assignment2-Q1

March 1, 2022

1 Q1) Gram-Schmidt Algorithm and QR decomposition

[1]:
```
import sys
import random
from math import floor, log10, sqrt
import copy
```

i) Write a code to generate a random matrix A of size m × n with m > n and calculate its Frobenius norm, $\| \cdot \|_F$. The entries of A must be of the form r.dddd (example 5.4316). The inputs are the positive integers m and n and the output should display the the dimensions and the calculated norm value.

1.1 Deliverable(s) : The code with the desired input and output (0.5)

[2]:
```
# Significant digit conversion
'''
This method convert a digit into significant digit
x: digit
sig: significant digit form ex: 5 = r.dddd
'''
def tidy(x, sig):
    y = abs(x)
    if y <= sys.float_info.min:
        return 0.0000
    return round(x, sig-int(floor(log10(y)))-1)
```

[3]:
```
# Calculate frobenius norm of matrix m
'''
This method calculate frobenius norm of a matrix
m: input matrix
sig: significant digit form ex: 5 = r.dddd
'''
def norm_frobenius(m, sig=5):
    sqr_sum = 0
    for r in range(len(m)):
        for c in range(len(m[r])):
            elm = abs(m[r][c])
            sqr_sum = tidy(sqr_sum + tidy((elm ** 2), sig), sig)
```

```
    return sqrt(sqr_sum)
```

[4]:
```
# Matrix generator of size m x n
def generate_matrix(m, n, low_num=0.10000, up_num=9.9999, sig=5):
    return [[tidy(random.uniform(low_num, up_num), sig) for i in range(n)] for␣
 ↪j in range(m)]
```

[5]:
```
# Function to run frobenius norm calculator along with matrix generation
'''
Main entry point method for the Q1 part i answer, to calculate frobenius norm␣
 ↪and display input matrix size
m: no of rows
n: no of columns
'''
def run_frobenius_calc(m, n):
    if m >= n:
        a = generate_matrix(m, n)
        print(f"\nMatrix size: {m}x{n}")
        frobenius_norm_val = tidy(norm_frobenius(a), 5)
        print(f"\nFrobenius norm value: {frobenius_norm_val}")
    else:
        print(f"\nPlease provide matrix size, where m > n.")
```

[6]:
```
# Test run
run_frobenius_calc(5, 4)
```

```

Matrix size: 5x4

Frobenius norm value: 22.641
```

ii) Write a code to decide if Gram-Schmidt Algorithm can be applied to columns of a given matrix A through calculation of rank. The code should print appropriate messages indicating whether Gram-Schmidt is applicable on columns of the matrix or not.

1.2 Deliverable(s) : The code that performs the test. (1)

[7]:
```
'''
This method checks if a given matrix is full rank
matrix: input matrix
d: significant digit
'''
def is_full_rank_matrix(matrix, d=5):
    r = len(matrix)
    c = len(matrix[0])

    # Significant digit conversion
    def tidy(x, sig):
```

```
        y = abs(x)
        if y <= sys.float_info.min:
            return 0.0000
        return round(x, sig-int(floor(log10(y)))-1)

    # Function for exchanging two rows of a matrix
    def swap(matrix, row1, row2, col):
        for i in range(col):
            temp = matrix[row1][i]
            matrix[row1][i] = matrix[row2][i]
            matrix[row2][i] = temp

    rank = c
    for row in range(0, rank, 1):
        # Diagonal element is not zero
        if matrix[row][row] != 0:
            for col in range(0, r, 1):
                if col != row:
                    multiplier = tidy((matrix[col][row] / matrix[row][row]), d)
                    for i in range(rank):
                        matrix[col][i] = tidy(matrix[col][i] - tidy((multiplier␣
↪* matrix[row][i]), d), d)
        else:
            reduce = True
            for i in range(row + 1, r, 1):
                if matrix[i][row] != 0:
                    swap(matrix, row, i, rank)
                    reduce = False
                    break
            if reduce:
                rank -= 1
                for i in range(0, r, 1):
                    matrix[i][row] = matrix[i][rank]
            row -= 1
    return True if rank == min(r, c) else False
```

[8]:
```
'''
Main entry point method for the Q1 part ii answer, to check gram schmidt␣
↪applicability on input matrix
m: input matrix
d: significant digit
'''
def gram_schmidt_applicability_calc(m, d=5):
    check = is_full_rank_matrix(m, d)
    if check:
        print("\nGram-Schmidt is applicable on columns of the matrix.")
    else:
```

```python
        print("\nGram-Schmidt is not applicable on columns of the matrix.")
```

[9]:
```python
# Test1
A = generate_matrix(7, 5)
gram_schmidt_applicability_calc(A)
```

```
Gram-Schmidt is applicable on columns of the matrix.
```

[10]:
```python
# Test2
a = [[3, 2, 4],
     [-1, 1, 2],
     [9, 5, 10]]
gram_schmidt_applicability_calc(a, 0)
```

```
Gram-Schmidt is not applicable on columns of the matrix.
```

[11]:
```python
# Test3
a = [[2.0,  1.0, -1.0],
     [-3.0,  -1.0,  2.0],
     [-2.0, 1.0,  2.0]]
gram_schmidt_applicability_calc(a)
```

```
Gram-Schmidt is applicable on columns of the matrix.
```

iii) Write a code to generate the orthogonal matrix Q from a matrix A by performing the Gram-Schmidt orthogonalization method. Ensure that A has linearly independent columns by checking the rank. Keep generating A until the linear independence is obtained.

