Numerical Linear Algebra: Homework #8

Due on September 19, 2022 at 10:00PM

Instructor: Professor Blake Barker
Section 1

Michael Snyder

Problem 8.1

Let A be an $m \times n$ matrix. Determine the exact number of floating point operations (i.e., additions, subtractions, multiplications, and divisions) involved in computing the factorization $A = \hat{Q}\hat{R}$ by Algorithm 8.1.

Solution: First we note that because we are considering the reduced QR factorization, Q is $m \times n$ and R is $n \times n$. With this in mind, and beginning from the inner loop, we find that

Inner Loop

$r_{ij}q_i$ results m multiplications and $v_j - r_{ij}q_i$ results in m subtractions. Thus, $v_j = v_j - r_{ij}q_i$ results in $2m$ floating point operations.

$q_i^* v_j$ results in m multiplications and $m-1$ additions. Thus, $r_{ij} = q_i^* v_j$ results in $m + m - 1 = 2m - 1$ floating point operations.

Adding these two assignments within the inner loop together, we have $4m - 1$ floating point operations.

Since these assignments happen for $i = 1$ to n and for $j = i+1$ to n, we are summing up the $4m-1$ floating point operations n times starting at $j = 1+1 = 2$ to n which means we have $\frac{n}{2}(n - (i+1) + 1) = \frac{n}{2}(n - 2 + 1) = \frac{n(n-1)}{2}$ terms. Thus we have

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} (4m-1) = \frac{1}{2} \cdot n(n-1) \cdot (4m-1) = \frac{n(n-1)}{2}(4m-1)$$

floating point operations.

Outer Loop

The outer loop consists of m divisions in v_i / r_{ii} and m multiplications and $m-1$ additions in $\left\| v_i \right\|$.
Then summing those up for $i = 1$ to n we have $n(3m-1)$ floating point operations.

We note here prior to summing all the floating point operations that the top loop of Algorithm 8.1 (i.e., with $v_i = a_i$) does not result in any floating point operations since it is only reassignment.

Now, adding up the inner and outer loops, we have

$$\text{inner loop } + \text{ outer loop } = \frac{n(n-1)}{2}(4m-1) + n(3m-1)$$

floating point operations.

modified_gram_schmidt

September 18, 2022

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

[ ]:
```
# The modified Gram-Schmidt Algorithm as presented in Algorithm 8.1
# in the book 'Numerical Linear Algebra' by Trefethen and Bau
def mgs(A):
    """An implementation of modified Gram-Schmidt for QR factorization. The␣
↪implementation uses orthogonal projections
    to compute the orthonormal vectors q that form the columns of the matrix Q.

    Args:
        A (arr): An m x n matrix A
    Output:
        Q (arr): an m x n matrix with orthonormal columns
        R (arr): an n x n upper-diagonal matrix
    """
    m = A.shape[0] # Get row-dim of A
    n = A.shape[1] # Get col-dim of A
    Q = np.zeros((m, n)) # Initialize matrix Q
    R = np.zeros((n, n)) # Initialize matrix R

    # Copy the matrix A into V. This is a loop in Algorithm 8.1,
    # but the loop is unecessary
    V = A.copy().astype(np.float64)

    for i in range(n):
        # Raise error if provided matrix is singular
        if np.linalg.norm(V[:, i]) == 0:
                raise ValueError("The provided matrix is singular. Modified␣
↪Gram-Schmidt only works for non-singular matrices")

        R[i, i] = np.linalg.norm(V[:, i]) # Compute each r_ii
        Q[:, i] = V[:, i] / R[i, i] # normalize each orthogonal v

        for j in range(i+1, n):
            R[i, j] = Q[:, i]@V[:, j] # Compute r_ij
            V[:, j] = V[:, j] - R[i, j]*Q[:, i] # Get the orthogonal projection␣
↪as soon as the latest
```

```
                                                # q_i is known



    return Q, R
```

[ ]:
```
A = np.array([[1, 2], [5, 6], [8, 9]])
print(A)
print("My version of Algorithm 8.1 \n" + str(mgs(A)))
print("Numpy's version of reduced QR factorization \n" +  str(np.linalg.qr(A)))
```

```
[[1 2]
 [5 6]
 [8 9]]
My version of Algorithm 8.1
(array([[ 0.10540926,  0.93127185],
       [ 0.52704628,  0.24507154],
       [ 0.84327404, -0.26957869]]), array([[ 9.48683298, 10.96256256],
       [ 0.        ,  0.9067647 ]]))
Numpy's version of reduced QR factorization
(array([[-0.10540926,  0.93127185],
       [-0.52704628,  0.24507154],
       [-0.84327404, -0.26957869]]), array([[ -9.48683298, -10.96256256],
       [  0.        ,   0.9067647 ]]))
```

