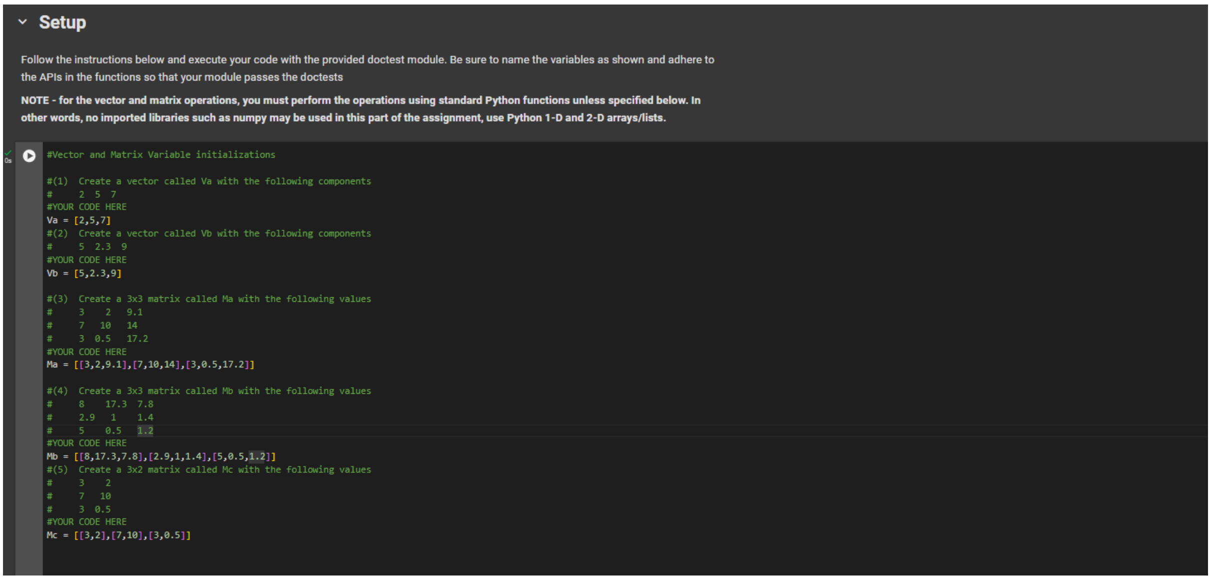
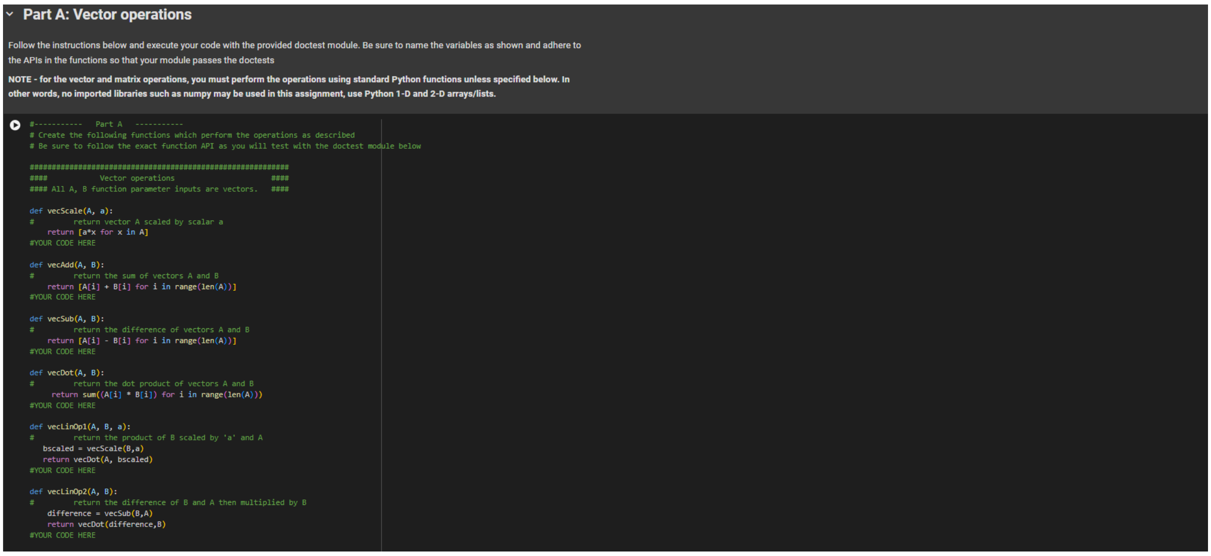


