Assignment 2

Vilhelm Roxling and Axel Goteman

September 17, 2014

Task 1

Below is the code for our *Orthogonalization* class written in python.

```
import numpy as np

class Orthogonalization:

    """
    It takes as an input the m x n array A (transposed for coding c
    onvenience) and assume_li which is a boolean(False by default) t
    elling us if the columns of A can be assumed to be linearly iden
    pendent or not.
    """

    def __init__(self, A, assume_li = False):
        self.assume_li = assume_li
        self.A = A
        [self.n, self.m] = A.shape

    def gramschmidt(self):
        tol = 1e-12

        Q = np.zeros((self.n, self.m))
        Q[0] = self.A[0]/np.linalg.norm(self.A[0])
        lin_dep_inds = []

        for i in range(1,self.n):
            f = self.A[i]
            projvec = zeros(self.m)

            for k in range(i):
                projvec += np.dot(Q[k],f)*Q[k]

            v = f - projvec
            vn = np.linalg.norm(v)

            if self.assume_li:
                Q[i] = v/vn
            else:
                if vn > tol:
                    Q[i] = v/vn
                else:
                    lin_dep_inds.append(i)

        return delete(Q, lin_dep_inds, axis=0)
```

Task 2

When testing our Gram-Schmidt method we used a matrix with columns that had very similar directions, but that where theoretically linearly independent. Below is the matrix we used:

$$A = \begin{pmatrix} ones(n) \\ \epsilon I \end{pmatrix} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ \epsilon & 0 & \dots & 0 \\ 0 & \epsilon & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \epsilon \end{pmatrix} \tag{1}$$

with a small ϵ (=10^{-7} in our example) for varying n. Thus all columns are allmost parallell to the vector $(1, 0, \dots, 0)$.

Below is the code for the tests:

```
from orthogonalization import *

nlist = [1, 10, 100, 1000]
norms = []
i_divs = []
eigs = []
dets = []
tol = 1e-6
eps = 1e-7

for n in nlist:
    A = np.vstack((np.ones(n), eps*np.eye(n))).T
    O = Orthogonalization(A, assume_li = True)
    Q = O.gramschmidt()
    QTQ = np.dot(Q,Q.T)
    norms.append(np.linalg.norm(Q, ord=2))
    i_divs.append(np.linalg.norm(QTQ-eye(n), ord=2))
    dev = np.abs(np.linalg.eig(QTQ)[0]-ones(n))
    eigs.append(dev[dev>tol])
    dets.append(np.linalg.det(QTQ))

print norms, "\n\n", i_divs, "\n\n", eigs, "\n\n", dets
```

If the method was perfect we would expect the 2-norm of Q to always be equal to one. The matrix Q^TQ would be equal to the nxn identity matrix, which would imply that all the eigenvalues should equal to one, and the determinant should also be one.

With our test matrix we got the following results:

- The norms stayed fairily close to one
- The norm of the deviation from the identity matrix got close to one for large n.
- The eigenvalues drifted from one to zero for large n.
- The determinant went from one to zero with increasing n.

Task 3

The results with scipy.linalg's method *qr* were much closer to the theoretical results, with only minor deviations appearing with large n.

The reason the Gram-Schmidt algorithm didn't produce the correct results is that it is numerically unstable. The reason for this is that the algorithm in each step relies on the previous steps being correct, making it sensitive to round-off errors. Thus a small error in the early steps will be magnified in every step. Since the orthogonal projection between two very similar vectors will produce a very small vector, any directional errors produced in the calculations will be magnified when normalizing the vector.

Task 4

For solving this problem we begin with recalling the method to obtain the intersection of two planes in $\mathbf{R^3}$. We find the normal vectors to the planes, and do the cross product with them. The resulting vector is in the direction of the intersection.

First consider the $3x2$ matrices

$$A^{(1)} = \begin{pmatrix} x^{(1)} & y^{(1)} \end{pmatrix}, \quad A^{(2)} = \begin{pmatrix} x^{(2)} & y^{(2)} \end{pmatrix} \tag{2}$$

If we now do the full QR factorization of those matrices creating $Q^{(i)} = (q_1^{(i)} q_2^{(i)} q_3^{(i)})$. The first two columns of $Q^{(i)}$ will span the same subspace(plane) as the columns of $A^{(i)}$. The third columns will be orthogonal to the first two, thus the normals to the planes. Then do the QR factorization of the matrix

$$A^{(3)} = \begin{pmatrix} q_3^{(1)} & q_3^{(2)} \end{pmatrix} \tag{3}$$

Then the third column of $Q^{(3)}$ must be orthogonal to the normals of the planes.

Task 5

We did the QR factorization using scipy.linalg's *qr* and then used the backward substitution method to solve $RX = Q^T b$. This took about seven times as long as using scipy.linalg's *solve* method.

```
n = 500
A = np.random.rand(n, n)
b = np.random.rand(n)
t0 = time.time()
x = la.solve(A, b)
print time.time()-t0
print np.linalg.norm(dot(A, x)-b)

t0 = time.time()
Q, r = la.qr(A)
QTb = np.dot(Q.T, b)

x = zeros(n)

for i in range(n):
    sum = 0

    for k in range(n-i, n):
        sum += r[n-i-1][k]*x[k]

    x[n-i-1] = (QTb[n-i-1]-sum)/r[n-i-1][n-i-1]

print "\n", time.time()-t0
print np.linalg.norm(dot(A,x)-b)
```