[ ]:
```
A = np.random.rand(5, 5)
print(A)
print("My version of Algorithm 8.1 \n" + str(mgs(A)))
print("Numpy's version of reduced QR factorization \n" +  str(np.linalg.qr(A)))
```

```
[[0.70863954 0.94103797 0.30406181 0.18355483 0.32602295]
 [0.39927149 0.81401655 0.20308521 0.9074034  0.9249954 ]
 [0.54039882 0.04462974 0.49830826 0.72436463 0.8351811 ]
 [0.51866884 0.23652526 0.90420323 0.19028472 0.52702052]
 [0.46583142 0.24385973 0.17683717 0.23034308 0.28039752]]
My version of Algorithm 8.1
(array([[ 0.59060783,  0.42762526, -0.21457682, -0.50902989,  0.40393611],
       [ 0.33276843,  0.61478594,  0.17257791,  0.66015862, -0.21380102],
       [ 0.45038945, -0.56041461, -0.19314912,  0.49165661,  0.45172116],
       [ 0.43227884, -0.2827369 ,  0.80317581, -0.22663736, -0.19167413],
       [ 0.38824208, -0.2125328 , -0.49170691, -0.1094918 , -0.74184484]]),
array([[ 1.19984786,  1.04368564,  0.93111797,  0.90829524,  1.21319966],
       [ 0.        ,  0.75914377, -0.3176161 ,  0.12765105,  0.03144031],
       [ 0.        ,  0.        ,  0.51283771,  0.01687159,  0.21377905],
       [ 0.        ,  0.        ,  0.        ,  0.79338764,  0.7051668 ],
       [ 0.        ,  0.        ,  0.        ,  0.        ,  0.00216881]]))
Numpy's version of reduced QR factorization
```

```
(array([[-0.59060783, -0.42762526, -0.21457682,  0.50902989, -0.40393611],
       [-0.33276843, -0.61478594,  0.17257791, -0.66015862,  0.21380102],
       [-0.45038945,  0.56041461, -0.19314912, -0.49165661, -0.45172116],
       [-0.43227884,  0.2827369 ,  0.80317581,  0.22663736,  0.19167413],
       [-0.38824208,  0.2125328 , -0.49170691,  0.1094918 ,  0.74184484]]),
array([[-1.19984786, -1.04368564, -0.93111797, -0.90829524, -1.21319966],
       [ 0.        , -0.75914377,  0.3176161 , -0.12765105, -0.03144031],
       [ 0.        ,  0.        ,  0.51283771,  0.01687159,  0.21377905],
       [ 0.        ,  0.        ,  0.        , -0.79338764, -0.7051668 ],
       [ 0.        ,  0.        ,  0.        ,  0.        , -0.00216881]]))
```

[ ]:
```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
mgs(A)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/home/masvgp/dev/byu_num_lin_alg_2022/gram_schmidt/modified_gram_schmidt.ipynb␣
↪Cell 5 in <cell line: 2>()
      <a href='vscode-notebook-cell://wsl%2Bubuntu/home/masvgp/dev/
↪byu_num_lin_alg_2022/gram_schmidt/modified_gram_schmidt.
↪ipynb#X10sdnNjb2RlLXJlbW90ZQ%3D%3D?line=0'>1</a> A = np.array([[1, 2, 3], [4,␣
↪5, 6], [7, 8, 9]])
----> <a href='vscode-notebook-cell://wsl%2Bubuntu/home/masvgp/dev/
↪byu_num_lin_alg_2022/gram_schmidt/modified_gram_schmidt.
↪ipynb#X10sdnNjb2RlLXJlbW90ZQ%3D%3D?line=1'>2</a> mgs(A)

/home/masvgp/dev/byu_num_lin_alg_2022/gram_schmidt/modified_gram_schmidt.ipynb␣
↪Cell 5 in mgs(A)
     <a href='vscode-notebook-cell://wsl%2Bubuntu/home/masvgp/dev/
↪byu_num_lin_alg_2022/gram_schmidt/modified_gram_schmidt.
↪ipynb#X10sdnNjb2RlLXJlbW90ZQ%3D%3D?line=21'>22</a> for i in range(n):
     <a href='vscode-notebook-cell://wsl%2Bubuntu/home/masvgp/dev/
↪byu_num_lin_alg_2022/gram_schmidt/modified_gram_schmidt.
↪ipynb#X10sdnNjb2RlLXJlbW90ZQ%3D%3D?line=22'>23</a>     # Raise error if␣
↪provided matrix is singular
     <a href='vscode-notebook-cell://wsl%2Bubuntu/home/masvgp/dev/
↪byu_num_lin_alg_2022/gram_schmidt/modified_gram_schmidt.
↪ipynb#X10sdnNjb2RlLXJlbW90ZQ%3D%3D?line=23'>24</a>     if np.linalg.norm(V[:,␣
↪i]) == 0:
---> <a href='vscode-notebook-cell://wsl%2Bubuntu/home/masvgp/dev/
↪byu_num_lin_alg_2022/gram_schmidt/modified_gram_schmidt.
↪ipynb#X10sdnNjb2RlLXJlbW90ZQ%3D%3D?line=24'>25</a>             raise␣
↪ValueError("The provided matrix is singular. Modified Gram-Schmidt only works␣
↪for non-singular matrices")
     <a href='vscode-notebook-cell://wsl%2Bubuntu/home/masvgp/dev/
↪byu_num_lin_alg_2022/gram_schmidt/modified_gram_schmidt.
↪ipynb#X10sdnNjb2RlLXJlbW90ZQ%3D%3D?line=26'>27</a>     R[i, i] = np.linalg.
↪norm(V[:, i]) # Compute each r_ii
```

```
    <a href='vscode-notebook-cell://wsl%2Bubuntu/home/masvgp/dev/
↪byu_num_lin_alg_2022/gram_schmidt/modified_gram_schmidt.
↪ipynb#X10sdnNjb2RlLXJlbW90ZQ%3D%3D?line=27'>28</a>     Q[:, i] = V[:, i] /␣
↪R[i, i] # normalize each orthogonal v

ValueError: The provided matrix is singular. Modified Gram-Schmidt only works␣
↪for non-singular matrices
```