Setup

Follow the instructions below and execute your code with the provided doctest module. Be sure to name the variables as shown and adhere to the APIs in the functions so that your module passes the doctests

NOTE - for the vector and matrix operations, you must perform the operations using standard Python functions unless specified below. In other words, no imported libraries such as numpy may be used in this part of the assignment, use Python 1-D and 2-D arrays/lists.

```python
#Vector and Matrix Variable initializations

#(1)  Create a vector called Va with the following components
#     2  5  7
#YOUR CODE HERE
Va = [2,5,7]
#(2)  Create a vector called Vb with the following components
#     5  2.3  9
#YOUR CODE HERE
Vb = [5,2.3,9]

#(3)  Create a 3x3 matrix called Ma with the following values
#     3    2   9.1
#     7   10   14
#     3  0.5   17.2
#YOUR CODE HERE
Ma = [[3,2,9.1],[7,10,14],[3,0.5,17.2]]

#(4)  Create a 3x3 matrix called Mb with the following values
#     8    17.3  7.8
#     2.9   1    1.4
#     5    0.5   1.2
#YOUR CODE HERE
Mb = [[8,17.3,7.8],[2.9,1,1.4],[5,0.5,1.2]]
#(5)  Create a 3x2 matrix called Mc with the following values
#     3    2
#     7   10
#     3  0.5
#YOUR CODE HERE
Mc = [[3,2],[7,10],[3,0.5]]
```

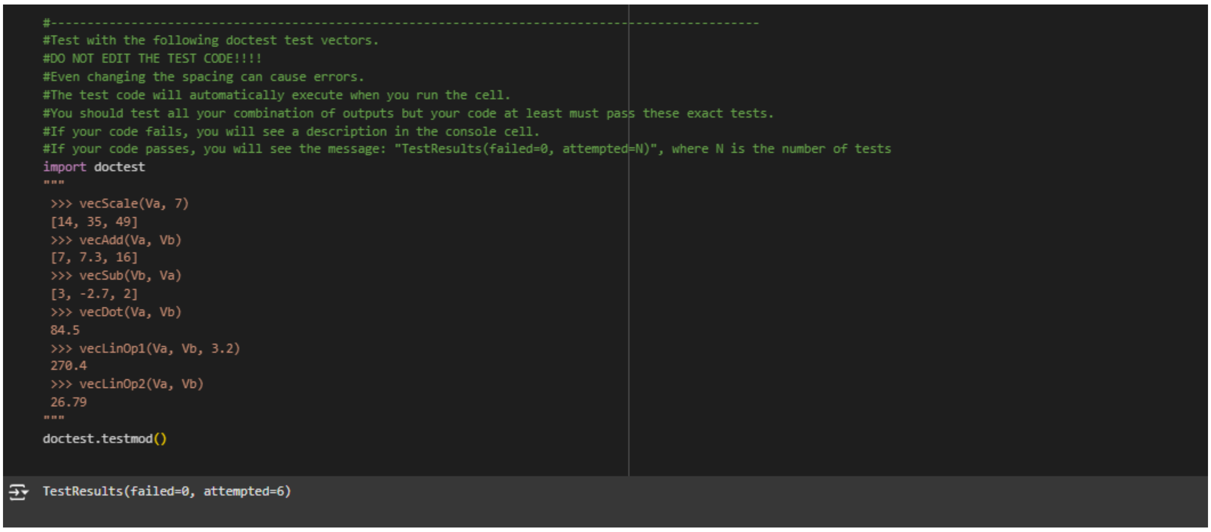


Part A: Vector operations

Follow the instructions below and execute your code with the provided doctest module. Be sure to name the variables as shown and adhere to the APIs in the functions so that your module passes the doctests

