, fullspan):
> S20:=evalf(S19.Q):
> A21:=evalf(R.Q):
> Q, R := QRDecomposition(A21, fullspan):
> S21:=evalf(S20.Q);
```

$$S21 := \begin{bmatrix} 0.0710017215710793 & -0.567218373000382 & -0.820501109391019 \\ 0.306946452841170 & -0.770237932874384 & 0.559032557787455 \\ -0.949074617162056 & -0.291542179467161 & 0.119417448536802 \end{bmatrix} \quad \textbf{(4)}$$

```
> A22:=evalf(R.Q);
```

(5)

$$A22 := \begin{bmatrix} 6.32340427524791 & 0.0000400081312439689 & 3.93761011888597\ 10^{-27} \\ 0.0000400081312439688 & 3.35792636775209 & -1.48746295449564\ 10^{-8} \\ 0. & -1.48746295449564\ 10^{-8} & 1.31866935600000 \end{bmatrix} \quad \textbf{(5)}$$

5 Special Features

1. The first special feature that we included is the visualization of one QR step using the original Gram-Schmidt Process.
2. The second special feature showcases a neat little identity namely that $Q \cdot Q^T = I.$ where I is the identity matrix.
3. The third special feature shows the general solution for the corresponding system of five differential equations.
4. The fourth special feature generates a random symmetric matrix where the entries are random integers within [-9,9]
5. The fifth and final special feature is that the labels on the form appear and disappear depending on the buttons pressed.

Summary of Observations

Beginning this project the first task was clearly to program the Gram-Schmidt Process which we did and which we still show in our program. However, after further research we realized that the original Gram-Schmidt Process, though concise and easy to understand, was numerically unstable. This meant that if one were to implement it within an algorithm such as QR, rounding error would eventually build up and cause inaccuracies. The solution was to program the Modified Gram-Schmidt Process into the program that was to do the QR factoring.

One interesting thing we observed was that often when comparing the eigenvectors that our program generated versus those that Maple computed some of the program's vectors would be -1* Maple's vectors. We realized right away that this was not a problem as eigenvectors are not unique and that if $v1$ is an eigenvector of λ1 then some complex constant $(C1 + C2 \cdot i) \cdot v1$ is also an eigenvector.

Another observation that we made is that the Gram-Schmidt Process only works for a set of linearly independent vectors. Our program handles this by outputting zeroes for dependent vectors. For example our program handles the following matrix by showing only four eigenvectors (the other one is a zero vectors) meaning there are only three eigenvalues and eigenvectors. Notice that maple's one eigenvalue is $5.9 \cdot 10^{-16} \approx 0$ and if the eigenvalue=0 then the corresponding vector can be anything including the zero vector.

```
> M2:=Matrix([[ 1  , 2  ,2  , 2  , 2 ],
              [ 2, -3, 4, -2, 4],
              [ 2, 4, 4, -7, 4], [2, -2, -7, -3, 4],[2, 4, 4, 4, 4]],
  datatype=float[8]);
```

(6)

$$M2 := \begin{bmatrix} 1. & 2. & 2. & 2. & 2. \\ 2. & -3. & 4. & -2. & 4. \\ 2. & 4. & 4. & -7. & 4. \\ 2. & -2. & -7. & -3. & 4. \\ 2. & 4. & 4. & 4. & 4. \end{bmatrix} \quad (6)$$

```
> Eigenvectors(M2,output=list);
```

$$\left[\left[11.9199068455546 + 0.\,I,\ 1,\ \left\{\begin{bmatrix} 0.252506085573376 + 0.\,I \\ 0.384882517781037 + 0.\,I \\ 0.699150668430521 + 0.\,I \\ -0.210373891159861 + 0.\,I \\ 0.505012171146752 + 0.\,I \end{bmatrix}\right\}\right], \left[6.56947760883391 + 0.\,I,\ 1,\ \left\{\begin{bmatrix} 0.309210384761871 + 0.\,I \\ 0.0320299764369675 + 0.\,I \\ -0.394059147144157 + 0.\,I \\ 0.604678558358526 + 0.\,I \\ 0.618420769523743 + 0.\,I \end{bmatrix}\right\}\right], \left[5.85118003268340\ 10^{-16} + 0.\,I,\ 1,\ \left\{\begin{bmatrix} -0.894427190999916 + 0.\,I \\ 1.06134608448857\ 10^{-16} + 0.\,I \\ 5.30860840261739\ 10^{-17} + 0.\,I \\ 2.16863503640687\ 10^{-16} + 0.\,I \\ 0.447213595499958 + 0.\,I \end{bmatrix}\right\}\right], \left[-10.1734027237753 + 0.\,I,\ 1,\ \left\{\begin{bmatrix} 0.188247838331020 + 0.\,I \\ -0.226611682242412 + 0.\,I \\ -0.443339737177513 + 0.\,I \\ -0.758228712018430 + 0.\,I \\ 0.376495676662041 + 0.\,I \end{bmatrix}\right\}\right], \left[-5.31598173061321 + 0.\,I,\ 1,\ \left\{\begin{bmatrix} 0.0720580742701628 + 0.\,I \\ -0.894140186764493 + 0.\,I \\ 0.399193698377805 + 0.\,I \\ 0.123271599529602 + 0.\,I \\ 0.144116148540326 + 0.\,I \end{bmatrix}\right\}\right] \quad (7)$$

The following matrix is the matrix that appears when the program loads up:

```
> M:=Matrix([[ 1 , 1 ,2 , 3 , 4 ],
          [ 1, 2, 1, 2, 1],
          [ 2, 1, 3, 2, 1], [3, 2, 2, 4, 3],[4, 1, 1, 3, 5]],
  datatype=float[8]);
```

$$M := \begin{bmatrix} 1. & 1. & 2. & 3. & 4. \\ 1. & 2. & 1. & 2. & 1. \\ 2. & 1. & 3. & 2. & 1. \\ 3. & 2. & 2. & 4. & 3. \\ 4. & 1. & 1. & 3. & 5. \end{bmatrix} \tag{8}$$

```
> Eigenvalues(M);
```

$$\begin{bmatrix} 11.8163589228568 + 0.\,I \\ -1.89966577318769 + 0.\,I \\ 2.98671794304486 + 0.\,I \\ 1.61689148231031 + 0.\,I \\ 0.479697424975743 + 0.\,I \end{bmatrix} \tag{9}$$

```
> Eigenvectors(M, output = list);
```

$$\left[\left[11.8163589228568 + 0.\,I,\ 1,\ \left\{\begin{bmatrix} -0.448467527976606 + 0.\,I \\ -0.248140908146708 + 0.\,I \\ -0.319104942937786 + 0.\,I \\ -0.541727206754065 + 0.\,I \\ -0.584813333389202 + 0.\,I \end{bmatrix}\right\}\right], \left[-1.89966577318769 + 0.\,I,\ 1,\ \left\{\begin{bmatrix} -0.873545923304706 + 0.\,I \\ -0.0231530145616150 + 0.\,I \\ 0.206789495904695 + 0.\,I \\ 0.177123309092343 + 0.\,I \\ 0.402798827647375 + 0.\,I \end{bmatrix}\right\}\right], \left[2.98671794304486 + 0.\,I,\ 1,\ \left\{\begin{bmatrix} -0.106217349324851 + 0.\,I \\ 0.365308595396209 + 0.\,I \\ 0.627887428483853 + 0.\,I \\ 0.237450646559613 + 0.\,I \\ -0.636114826452275 + 0.\,I \end{bmatrix}\right\}\right], \left[1.61689148231031 + 0.\,I,\ 1,\ \right.\right. \tag{10}$$

$$\left\{\left\{\begin{bmatrix} -0.155799476261726 + 0.\,I \\ 0.655729921380014 + 0.\,I \\ -0.649201767774496 + 0.\,I \\ 0.333715971818296 + 0.\,I \\ -0.113646417785044 + 0.\,I \end{bmatrix}\right\}, \begin{bmatrix} 0.479697424975743 + 0.\,I, 1, \\ \left\{\begin{bmatrix} -0.0154529649988456 + 0.\,I \\ -0.611929675663345 + 0.\,I \\ -0.199260085432479 + 0.\,I \\ 0.712327036859061 + 0.\,I \\ -0.279622761642055 + 0.\,I \end{bmatrix}\right\} \end{bmatrix}\right\}$$

Research Sources:

Orthogonal Bases and the QR Algorithm
By: Peter J. Olver
University of Minnesota
http://www-users.math.umn.edu/~olver/aims_/qr.pdf

Elementary Linear Algebra 11th Edition Applications Version
Howard Anton Professor Emeritus, Drexel University
Chris Rorres University of Pennsylvania
Section 6.3

The QR Algorithm
Chapter 4
http://people.inf.ethz.ch/arbenz/ewp/Lnotes/chapter4.pdf

EMathHelp
Gram-Schmidt Calculator
https://www.emathhelp.net/calculators/linear-algebra/gram-schmidt-calculator/