John Russell Strauss

About

Work

Resume

QR Decomposition via Givens Rotations

Simply put, Givens rotations is a method of matrix decomposition. Just as with polynomials in Algebra, a matrix can be factored into two separate matrices and used to solve a system of equations. There are many different methods of matrix decomposition, using Given rotations being one of them.

The following code will decompose the Hilbert matrix of any dimension with very high accuracy.

To start out, we calculate the Hilbert matrix as explained in the previous problems. Again we created the identity matrix, and set both our Q and Gn to the identity to begin with.

```
// Beginning to create Givens rotations:
// The Gn matrix and the Q matrix will begin as the identity.

for (int i=0; i<n; i++){
    for (int j=0; j<n; j++){
        if (i==j){
            Gn[i][j] = 1;
            Q[i][j] = 1;
        }
        else{
            Gn[i][j] = 0;
            Q[i][j] = 0;
        }
    }
}
```

Our first step is to calculate our rotation matrices to be used in multiplying and finding each Gn. The cosine and sine squares move up and left in the matrix as we cancel out each element in the column. So for example, when we try to cancel out the bottom left element of the Hilbert matrix will start in the bottom right corner and as we move upward through the Hilbert matrix columns, the cosine and sine square in the rotation matrix will move up one row and left one column as we proceed. The rest of the elements are filled with the identity. It follows the form:

$$R(i,j,\theta) = \begin{pmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & \ddots & & & & & \\ & & & \cos\theta & & -\sin\theta & & \\ & & & & \ddots & & & \\ & & & \sin\theta & & \cos\theta & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \\ & & & & & & & & 1 \end{pmatrix}$$

The values of cosine and sine are calculated with:

$$\cos\theta = \frac{a}{\sqrt{a+b}} \qquad \sin\theta = \frac{-b}{\sqrt{a+b}}$$

where b is the element in the Hilbert matrix that we are trying to cancel out, and a is the element above it. So to do this, I first calculated a and b:

```
// The for loops that begin the Givens rotation matrices.

for (int i=0; i<n; i++) {
        for (int j=(n-1); j>i; j--) {

            a = An[j-1][i];
            b = An[j][i];
            cosX = a/(Math.sqrt(a*a+b*b));
            sinX = -b/(Math.sqrt(a*a+b*b));
```

The i for loop begins the process and goes through the rows of the Hilbert matrix while the j loop goes through the columns, moving from bottom to top. The first time through the loop, An equals our Hilbert matrix, the next time through, it will equal G1*Hilbert, or A1, and so forth. Now that we have found the values for our sinX and cosX variables, we will put them into their corrosponding places into the Gn matrix.

```
            Gn[j][j] = cosX;
            Gn[j][j-1] = sinX;
            Gn[j-1][j] = -sinX;
            Gn[j-1][j-1] = cosX;
```

Next, we multiply Gn with An to find our An+1 and print our each matrix to make sure it is being calculated correctly. Then we multiply Gn*Q. At the end of the process, this will represent Q = Gn…G3G2G1.

```
            System.out.println("G" + iteration + ":");
            printMatrix(Gn);

            An = matrixMultiplication(Gn, An);

            System.out.println("A" + iteration + ":");
            printMatrix(An);

            Q = matrixMultiplication(Gn, Q);
```