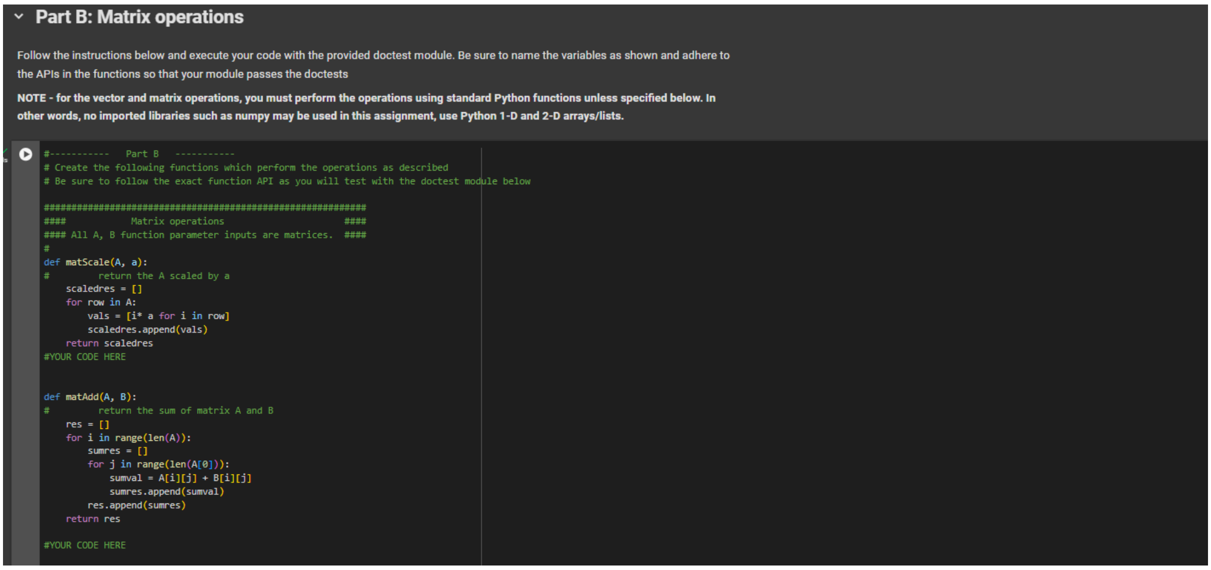
NOTE - for the vector and matrix operations, you must perform the operations using standard Python functions unless specified below. In other words, no imported libraries such as numpy may be used in this assignment, use Python 1-D and 2-D arrays/lists.

```python
#----------   Part A   ----------
# Create the following functions which perform the operations as described
# Be sure to follow the exact function API as you will test with the doctest module below

############################################################
####            Vector operations                       ####
#### All A, B function parameter inputs are vectors.    ####

def vecScale(A, a):
#         return vector A scaled by scalar a
    return [a*x for x in A]
#YOUR CODE HERE

def vecAdd(A, B):
#         return the sum of vectors A and B
    return [A[i] + B[i] for i in range(len(A))]
#YOUR CODE HERE

def vecSub(A, B):
#         return the difference of vectors A and B
    return [A[i] - B[i] for i in range(len(A))]
#YOUR CODE HERE

def vecDot(A, B):
#         return the dot product of vectors A and B
     return sum((A[i] * B[i]) for i in range(len(A)))
#YOUR CODE HERE

def vecLinOp1(A, B, a):
#         return the product of B scaled by 'a' and A
   bscaled = vecScale(B,a)
   return vecDot(A, bscaled)
#YOUR CODE HERE

def vecLinOp2(A, B):
#         return the difference of B and A then multiplied by B
    difference = vecSub(B,A)
    return vecDot(difference,B)
#YOUR CODE HERE
```



```python
#-------------------------------------------------------------------------------------------------
#Test with the following doctest test vectors.
#DO NOT EDIT THE TEST CODE!!!!
#Even changing the spacing can cause errors.
#The test code will automatically execute when you run the cell.
#You should test all your combination of outputs but your code at least must pass these exact tests.
#If your code fails, you will see a description in the console cell.
#If your code passes, you will see the message: "TestResults(failed=0, attempted=N)", where N is the number of tests
import doctest
"""
 >>> vecScale(Va, 7)
 [14, 35, 49]
 >>> vecAdd(Va, Vb)
 [7, 7.3, 16]
 >>> vecSub(Vb, Va)
 [3, -2.7, 2]
 >>> vecDot(Va, Vb)
 84.5
 >>> vecLinOp1(Va, Vb, 3.2)
 270.4
 >>> vecLinOp2(Va, Vb)
 26.79
"""
doctest.testmod()
```

```
TestResults(failed=0, attempted=6)
```