`Deliverable(s) : The code that produces matrix Q from A (1)`

---

[12]:
```python
'''
This method return the column from a matrix
matrix: input matrix
col: column index
'''
def get_matrix_column(matrix, col):
    column = []
    for row in matrix:
        elem = row[col]
        column.append(elem)
    return column
```

[13]:
```python
'''
This method set given column value in a matrix to given index
matrix: input matrix
```

```
n: no of rows
col_idx: column index to be set
col: column vector
'''
def set_matrix_column(matrix, n, col_idx, col):
    for row_idx in range(n):
        matrix[row_idx][col_idx] = col[row_idx]
    return matrix
```

[14]:
```
'''
This method calculates the dot product of gievn input vectors
x: vector a
y: vector b
sig: significant digit
'''
def inner_dot(x, y, sig=5):
    return tidy(sum(tidy(x_i * y_i, sig) for x_i, y_i in zip(x, y)), sig)
```

[15]:
```
'''
This method calculate the Q and R matrices using gram schmidt method
matrix: input matrix A
m: no of rows
n: no of columns
d: significant digit
'''
def gram_schmidt(matrix, m, n, d=5):
    n_add, n_mul, n_div = 0, 0, 0
    # Initialize Q and R matrices
    q = [[0 for x in range(n)] for y in range(m)]
    r = [[0 for x in range(n)] for y in range(n)]
    for j in range(n):
        # Step-1, v1 = a1
        v = get_matrix_column(matrix, j)
        # Skip the first column
        if j > 0:
            for i in range(j):
                # Find the inner product
                r[i][j] = inner_dot(get_matrix_column(q, i),␣
↪get_matrix_column(matrix, j))
                n_add = n_add + m - 1
                n_mul = n_mul + m
                # Subtract the projection from v which causes v to become␣
↪perpendicular to all columns of Q
                v=[tidy(x_i - y_i, d) for x_i, y_i in zip(v, [tidy(r[i][j] * x,␣
↪d) for x in get_matrix_column(q, i)])]
                n_mul = n_mul + m
                n_add = n_add + m - 1
```

```python
        # Find the L2 norm of the jth diagonal of R
        r[j][j] = tidy(sqrt(tidy(sum([tidy(x**2, d) for x in v]), d)), d)
        n_mul = n_mul + m + 1
        n_add = n_add + m - 1
        # The orthogonalized result is found and stored in the ith column of Q.
        q = set_matrix_column(q, n, j, [tidy(x / r[j][j], d) for x in v])
        n_div = n_div + m
    return (q, r, n_add, n_mul, n_div)
```

[16]:
```python
# Matrix generator of size m x n
def generate_matrix(m, n, low_num=0.10000, up_num=9.9999, sig=5):
    return [[tidy(random.uniform(low_num, up_num), sig) for i in range(n)] for␣
 ↪j in range(m)]
```

[17]:
```python
# Pretty print matrix
def pritty_print_matrix(mat):
    print("[", end="")
    for row_idx in range(len(mat)):
        if row_idx == 0:
            print(f"{mat[row_idx]},")
        elif row_idx == (len(mat) -1):
            print(f" {mat[row_idx]}]\n")
        else:
            print(f" {mat[row_idx]},")
```

[18]:
```python
'''
Main entry point method for the Q1 part iii answer, to calculate Q and R␣
 ↪matrices from input matrix
m: no of rows
n: no of columns
max_itr: no of maximum iteration untill finds linearly independent columns
'''
def run_gram_schmidt_calc(m=3, n=3, max_itr=100):
    itr = 1
    while True and itr <= max_itr:
        matrix = generate_matrix(m,n)
        matrix_ = copy.deepcopy(matrix)
        print("Input Matrix:")
        pritty_print_matrix(matrix)
        if is_full_rank_matrix(matrix_):
            q, r, n_add, n_mul, n_div = gram_schmidt(matrix, m, n)
            print("Q Matrix:")
            pritty_print_matrix(q)
            print("R Matrix:")
            pritty_print_matrix(r)
            break
        else:
```

```
            print("\nGram-Schmidt is not applicable as generated Matrix does␣
↪not have linearly independent columns.")
```

