Lecture 10 HW

Mike Snyder

10.2 (a) see attached code.
(b) see attached code

10.4 Consider the 2×2 orthogonal matrices

$$F = \begin{bmatrix} -c & s \\ s & c \end{bmatrix}, \quad J = \begin{bmatrix} c & s \\ -s & c \end{bmatrix},$$

where $s = \sin\theta$ and $c = \cos\theta$ for some θ.

(a) Describe exactly what geometric effects left-multiplications by F and J have on the plane $\mathbb{R}^2$.

Solution: The matrix F takes $\binom{1}{0}$ accross the line $y=x$, by θ, the y-axis and $\binom{0}{1}$ across the line
The matrix J is a counter-clockwise rotation of $\binom{1}{0}$ and $\binom{0}{1}$ by angle θ.

(b) Describe an algorithm for QR factorization that is analogous to Algorithm 10.1, but based on Givens rotations instead of Householder reflections.

Solution: We use a Givens rotation of angle θ, which we denote $J(\theta)$, to obtain the vector equivalent to $\|x\|_2 e_1$ in Algorithm 10.1. From Part (b), we know that J is a counterclockwise rotation, by angle θ. Thus, given

$$x = A_{k:m,k}$$

with angle

$$\theta = \cos^{-1}\left(\frac{x^* e_1}{\|x\|_2}\right),$$

from the x-axis,

we may rotate x by $-J(\theta)x$ to obtain $\|x\|_2 e_1$. Then we may form

$$v_k = \|x\|_2 e_1 - x$$
$$= -J(\theta)x - x.$$

The remainder of the algorithm to compute Q & R follow from the text, and we give them here.

$$\text{for } k = 1 \text{ to } n$$
$$x = A_{k:m,k}$$
$$\theta = \cos^{-1}\left(\frac{x^* e_1}{\|x\|_2}\right)$$
$$v_k = -J(\theta)x - x$$
$$v_k = v_k / \|v_k\|_2$$
$$A_{k:m,k:n} = A_{k:m,k:n} - 2v_k(v_k^* A_{k:m,k:n})$$

and using e_k for $k = n$ downto 1, we compute Q using

$$\text{for } k = n \text{ downto } 1$$
$$q_{k:m} = q_{k:m} - 2v_k(v_k^* q_{k:m}).$$

(c) Show that your algorithm involves six flops per entry operated on rather that 4, so that the asymptotic operation count is 50% greater than (10.9).

<u>solution</u>: I am not sure where the explicit numbers 6 and 4 are coming from, but from my analysis,

$$v_k = -J(\theta)x - x$$

has n^2 multiplications and $n(n-1)$ additions for $-J(\theta)x$ and another n additions for $-J(\theta)x - x$. Thus, this computation has

$$n^2 + n(n-1) + n = 2n^2$$

flops.

Then

$$v_k = \operatorname{sign}(x_1)\|x\|_2 e_1 + x$$

has $n+1$ multiplications for $\operatorname{sign}(x)\|x\| e_1$, $n-1$ additions for n multiplications and $n-1$ additions for $\|x\|_2$ and another n additions for $\operatorname{sign}(x_1)\|x\|_2 e_1 + x$. Thus, we have

$$n+1 + n + n-1 + n = 4n$$

flops.

If you include the additional flops for computing θ, the Givens rotation method becomes even worse.

householder

September 21, 2022

[ ]:
```
import numpy as np
```

[ ]:
```
def e(i, length):
    # Given a position and a length, return a vector with a 1.0 in the ith␣
 ↪position and 0.0 else.
    e_i = np.zeros(length, dtype=np.float64)
    e_i[i] = 1.0
    return e_i
```

[ ]:
```
def reflector(v):
    """Given a unit vector v, return the householder reflector I - 2 * np.
 ↪outer(v, v)

    Args:
        v (arr): Vector
    Returns:
        reflector (arr): A len(v) x len(v) array representing a householder␣
 ↪reflector
    """
    I = np.eye(len(v))
    outer = np.outer(v, v)


    return (I - 2 * outer)
```