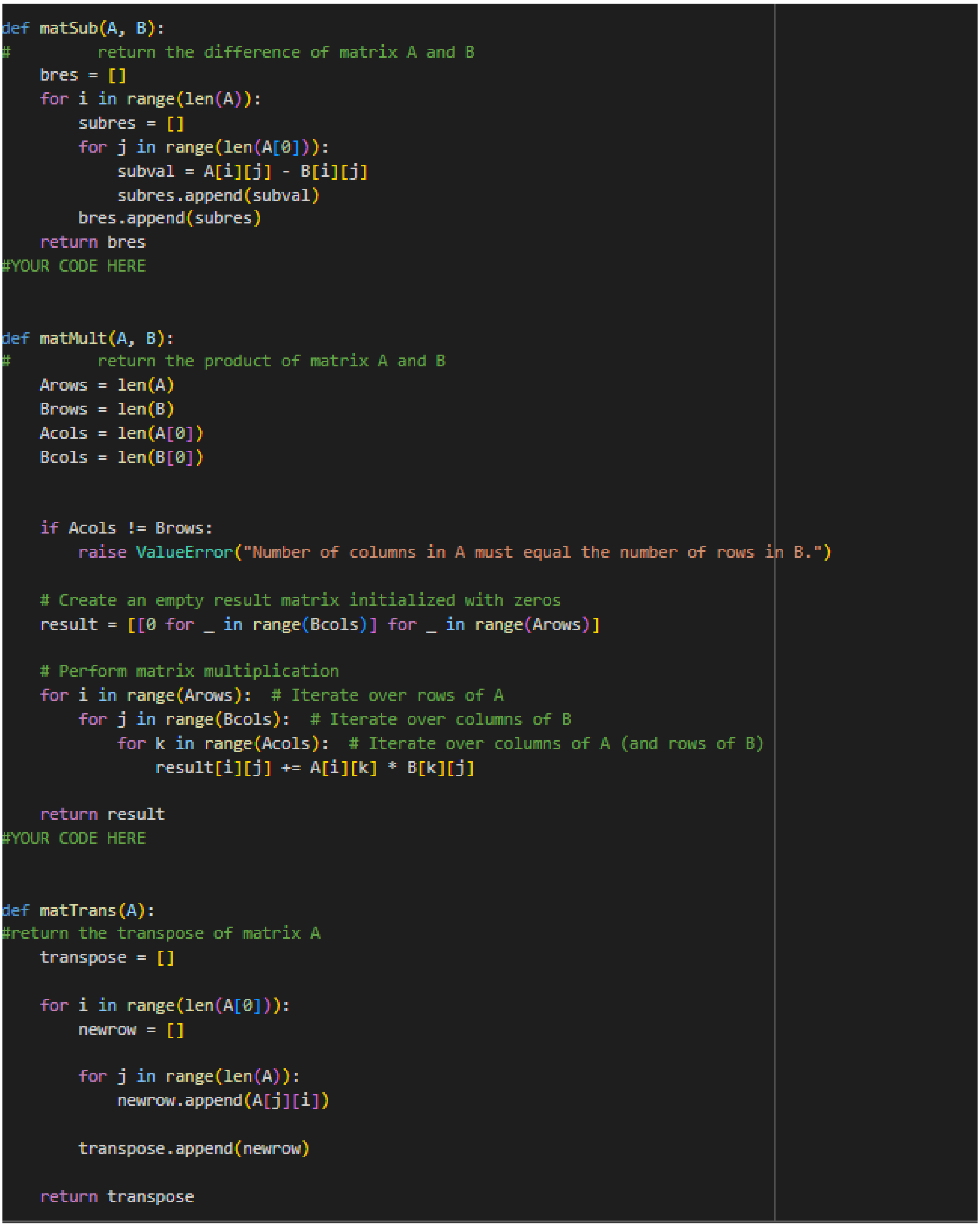


Part B: Matrix operations

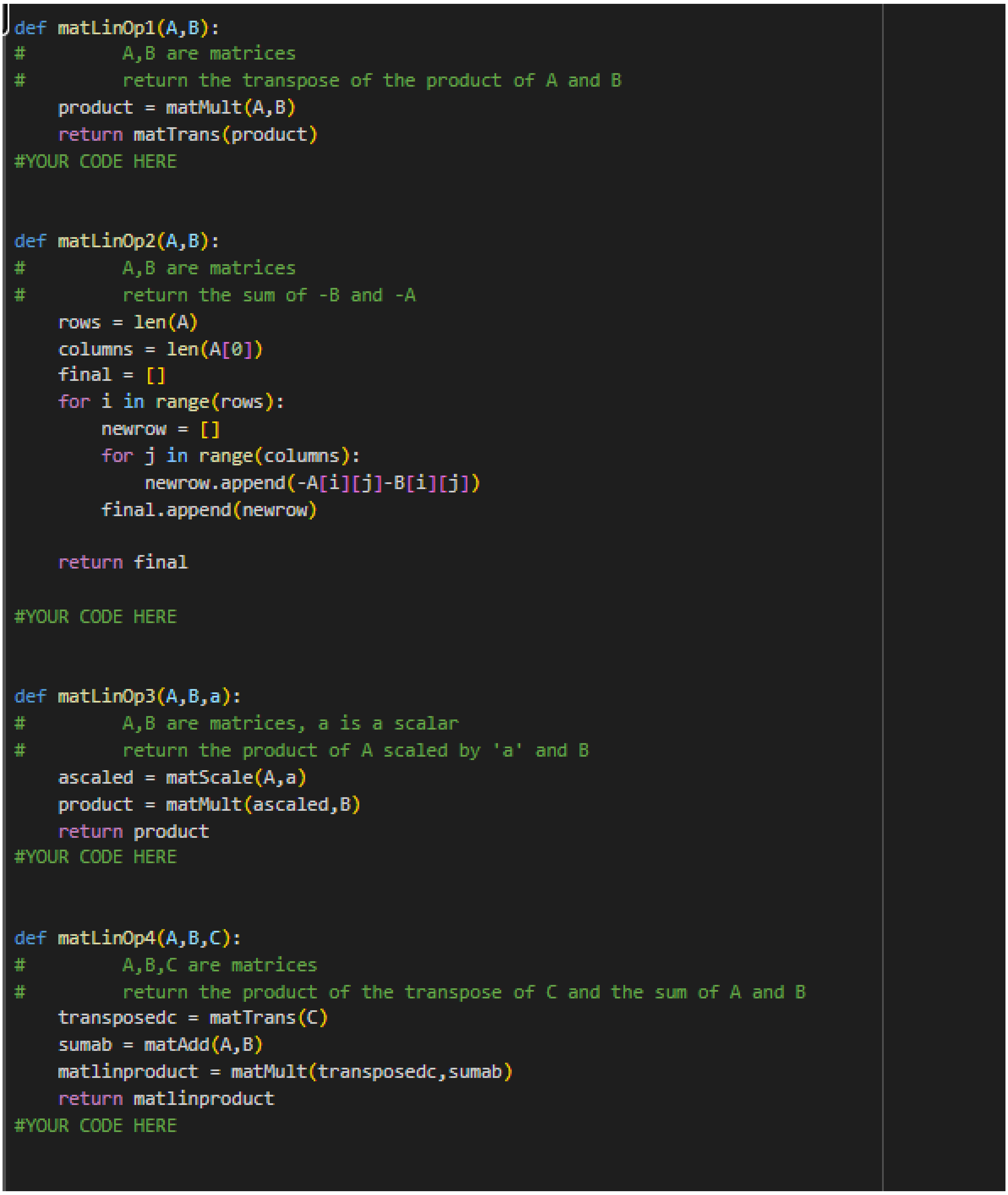
Follow the instructions below and execute your code with the provided doctest module. Be sure to name the variables as shown and adhere to the APIs in the functions so that your module passes the doctests

NOTE - for the vector and matrix operations, you must perform the operations using standard Python functions unless specified below. In other words, no imported libraries such as numpy may be used in this assignment, use Python 1-D and 2-D arrays/lists.