[19]:
```
# Test1
run_gram_schmidt_calc(m=7, n=5)
```

```
Input Matrix:
[[9.6216, 1.7227, 4.4558, 3.7599, 3.0948],
 [3.6333, 8.667, 0.25117, 5.1295, 7.8489],
 [6.5194, 3.942, 3.2745, 1.3036, 4.2306],
 [1.3414, 9.6932, 2.0212, 1.4593, 6.4957],
 [1.0318, 2.1298, 8.4936, 2.1076, 1.0396],
 [0.52691, 4.7483, 9.951, 9.8365, 3.8188],
 [1.002, 8.3868, 8.9438, 3.0133, 8.1894]]

Q Matrix:
[[0.77933, -0.25524, -0.0074464, 0.0089938, -0.13378],
 [0.29429, 0.42918, -0.12873, 0.23911, 0.2574],
 [0.52806, 0.0089687, -0.00072409, -0.14821, 0.025009],
 [0.10865, 0.58417, 0.047511, -0.047074, 0.19764],
 [0.083574, 0.10015, 0.50228, 0.11092, -0.028755],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0]]

R Matrix:
[[12.346, 7.206, 6.205, 5.4629, 7.7484],
 [0, 15.253, 1.0312, 2.3171, 6.5153],
 [0, 0, 15.672, 0.43867, -0.20572],
 [0, 0, 0, 10.806, 1.0871],
 [0, 0, 0, 0, 9.6581]]
```

iv) Write a code to create a QR decomposition of the matrix A by utilizing the code developed in the previous sub-parts of this question. Find the matrices Q and R and then display the value $A - (Q.R) F$, where $\cdot$ F is the Frobenius norm. The code should also display the total number of additions, multiplications and divisions to find the result.

```
Deliverable(s) : The code with the said input and output. The results obtained fo
    generated with m = 7 and n = 5 with random entries described above. (2.5)
```

---

[20]:
```
def get_matrix_multiplication(mat1, mat2):
    result = [[sum(a * b for a, b in zip(mat1_row, mat2_col)) for mat2_col in␣
↪zip(*mat2)] for mat1_row in mat1]
    return result
```

[21]:
```
def get_matrix_subtraction(mat1, mat2):
```

```
    result = [[mat1[m][n] - mat2[m][n] for n in range(len(mat1[0]))] for m in␣
↪range(len(mat1))]
    return result
```

[22]:
```
'''
Main entry point method for the Q1 part iv answer, to calculate Q and R␣
↪matrices from input matrix and no of operations
m: no of rows
n: no of columns
max_itr: no of maximum iteration untill finds linearly independent columns
'''
def run_gram_schmidt_calc(m=7, n=5, max_itr=100):
    itr = 1 # Iteration for if randomaly generated matrices are not linearly␣
↪independent than stop at 100th Itr
    while True and itr <= max_itr:
        matrix = generate_matrix(m,n)
        matrix_ = copy.deepcopy(matrix)
        a_ = copy.deepcopy(matrix)
        print("Input Matrix:")
        pritty_print_matrix(matrix)
        if is_full_rank_matrix(matrix_):
            q, r, n_add, n_mul, n_div = gram_schmidt(matrix, m, n)
            qr = get_matrix_multiplication(q, r)
            a_minus_qr = get_matrix_subtraction(a_, qr)
            f_norm_of_a_minus_qr = tidy(norm_frobenius(a_minus_qr), 5)
            print("Q Matrix:")
            pritty_print_matrix(q)
            print("R Matrix:")
            pritty_print_matrix(r)
            print(f"\nNo. of Addition: {n_add}\nNo. of Multiplication:␣
↪{n_mul}\nNo. of Division: {n_div}")
            print(f"\nTotal operations: {n_add + n_div + n_mul}")
            print(f"\nFrobenius Norm of A - QR: {f_norm_of_a_minus_qr}")
            break
        else:
            print("\nGram-Schmidt is not applicable as generated Matrix does␣
↪not have linearly independent columns.")
```

[23]:
```
# Test1
run_gram_schmidt_calc()
```

```
Input Matrix:
[[0.72069, 5.169, 6.3765, 1.0047, 4.1166],
 [1.003, 6.9367, 2.027, 4.7047, 6.1976],
 [6.6766, 2.379, 2.5188, 5.3242, 3.5468],
 [3.8311, 9.3188, 7.8407, 5.2528, 0.39205],
 [7.8655, 1.2349, 7.3091, 2.5664, 4.6747],
```

```
 [7.2353, 3.1841, 8.3622, 1.9928, 3.2635],
 [5.6851, 8.2997, 2.2291, 7.1676, 5.1065]]

Q Matrix:
[[0.050051, 0.33727, 0.35602, -0.16706, 0.25177],
 [0.069658, 0.45176, -0.14226, 0.25823, 0.45097],
 [0.46368, 0.0041827, -0.10257, 0.33354, 0.094145],
 [0.26607, 0.54777, 0.20492, 0.086123, -0.42592],
 [0.54625, -0.10257, 0.35801, -0.047844, 0.20598],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0]]

R Matrix:
[[14.399, 4.9991, 7.707, 5.6462, 4.9403],
 [0, 14.584, 6.6221, 5.1006, 3.9383],
 [0, 0, 10.554, 1.1375, 1.9741],
 [0, 0, 0, 8.3996, 1.9058],
 [0, 0, 0, 0, 8.5659]]


No. of Addition: 150
No. of Multiplication: 180
No. of Division: 35

Total operations: 365

Frobenius Norm of A - QR: 18.184
```