[ ]:
```
def formQ(W):
        """Form Q from the lower diagonal vk's formed in the householder␣
 ↪algorithm.

        Args:
            W (arr): An m x n matrix containing the vk in the kth column

        Returns:
            arr: Returns an m x m orthonormal matrix Q such that QR = A
        """
        m = W.shape[0]
        n = W.shape[1]
```

```
        Q = np.eye(m, m)
        for k in range(n):
                Qk = np.eye(m, m)
                Qk[k:m, k:m] = reflector(W[k:m, k])
                Q = Q @ Qk

        return np.around(Q, decimals=6)
```

```
[ ]: def house(A):
         """Compute the factor R of a QR factorization of an m x n matrix A with m␣
      ↪>= .

         Args:
             A (arr): A numpy array of shape m x n
         Returns:
             R (arr): The upper diagonal matrix R in a QR factorization
         """
         m = A.shape[0] # Get row-dim of A
         n = A.shape[1] # Get col-dim of A
         W = np.zeros((m, n))
         #Q = np.eye(m, m)
         # V = np.zeros((m, n))

         # Cast A as type np.float64
         A = A.astype(np.float64)

         for k in range(n):
             # Get the first column of the (m-k+1, n-k+1) submatrix of A
             x = A[k:m, k]

             # Compute e_1, sign function, and the norm of x
             e_1 = e(0, len(x))
             sign = np.sign(x[0])
             norm_x = np.linalg.norm(x)

             # Compute vk, the vector reflected across the Householder hyperplane
             vk = (sign * norm_x * e_1) + x
             vk = vk / np.linalg.norm(vk)

             # Store vk in W
             W[k:m, k] = vk

             # Apply the reflector to the k:m, k:n submatrix of A to put
             # zeros below the diagonal of the kth column of A
             A[k:m, k:n] = reflector(vk) @ A[k:m, k:n]
```

```
    return np.around(A, decimals=6), W
```

[ ]:
```
A = np.array([[1, 2, 1], [4, 5, 6], [6, 9, 8]])
```

[ ]:
```
R, W = house(A)
print("R: \n" + str(R))
```

```
R:
[[ -7.28011  -10.439403 -10.027321]
 [ -0.          1.00939    -0.672927]
 [ -0.          0.          0.      ]]
```

[ ]:
```
Q = formQ(W)
print("Q: \n" + str(Q))
```

```
Q:
[[-0.137361  0.560772  0.816497]
 [-0.549442 -0.729004  0.408248]
 [-0.824163  0.392541 -0.408248]]
```

[ ]:
```
np.linalg.qr(A)
```

[ ]:
```
(array([[-0.13736056,  0.56077215, -0.81649658],
        [-0.54944226, -0.7290038 , -0.40824829],
        [-0.82416338,  0.39254051,  0.40824829]]),
 array([[-7.28010989e+00, -1.04394029e+01, -1.00273212e+01],
        [ 0.00000000e+00,  1.00938988e+00, -6.72926585e-01],
        [ 0.00000000e+00,  0.00000000e+00, -6.16000306e-15]]))
```

[ ]:
```
B = np.array([[2, 3, 4], [5, 6, 7], [8, 9, 10]])
print(B)
```

```
[[ 2  3  4]
 [ 5  6  7]
 [ 8  9 10]]
```

[ ]:
```
R, W = house(B)
print("R: \n" + str(R))
```

```
R:
[[ -9.643651 -11.199078 -12.754506]
 [  0.          0.762001    1.524002]
 [  0.         -0.         -0.      ]]
```

[ ]:
```
Q = formQ(W)
print("Q: \n" + str(Q))
```

```
Q:
[[-0.20739   0.889001 -0.408248]
 [-0.518476  0.254     0.816497]
 [-0.829561 -0.381    -0.408248]]
```

[ ]:
```python
np.linalg.qr(B)
```

[ ]:
```
(array([[-0.20739034,  0.88900089,  0.40824829],
        [-0.51847585,  0.25400025, -0.81649658],
        [-0.82956136, -0.38100038,  0.40824829]]),
 array([[-9.64365076e+00, -1.11990783e+01, -1.27545058e+01],
        [ 0.00000000e+00,  7.62000762e-01,  1.52400152e+00],
        [ 0.00000000e+00,  0.00000000e+00,  1.11022302e-15]]))
```

[ ]:
```python
C = np.array([[1, 2], [3, 4], [5, 6]])
R, W = house(C)
print("R: \n" + str(R))
```

```
R:
[[-5.91608  -7.437357]
 [-0.        0.828079]
 [ 0.       -0.      ]]
```

[ ]:
```python
Q = formQ(W)
print("Q: \n" + str(Q))
```

```
Q:
[[-0.169031  0.897085  0.408248]
 [-0.507093  0.276026 -0.816497]
 [-0.845154 -0.345033  0.408248]]
```

[ ]:
```python
np.linalg.qr(C)
```

[ ]:
```
(array([[-0.16903085,  0.89708523],
        [-0.50709255,  0.27602622],
        [-0.84515425, -0.34503278]]),
 array([[-5.91607978, -7.43735744],
        [ 0.        ,  0.82807867]]))
```

[ ]:
```python

```