Math 6108 Homework 6

Jacob Hauck

October 7, 2024

Question 1.

Question 2.

Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathbb{R}^n$ be orthonormal. Let $A \in \mathbb{R}^n$. If $A\mathbf{x}_1, A\mathbf{x}_2, \ldots, A\mathbf{x}_n$ are also orthonormal, then A is orthogonal.

Proof. Let $X = \begin{bmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n\end{bmatrix}$, and let $B = \begin{bmatrix}A\mathbf{x}_1 & A\mathbf{x}_2 & \cdots & A\mathbf{x}_n\end{bmatrix}$. Then $B = AX$. Since the columns of B and X are orthonormal, they are both orthogonal matrices. Therefore,

$$A = BX^T.$$

Since $(BX^T)^T(BX^T) = XB^TBX^T = I$ and $(BX^T)(BX^T)^T = BX^TXB^T = I$ by the orthogonality of B and X, it follows that A is invertible with $A^{-1} = (BX^T)^T = A^T$. This implies that A is orthogonal. $\square$

Question 3.

The algorithm for the Gram-Schmidt process is described in Algorithm 1. An implementation in Python is provided in Listing 1. We note that this implementation detects linear dependence of the columns of A as a part of the Gram-Schmidt process by checking if the produced orthogonal vectors are zero (well, almost zero, to account for numerical rounding error). This is possible because the columns of A are linearly dependent if and only the Gram-Schmidt process produces a zero vector at some point. This is easy to prove.

For $j < i$, each $\mathbf{b}_j$ is a linear combination of the first j columns of A. We can prove this by induction. For the base case, $\mathbf{b}_1 = \|\mathbf{a}_1\|^{-1}\mathbf{a}_1$. For some $1 \le k < i-1$, suppose for induction that $\mathbf{b}_j = \sum_{m=1}^{j} c_{jm}\mathbf{a}_m$ for $1 \le j \le k$ and some constants c_{jm}. Then

$$\mathbf{b}_{k+1} = \mathbf{a}_{k+1} - \sum_{p=1}^{k}\langle \mathbf{b}_p, \mathbf{a}_{k+1}\rangle \sum_{m=1}^{p} c_{pm}\mathbf{a}_m$$

which completes the proof by induction.

Suppose that $\mathbf{b}_i = \mathbf{0}$. Then

$$\mathbf{0} = \mathbf{b}_i = \mathbf{a}_i - \sum_{p=1}^{i-1}\langle \mathbf{b}_p, \mathbf{a}_i\rangle \sum_{m=1}^{p} c_{pm}\mathbf{a}_m.$$

The coefficient of $\mathbf{a}_i$ is non-zero, so a non-trivial linear combination of the columns of A is $\mathbf{0}$, meaning that the columns of A are linearly dependent.

Conversely, if the columns of A are linearly dependent, then there exists $c_1, \ldots, c_m$ not all equal to zero such that

$$\sum_{i=1}^{m} c_i\mathbf{a}_i = \mathbf{0}.$$

Let k be the largest integer such that $c_k \neq 0$. Then

$$\mathbf{0} = \sum_{i=1}^{k} c_i \mathbf{a}_i = c_k \mathbf{b}_k + \sum_{p=1}^{k-1} \langle \mathbf{b}_p, \mathbf{a}_k \rangle \mathbf{b}_p + \sum_{i=1}^{k-1} \left(c_i \mathbf{b}_i + c_i \sum_{p=1}^{i-1} \langle \mathbf{b}_p, \mathbf{a}_i \rangle \mathbf{b}_p \right).$$

The coefficient of $\mathbf{b}_k$ is nonzero, so it follows that the columns of B are linearly dependent. Since the columns of B are also orthogonal because of the Gram-Schmidt process, one of them must be zero.

The command `python -m gs` can be used to run the tests, which verify that the function works across a range of input types that cover every code path. The output from running these tests is given in Listing 2.

Algorithm 1: Gram-Schmidt Orthogonalization

Input: Matrix $A \in \mathbb{R}^{n \times m}$ with linearly independent columns $\mathbf{a}_1, \ldots, \mathbf{a}_m \in \mathbb{R}^n$

Output: Matrix $B \in \mathbb{R}^{n \times m}$, whose columns $\mathbf{b}_1, \ldots, \mathbf{b}_m \in \mathbb{R}^n$ are the orthogonal vectors obtained by applying the Gram-Schmidt process to the columns of A

$c \leftarrow 1$;
repeat
  $\mathbf{b}_c \leftarrow \mathbf{a}_c - \sum_{p=1}^{c-1} \langle \mathbf{b}_p, \mathbf{a}_c \rangle \mathbf{b}_p$; `// Sum is 0 by convention if` $c = 1$
  $\mathbf{b}_c \leftarrow \frac{\mathbf{b}_c}{\|\mathbf{b}_c\|}$
until $c = n$;