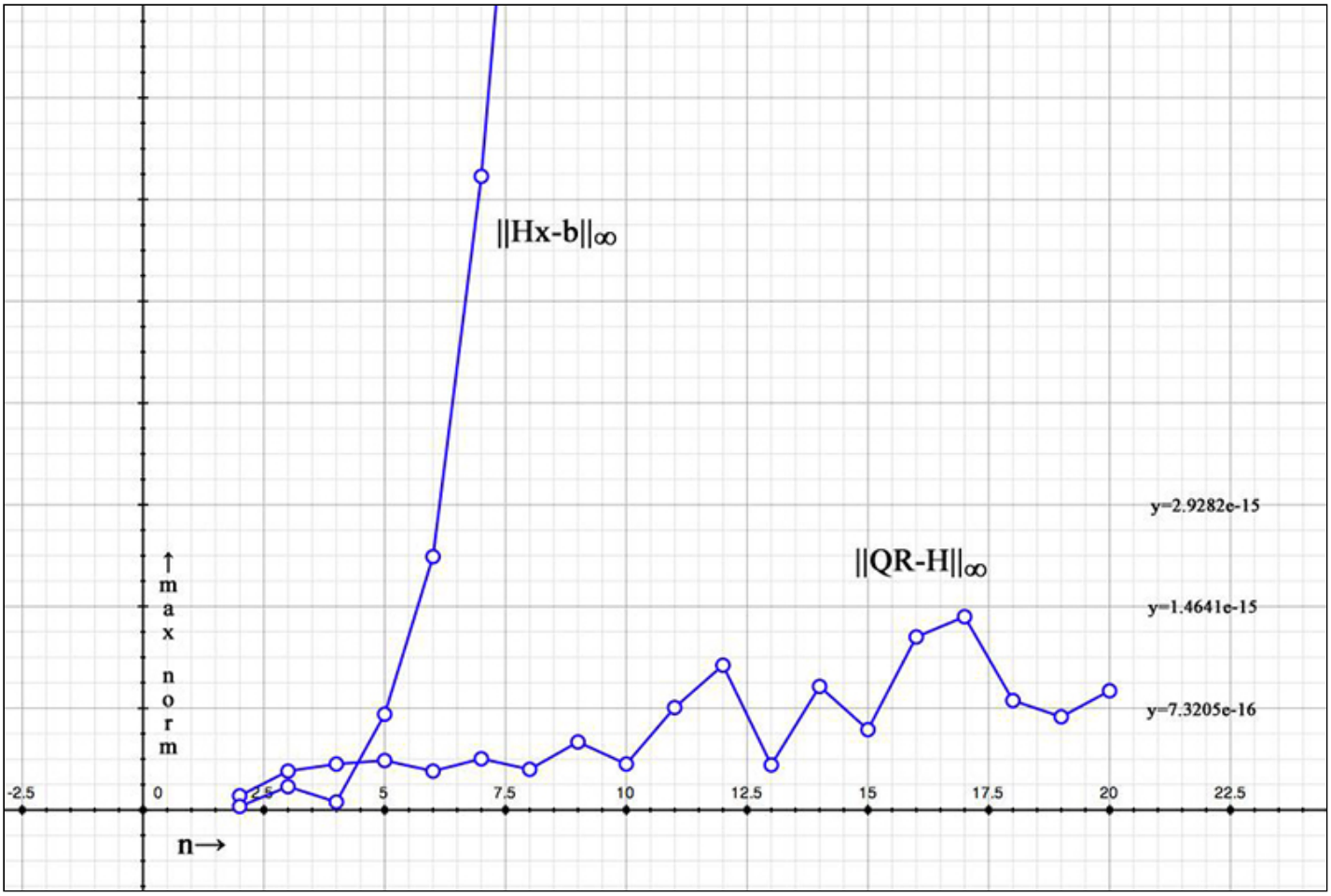
After doing this, we turn Gn back into the identity and loop back through the process. Since An has changed, it will redo the process with the new An each time through the i loop.

```
             // Turning the Gn matrix back into the identity.

            for (int ident=0; ident<n; ident++){
                for (int ident2=0; ident2<n; ident2++){
                    if (ident==ident2)
                        Gn[ident][ident2] = 1;
                    else
                        Gn[ident][ident2] = 0;
                }
            }
            iteration += 1;
        }//end j
}//end i
```

Now the only step left is to calculate the max norm of the error. The same method is used as in the previous problem.

Givens error:



As you can see, when evaluating our error, our Givens algorithm is generally very accurate, maxing out at 1.4641e-15 at n=17, although our error in compared to the max norm of Hx-b only stays accurate for the first 8 or 9 dimensions. To calculate X, we used the equation Qtb=y to solve for y, and then used back substitution in RX=y to solve for X. The larger error in maxNorm(HX-b) is because we used Python to solve for Hx-b which can not handle such large numbers as Java can. Regardless, our QR Givens rotation factorization was successful and accurate. The Givens method is overall much more stable than the other 2 methods, LU and Householders.

