Assignement 2, Numerical Linear Algebra

September 13, 2017

Ramona Bendias, Eda Dagasan

1 Task 1

In [33]:
```
import numpy as np
import scipy.linalg as la


class Orthogonalization():
    """
    A class of matrices with methods that orthogonalize the objects with
    different algorithms and that evaluate the result in different ways.
    """

    def __init__(self, givenmatrix):
        self.givenmatrix = givenmatrix

    def gramschmidt(self):
        """
        Gram-Schmidt orthogonalization of an object of the class.
        The method returns a matrice 'orthogonalmatrix' that is an
        orthogonal basis of range(A).
        """
        n = len(self.givenmatrix[0])
        m = len(self.givenmatrix)
        trianglematrix = np.zeros(shape=(n,n))
        orthogonalmatrix = np.array(np.zeros(shape=(self.givenmatrix.shape)))
        v = np.zeros(shape=(n,n))
        for j in range(n):
            v[j] = self.givenmatrix[j]
            for i in range(j-1):
                trianglematrix[i][j] = np.dot(orthogonalmatrix[i],self.givenmatrix[j])
                v[j] = v[j] - trianglematrix[i][j]*orthogonalmatrix[i]
            trianglematrix[j][j] = la.norm(v[j])
            orthogonalmatrix[j] = np.divide(v[j],trianglematrix[j][j])

        return orthogonalmatrix

    def norm(self, matrix):
        """
        Returns the 2-norm of a the input matrix A.
        """
        return np.linalg.norm(matrix, ord=2)
```

```python
    def qtq(self, matrix):
        """
        Returns the matrix product of the transpose of the input matrix A with
        itself.
        """
        return np.dot(matrix.transpose(),matrix)

    def deviation(self, matrix):
        """
        Returns the deviation of the output matrix of qtq from the identity
        matrix.
        """
        qtq = self.qtq(matrix)
        I = np.identity(len(qtq))
        return self.norm(I-qtq)

    def allclose(self, matrix):
        """
        Returns True/False if all the entries of QTQ are closer than a
        certain tolerance to the identity matrix.
        """
        qtq = self.qtq(matrix)
        I = np.identity(len(qtq))
        return np.allclose(qtq, I)

    def eigenvalues(self, matrix):
        """
        Returns an array with the eigenvalues of qtq of the input matrix A.
        """
        return la.eigvals(self.qtq(matrix))

    def determinant(self, matrix):
        """
        Returns the determinant of qtq of the input matrix A.
        """
        return la.det(self.qtq(matrix))
```

In [22]:
```python
A0 = np.random.rand(5,4)
print('Random matrix', A0)
A = Orthogonalization(A0)
V = A.gramschmidt()
print('Orthogonalized matrix', V)
```

```
Random matrix [[ 0.52745249  0.5659426   0.5077907   0.9493482 ]
 [ 0.8432061   0.8955658   0.7286134   0.20123158]
 [ 0.70375675  0.96840774  0.49015722  0.87452278]
 [ 0.025872    0.3669816   0.57318361  0.03942858]
 [ 0.41453031  0.45103638  0.57348573  0.4454796 ]]
Orthogonalized matrix [[ 0.39785244  0.42688516  0.38302173  0.71608439]
 [ 0.58403995  0.62030648  0.50466823  0.13938144]
 [ 0.25538695  0.79755601 -0.21402137 -0.50286854]
 [-0.7584177  -0.24019055  0.23949374 -0.55655533]
 [ 0.          0.          0.          0.        ]]
```

2 Task 2

In [17]:
```
def test(m,n, eigenvalues=None):
    '''
    @param m: integer which gives the number of rows
    @param n: integer which gives the number of columns
    Creates a random matrix with the shape mxn and returns after the gramschmidt factorisation
    values of Q and QTQ.
    '''
    matrix = np.random.rand(m,n)
    QR = Orthogonalization(matrix)
    norm = QR.norm(QR.gramschmidt())
    if eigenvalues:
        eigenvalues = QR.eigenvalues(QR.qtq(QR.gramschmidt()))
    else:
        eigenvalues = "not calculated  (To calculat them add e.g. a number to the input)"
    determinant = QR.determinant(QR.qtq(QR.gramschmidt()))
    deviation = QR.deviation(QR.qtq(QR.gramschmidt()))

    return "The two norm of the orthorganal matrix should be 1 and is: {}, \
           the deviation of the QTQ and I is: {}, the eigenvalues of the QTQ \
           are: {} and the determinat of the same matrix is: {}".format(norm, deviation, eigen
```