Listing 1: Python implementation of the Gram-Schmidt process

```python
import numpy as np


class LinearDependenceError(BaseException):
    """Exception class that is raised to indicate linearly dependent input vectors
    """
    pass


def gram_schmidt(a, eps_d=1e-10):
    """
    Perform the Gram-Schmidt orthogonalization process on the columns of
    a matrix, returning orthogonalized vectors as the columns of a new matrix.

    :param a: n x m matrix with linearly independent columns. Raises
        SingularMatrixError if columns of a are linearly dependent or almost
        linearly dependent (see eps_d).
    :param eps_d: Tolerance for approximate linear independence (minimum norm of
        the computed orthogonal columns). Default = 10^{-10}
    :return: n x m matrix b whose columns are orthogonal (orthonormal, if
        normalize == True) vectors obtained by performing Gram-Schmidt
        orthogonalization on the columns of a.
    """

    # ==== Input Validation ====

    # Ensure input has the correct data type
```

```
    a = np.array(a, dtype=float)
    assert len(a.shape) == 2

    # Early check for linear dependence
    if a.shape[1] > a.shape[0]:
        raise LinearDependenceError('Matrix has linearly dependent columns '
                                    '(more columns than rows)')

    # ==== Run Gram-Schmidt process ====

    # Initialization

    # The first step for each column is copying the corresponding column from a,
    # so we initialize the output equal to a. Since we already copied the input
    # with np.array(), we can use that memory for our output matrix
    b = a

    # Normalize the first column of b
    norm = np.linalg.norm(b[:, 0])

    # Check for approximate linear dependence before possible divide-by-zero
    if norm < eps_d:
        raise LinearDependenceError('Matrix has linearly dependent or almost '
                                    'linearly dependent columns because first '
                                    'column is almost 0')
    b[:, 0] /= norm

    # Iteration
    for col in range(1, a.shape[1]):
        # Recall that b[:, col] == a[:, col] because of initialization

        # Subtract out previous orthonormal columns
        b[:, col] -= b[:, :col] @ (b[:, :col].T @ b[:, col])

        # Normalize new column
        norm = np.linalg.norm(b[:, col])

        # Check for approximate linear dependence before possible divide-by-zero
        if norm < eps_d:
            raise LinearDependenceError('Aborting orthogonalization; matrix has '
                                        'linearly dependent or almost linearly '
                                        'dependent columns')
        b[:, col] /= norm

    # Return orthogonal columns
    return b


# Test example
if __name__ == '__main__':
    # Set RNG seed for reproducible results
```

```python
    np.random.seed(2024)

    print('Test 1: random square matrix')
    a = np.random.random((5, 5))
    print('Input matrix')
    print(a)
    print()
    print('Orthonormalized matrix')
    b = gram_schmidt(a)
    print(b)
    print()
    print('Implementation worked?', np.allclose(b.T @ b, np.eye(5)))
    print()

    print('Test 2: random non-square matrix')
    a = np.random.random((5, 3))
    print('Input matrix')
    print(a)
    print()
    print('Orthonormalized matrix')
    b = gram_schmidt(a)
    print(b)
    print()
    print('Implementation worked?', np.allclose(b.T @ b, np.eye(3)))
    print()

    print('Test 3: random matrix with too many columns')
    a = np.random.random((3, 5))
    print('Input matrix')
    print(a)
    print()
    try:
        gram_schmidt(a)  # should raise an error
    except LinearDependenceError as e:
        print(e)
    print()

    print('Test 4: matrix with first column 0')
    a = np.array([
        [0, 1, 2],
        [0, 3, 4],
        [0, 5, 6]
    ])
    print('Input matrix')
    print(a)
    print()
    try:
        gram_schmidt(a)  # should raise an error
    except LinearDependenceError as e:
        print(e)
    print()
```

```python

    print('Test 5: singular matrix')
    a = np.array([
        [1, 2, -1],
        [2, 5, -3],
        [3, 3, 0]
    ])
    print('Input matrix')
    print(a)
    print()
    try:
        gram_schmidt(a)  # should raise an error
    except LinearDependenceError as e:
        print(e)
```

Listing 2: Output for test cases

```
>python -m gs
Test 1: random square matrix
Input matrix
[[0.58801452 0.69910875 0.18815196 0.04380856 0.20501895]
 [0.10606287 0.72724014 0.67940052 0.4738457  0.44829582]
 [0.01910695 0.75259834 0.60244854 0.96177758 0.66436865]
 [0.60662962 0.44915131 0.22535416 0.6701743  0.73576659]
 [0.25799564 0.09554215 0.96090974 0.25176729 0.28216512]]

Orthonormalized matrix
[[ 0.66075857  0.10599106 -0.32621454 -0.56072372 -0.36240445]
 [ 0.11918405  0.62410254  0.17611003 -0.29186203  0.69288743]
 [ 0.02147069  0.73799494  0.08061366  0.44264935 -0.50245942]
 [ 0.68167657 -0.1644837  -0.10793795  0.62668109  0.32230789]
 [ 0.28991262 -0.16604366  0.91892338 -0.10834125 -0.17950537]]

Implementation worked? True

Test 2: random non-square matrix
Input matrix
[[0.76825393 0.7979234  0.5440372 ]
 [0.38270763 0.38165095 0.28582739]
 [0.74026815 0.23898683 0.4377217 ]
 [0.8835387  0.28928114 0.78450686]
 [0.75895366 0.41778538 0.22576877]]

Orthonormalized matrix
[[ 0.47270878  0.73926285  0.17771326]
 [ 0.23548107  0.33566518  0.12907525]
 [ 0.45548905 -0.37843189 -0.14903895]
 [ 0.54364382 -0.44335429  0.57756706]
 [ 0.46698629 -0.03233593 -0.77198527]]

Implementation worked? True

```

```
Test 3: random matrix with too many columns
Input matrix
[[0.42009814 0.06436369 0.59643269 0.83732372 0.89248639]
 [0.20052744 0.50239523 0.89538184 0.25592093 0.86723234]
 [0.01648793 0.55249695 0.52790539 0.92335039 0.24594844]]

Matrix has linearly dependent columns (more columns than rows)

Test 4: matrix with first column 0
Input matrix
[[0 1 2]
 [0 3 4]
 [0 5 6]]

Matrix has linearly dependent or almost linearly dependent columns because first
    ↪ column is almost 0

Test 5: singular matrix
Input matrix
[[ 1  2 -1]
 [ 2  5 -3]
 [ 3  3  0]]

Aborting orthogonalization; matrix has linearly dependent or almost linearly dependent
    ↪  columns
```