The code in its entirety:

```
import java.util.*;
import java.text.*;

/** ================================================
 ** ==    ALL CODE COPYRIGHT RUSSELL STRAUSS 2009   ==
 ** ================================================
 */


public class Givens {

    public static void main(String[] args) {

        /** Asking for the number of dimensions to work with the computing factorization of the Hilbert matrix,
        * formatting the numbers and matrices, and computing/printing the Hilbert matrix:
        */

        DecimalFormat fmt = new DecimalFormat("0.####");
        Scanner scan = new Scanner (System.in);
```

```
        System.out.println("In an nxn Hilbert matrix, how many dimensions, n?");
        int n = scan.nextInt();

        double[][] Hilbert = new double[n][n];
        double[][] Gn = new double[n][n];
        double[][] An = new double[n][n];
        double[][] Q = new double[n][n];

        System.out.println();
        System.out.println("Number of rows and columns in our original Hilbert matrix: " + n + " x " + n + ".");
        System.out.println("Note: Only displaying four decimal places for organization, but no accuracy is lost.");
        System.out.println();


        // THE HILBERT MATRIX:
        System.out.println("The starting Hilbert matrix:");
        System.out.println();
        for (int i=0; i<n; i++) {
            System.out.print("[ ");
            for (int j=0; j<n; j++) {
                double Hij = (1/(((double)i+1)+((double)j+1)-1));
                Hilbert[i][j] = Hij;
                An[i][j] = Hij;
                fmt.setMinimumFractionDigits(4);
                System.out.print("{");
                System.out.print(fmt.format(Hilbert[i][j]));
                System.out.print("} ");
            }
        System.out.print("]");
        System.out.println();

        }
        System.out.println("--------------------------------");

        // Beginning to create Givens rotations:

        // The Gn matrix and the Q matrix will begin as the identity.

        for (int i=0; i<n; i++){
            for (int j=0; j<n; j++){
                if (i==j){
                    Gn[i][j] = 1;
                    Q[i][j] = 1;
                }
                else{
                    Gn[i][j] = 0;
                    Q[i][j] = 0;
                }
            }
        }

        int iteration = 1;
        double a = An[0][n-2];
        double b = An[0][n-1];
        double cosX;
        double sinX;

        // The for loops that begin the Givens rotation matrices.

        for (int i=0; i<n; i++) {
                for (int j=(n-1); j>i; j--) {

                    a = An[j-1][i];
                    b = An[j][i];
                    cosX = a/(Math.sqrt(a*a+b*b));
```

```
                    sinX = -b/(Math.sqrt(a*a+b*b));

                    Gn[j][j] = cosX;
                    Gn[j][j-1] = sinX;
                    Gn[j-1][j] = -sinX;
                    Gn[j-1][j-1] = cosX;

                    System.out.println("G" + iteration + ":");
                    printMatrix(Gn);

                    An = matrixMultiplication(Gn, An);

                    System.out.println("A" + iteration + ":");
                    printMatrix(An);

                    Q = matrixMultiplication(Gn, Q);

                    // Turning the Gn matrix back into the identity.

                    for (int ident=0; ident<n; ident++){
                        for (int ident2=0; ident2<n; ident2++){
                            if (ident==ident2)
                                Gn[ident][ident2] = 1;
                            else
                                Gn[ident][ident2] = 0;
                        }
                    }
                    iteration += 1;
                }//end j
            }//end i


            System.out.println("Q:");
            printMatrix(Q);

            System.out.println("R:");
            printMatrix(An);

            Q = transpose(Q);
            double[][] answer = matrixMultiplication(Q, An);
            System.out.println("Did it work? Q x R:");
            printMatrix(answer);

            // Calculating the maximum norm of QR-H.

            answer = subtractMatrix(answer, Hilbert);
            double maxNorm = normOfInfinity(answer);
            System.out.println();
            System.out.println("The maximum norm of QR - H:");
            System.out.println(maxNorm);


        } //end of main
    } // end of Givens class
```