In [12]:
```
test(1000,1000)
```

Out[12]:
```
'The two norm of the orthorganal matrix should be 1 and is: 31.583760756540165, the deviation
```

In [13]:
```
test(5,5,1)
```

Out[13]:
```
'The two norm of the orthorganal matrix should be 1 and is: 1.7488222816236745, the deviation
```

The values of the test shouldn't be like they are. The norm of the Orthogonalmatrix should be 1 and is complete different. There is no rules e.g the bigger the matrix is the bigger the norm gets. The deviation of the QTQ from the identidymatrix is very high and if we use the numpy command allclose it is always FALSE, which means that the deviation is two big (measured by standard tolerance setting of numpy). Also the eigenvalues are not always 1 or -1, this is the same for the determinant.

3 Task 3

In [23]:
```
B,S = la.qr(A0)

print('Random matrix:\n', A0)
print('Triangular matrix:\n', S)
print('B@S \n',B@S)
print('Norm B:\n', A.norm(B))
print('Dev of BTB from I:\n', A.deviation(B))
print('BtB close to I?\n', A.allclose(B))
print('Eigenvalues of BTB:', A.eigenvalues(B))
print('Determinant of BTB:', A.determinant(B))
```

```
Random matrix:
 [[ 0.52745249  0.5659426   0.5077907   0.9493482 ]
 [ 0.8432061   0.8955658   0.7286134   0.20123158]
 [ 0.70375675  0.96840774  0.49015722  0.87452278]
 [ 0.025872    0.3669816   0.57318361  0.03942858]
```

```
 [ 0.41453031  0.45103638  0.57348573  0.4454796 ]]
Triangular matrix:
 [[-1.28723772 -1.50060911 -1.14952722 -1.14318587]
 [ 0.          0.38265425  0.41445617  0.16806105]
 [ 0.          0.          0.43956975 -0.11001063]
 [ 0.          0.          0.          0.74788512]
 [ 0.          0.          0.          0.        ]]
B@S
 [[ 0.52745249  0.5659426   0.5077907   0.9493482 ]
 [ 0.8432061   0.8955658   0.7286134   0.20123158]
 [ 0.70375675  0.96840774  0.49015722  0.87452278]
 [ 0.025872    0.3669816   0.57318361  0.03942858]
 [ 0.41453031  0.45103638  0.57348573  0.4454796 ]]
Norm B:
 1.0
Dev of BTB from I:
 4.59431270809e-16
BtB close to I?
 True
Eigenvalues of BTB: [ 1.+0.j  1.+0.j  1.+0.j  1.+0.j  1.+0.j]
Determinant of BTB: 1.0
```

When using pythons method for the factorization, QR gives the same A as the input, which indicates that our own methods contain some errors.