And its output:

```
In an nxn Hilbert matrix, how many dimensions, n?
4

Number of rows and columns in our original Hilbert matrix: 4 x 4.
Note: Only displaying four decimal places for organization, but no accuracy is lost.
```

```
The starting Hilbert matrix:

[ {1.0000} {0.5000} {0.3333} {0.2500} ]
[ {0.5000} {0.3333} {0.2500} {0.2000} ]
[ {0.3333} {0.2500} {0.2000} {0.1667} ]
[ {0.2500} {0.2000} {0.1667} {0.1429} ]
--------------------------------
G1:

[ {1.0000} {0.0000} {0.0000} {0.0000} ]

[ {0.0000} {1.0000} {0.0000} {0.0000} ]

[ {0.0000} {0.0000} {0.8000} {0.6000} ]

[ {0.0000} {0.0000} {-0.6000} {0.8000} ]

A1:

[ {1.0000} {0.5000} {0.3333} {0.2500} ]

[ {0.5000} {0.3333} {0.2500} {0.2000} ]

[ {0.4167} {0.3200} {0.2600} {0.2190} ]

[ {0.0000} {0.0100} {0.0133} {0.0143} ]

G2:

[ {1.0000} {0.0000} {0.0000} {0.0000} ]

[ {0.0000} {0.7682} {0.6402} {0.0000} ]

[ {0.0000} {-0.6402} {0.7682} {0.0000} ]

[ {0.0000} {0.0000} {0.0000} {1.0000} ]

A2:

[ {1.0000} {0.5000} {0.3333} {0.2500} ]

[ {0.6509} {0.4609} {0.3585} {0.2939} ]

[ {0.0000} {0.0324} {0.0397} {0.0402} ]

[ {0.0000} {0.0100} {0.0133} {0.0143} ]

G3:

[ {0.8381} {0.5455} {0.0000} {0.0000} ]

[ {-0.5455} {0.8381} {0.0000} {0.0000} ]

[ {0.0000} {0.0000} {1.0000} {0.0000} ]

[ {0.0000} {0.0000} {0.0000} {1.0000} ]

A3:

[ {1.1932} {0.6705} {0.4749} {0.3698} ]

[ {0.0000} {0.1136} {0.1186} {0.1099} ]

[ {0.0000} {0.0324} {0.0397} {0.0402} ]
```

```
[ {0.0000} {0.0100} {0.0133} {0.0143} ]

G4:

[ {1.0000} {0.0000} {0.0000} {0.0000} ]

[ {0.0000} {1.0000} {0.0000} {0.0000} ]

[ {0.0000} {0.0000} {0.9556} {0.2946} ]

[ {0.0000} {0.0000} {-0.2946} {0.9556} ]

A4:

[ {1.1932} {0.6705} {0.4749} {0.3698} ]

[ {0.0000} {0.1136} {0.1186} {0.1099} ]

[ {0.0000} {0.0339} {0.0419} {0.0427} ]

[ {0.0000} {0.0000} {0.0010} {0.0018} ]

G5:

[ {1.0000} {0.0000} {0.0000} {0.0000} ]

[ {0.0000} {0.9581} {0.2864} {0.0000} ]

[ {0.0000} {-0.2864} {0.9581} {0.0000} ]

[ {0.0000} {0.0000} {0.0000} {1.0000} ]

A5:

[ {1.1932} {0.6705} {0.4749} {0.3698} ]

[ {0.0000} {0.1185} {0.1257} {0.1175} ]

[ {0.0000} {0.0000} {0.0061} {0.0094} ]

[ {0.0000} {0.0000} {0.0010} {0.0018} ]

G6:

[ {1.0000} {0.0000} {0.0000} {0.0000} ]

[ {0.0000} {1.0000} {0.0000} {0.0000} ]

[ {0.0000} {0.0000} {0.9857} {0.1684} ]

[ {0.0000} {0.0000} {-0.1684} {0.9857} ]

A6:

[ {1.1932} {0.6705} {0.4749} {0.3698} ]

[ {0.0000} {0.1185} {0.1257} {0.1175} ]

[ {0.0000} {0.0000} {0.0062} {0.0096} ]

[ {0.0000} {0.0000} {0.0000} {0.0002} ]

Q:
```

```
[ {0.8381} {0.4191} {0.2794} {0.2095} ]

[ {-0.5226} {0.4417} {0.5288} {0.5021} ]

[ {0.1540} {-0.7278} {0.1395} {0.6536} ]

[ {-0.0263} {0.3157} {-0.7892} {0.5261} ]

R:

[ {1.1932} {0.6705} {0.4749} {0.3698} ]

[ {0.0000} {0.1185} {0.1257} {0.1175} ]

[ {0.0000} {0.0000} {0.0062} {0.0096} ]

[ {0.0000} {0.0000} {0.0000} {0.0002} ]

Did it work? Q x R:

[ {1.0000} {0.5000} {0.3333} {0.2500} ]

[ {0.5000} {0.3333} {0.2500} {0.2000} ]

[ {0.3333} {0.2500} {0.2000} {0.1667} ]

[ {0.2500} {0.2000} {0.1667} {0.1429} ]


The maximum norm of QR - H:
3.608224830031759E-16
```