4 Task 4

```
In [42]: from numpy import *
         from scipy.linalg import *


         class Orthogonalization():
             """
             A class of matrices with methods that orthogonalize the objects with
             different algorithms and that evaluate the result in different ways.
             """

             def __init__(self, givenmatrix):
                 self.givenmatrix = givenmatrix

             def householder(self):
                 """
                 Will not work with a already triangular matrix (:
                 """
                 A = self.givenmatrix
                 m = shape(A)[0]
                 n = shape(A)[1]
                 Q = identity(m)
                 for i in range(n):
                     Q_i = identity(m)
                     x = A[i:m,i]
                     s = sign(x[0])
                     u = s*array([norm(x)]+(m-i-1)*[0.])
                     v_i = x + u
```

```
            v_i /= norm(v_i)
            Q_i_hat = eye(m-i) - 2*outer(v_i,v_i)
            Q_i[i:m,i:m] = Q_i_hat
            Q = Q_i@Q
            A = Q_i@A
        return Q, A

    def norm(self, matrix):
        """
        Returns the 2-norm of a the input matrix A.
        """
        return np.linalg.norm(matrix, ord=2)


    def qtq(self, matrix):
        """
        Returns the matrix product of the transpose of the input matrix A with
        itself.
        """
        return np.dot(matrix.transpose(),matrix)

    def deviation(self, matrix):
        """
        Returns the deviation of the output matrix of qtq from the identity
        matrix.
        """
        qtq = self.qtq(matrix)
        I = np.identity(len(qtq))
        return self.norm(I-qtq)

    def allclose(self, matrix):
        """
        Returns True/False if all the entries of QTQ are closer than a
        certain tolerance to the identity matrix.
        """
        qtq = self.qtq(matrix)
        I = np.identity(len(qtq))
        return np.allclose(qtq, I)

    def eigenvalues(self, matrix):
        """
        Returns an array with the eigenvalues of qtq of the input matrix A.
        """
        return la.eigvals(self.qtq(matrix))

    def determinant(self, matrix):
        """
        Returns the determinant of qtq of the input matrix A.
        """
        return la.det(self.qtq(matrix))


print('Random matrix\n', A0)
A = Orthogonalization(A0)
```

```
Q,R = A.householder()
print('Triangular matrix:\n', R)
print('Q@R:\n', Q@R)
print('Norm of Q:\n', A.norm(Q))
print('Deviation of QTQ from I:\n', A.deviation(Q))
print('QTQ allclose to I?\n', A.allclose(Q))
print('Eigenvalues of QTQ:\n', A.eigenvalues(Q))
print('Determinant of QTQ:\n', A.determinant(Q))
```

```
Random matrix
 [[ 0.52745249  0.5659426   0.5077907   0.9493482 ]
 [ 0.8432061   0.8955658   0.7286134   0.20123158]
 [ 0.70375675  0.96840774  0.49015722  0.87452278]
 [ 0.025872    0.3669816   0.57318361  0.03942858]
 [ 0.41453031  0.45103638  0.57348573  0.4454796 ]]
Triangular matrix:
 [[ -1.28723772e+00  -1.50060911e+00  -1.14952722e+00  -1.14318587e+00]
 [  4.46734965e-17   3.82654249e-01   4.14456171e-01   1.68061053e-01]
 [ -6.77454623e-17  -8.44330888e-19   4.39569746e-01  -1.10010627e-01]
 [ -1.25103166e-16   8.27949866e-17  -3.46478562e-18   7.47885120e-01]
 [  1.11695809e-16   8.49271310e-20  -1.79785409e-19   6.93889390e-18]]
Q@R:
 [[ 0.52745249  0.36422458 -0.0407859   0.40345117]
 [ 0.16463258  0.10451253  0.22235582  0.72357642]
 [-0.26289191 -0.24528104 -0.46710191  0.18334468]
 [-0.9034139  -1.30470581 -1.01468467 -1.01405479]
 [ 0.68293639  0.70160498  0.62252192  0.39786288]]
Norm of Q:
 1.0
Deviation of QTQ from I:
 8.90368157456e-16
QTQ allclose to I?
 True
Eigenvalues of QTQ:
 [ 1.+0.j  1.+0.j  1.+0.j  1.+0.j  1.+0.j]
Determinant of QTQ:
 0.9999999999999994
```

We have obtained an upper triangular matrix R and an orthogonal matrix Q (2-norm equal to 1 and QTQ has the expected properties). However QR does not give exactly A, which is why something is probably wrong in the code.

5 Task 5

When using the Householder transformation, an upper triangular matrix was constructed but iteratively picking a vector and turning all entries except one into zeros. Then it was achieved by reflecting the vector through a certain plane and onto the basis vector e_1. The same idea can be used, but instead of reflecting, the vector can be rotated onto e_1. For simplicity the algorithm rotates it into each of the hyperplanes until it is laying in the (e_1, e_2)-plane. It can then finally be rotated onto the first basis vector. (Theoretically only one rotation could take the vector onto the e_1-axis, but unless the vector is already laying in the (e_1, e_2)-plane, it is too difficult to decide the rotation matrix).

```
In [20]: from numpy import *
         from scipy.linalg import *
```

```python
class Orthogonalization():
    """
    A class of matrices with methods that orthogonalize the objects with
    different algorithms and that evaluate the result in different ways.
    """

    def __init__(self, givenmatrix):
        self.givenmatrix = givenmatrix

    def givens(self):
        """
        Takes a (mxn)-matrix with m≥n and returns its QR-factorization as the
        two matrices Q, A, by using Givens rotations.
        """
        A = self.givenmatrix
        m = shape(A)[0]
        n = shape(A)[1]
        Q = identity(m)
        for i in range(n): #counting through the columns of the matrix
            Q_i = identity(m)
            x = A[i:m,i]
            l = len(x)
            Q_i_hat = identity(l)
            for j in range(l-1):
                J_j = identity(l) #rotation matrix to be
                a = x[l-(j+2)]
                b = x[l-(j+1)]
                r = sqrt(a**2+b**2)
                c = a/r
                s = -b/r
                rotation = array([[c, -s], [s, c]])
                J_j[l-2-j:l-j, l-2-j:l-j] = rotation #rotation matrix in the
                                                      #(n-(i-1), n-i)-plane
                Q_i_hat = J_j@Q_i_hat #matrix for all the rotations of one vector
                x = dot(J_j,x)
            Q_i[i:m,i:m] = Q_i_hat
            Q = Q_i@Q
            A = Q_i@A
        return Q, A

    def norm(self, matrix):
        """
        Returns the 2-norm of a the input matrix A.
        """
        return np.linalg.norm(matrix, ord=2)


    def qtq(self, matrix):
        """
        Returns the matrix product of the transpose of the input matrix A with
        itself.
        """
```

```
                return np.dot(matrix.transpose(),matrix)

            def deviation(self, matrix):
                """
                Returns the deviation of the output matrix of qtq from the identity
                matrix.
                """
                qtq = self.qtq(matrix)
                I = np.identity(len(qtq))
                return self.norm(I-qtq)

            def allclose(self, matrix):
                """
                Returns True/False if all the entries of QTQ are closer than a
                certain tolerance to the identity matrix.
                """
                qtq = self.qtq(matrix)
                I = np.identity(len(qtq))
                return np.allclose(qtq, I)

            def eigenvalues(self, matrix):
                """
                Returns an array with the eigenvalues of qtq of the input matrix A.
                """
                return la.eigvals(self.qtq(matrix))

            def determinant(self, matrix):
                """
                Returns the determinant of qtq of the input matrix A.
                """
                return la.det(self.qtq(matrix))

        A0 = np.random.rand(3,3)
        print('Random matrix\n', A0)
        A = Orthogonalization(A0)

        q,r = A.givens()
        print('Triangular matrix:\n', r)
        print('q@r:\n',q@r)
        print('Norm:', A.norm(q))
        print('Deviation from I of qtq:\n', A.deviation(q))
        print('Qtq allclose to I?\n', A.allclose(q))
        print('Eigenvalues of qtq:\n', A.eigenvalues(q))
        print('Determinant of qtq:\n', A.determinant(q))
```

```
Random matrix
 [[ 0.56805938  0.11893226  0.16066496]
 [ 0.07767156  0.02337333  0.01248294]
 [ 0.55440036  0.57431109  0.17971081]]
Triangular matrix:
 [[  7.97548800e-01   4.86207618e-01   2.40572793e-01]
 [ -1.09369058e-16   3.28826048e-01   1.71567868e-02]
 [  1.90882390e-17   0.00000000e+00   9.76743995e-03]]
q@r:
```

```
 [[ 0.56805938  0.37832823  0.17981006]
 [-0.55147856 -0.36017384 -0.16057911]
 [ 0.09624984 -0.26770688  0.01215394]]
Norm: 1.0
Deviation from I of qtq:
 2.61509564004e-16
Qtq allclose to I?
 True
Eigenvalues of qtq:
 [ 1.+0.j  1.+0.j  1.+0.j]
Determinant of qtq:
 1.0000000000000004
```

Clearily, the same problem as for the Householder reflections occur, that QR doesn't give A.

We got some help to fill in our gaps using the following references: http://statweb.stanford.edu/~susan/courses/b494/index/node30.html https://en.wikipedia.org/wiki/Givens_rotation

This method for QR-factorization must have a higher computational cost, since it realises several rotations (i.e. matrix multiplications) in order to move the vector onto the e_1-axis, where Householder only used one single reflexion.