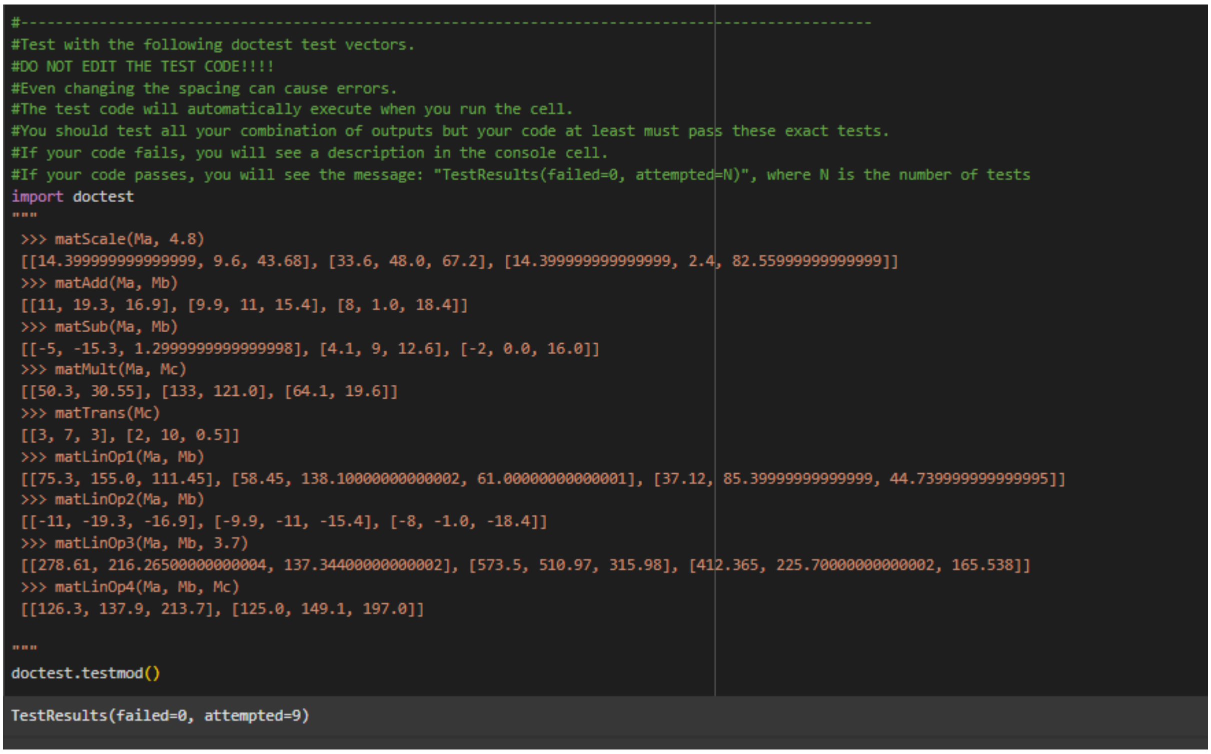
```
#----------   Part B   ----------
# Create the following functions which perform the operations as described
# Be sure to follow the exact function API as you will test with the doctest module below

###########################################################
####            Matrix operations                     ####
#### All A, B function parameter inputs are matrices.  ####
#
def matScale(A, a):
#         return the A scaled by a
    scaledres = []
    for row in A:
        vals = [i* a for i in row]
        scaledres.append(vals)
    return scaledres
#YOUR CODE HERE


def matAdd(A, B):
#         return the sum of matrix A and B
    res = []
    for i in range(len(A)):
        sumres = []
        for j in range(len(A[0])):
            sumval = A[i][j] + B[i][j]
            sumres.append(sumval)
        res.append(sumres)
    return res

#YOUR CODE HERE
```



```
def matSub(A, B):
#         return the difference of matrix A and B
    bres = []
    for i in range(len(A)):
        subres = []
        for j in range(len(A[0])):
            subval = A[i][j] - B[i][j]
            subres.append(subval)
        bres.append(subres)
    return bres
#YOUR CODE HERE


def matMult(A, B):
#         return the product of matrix A and B
    Arows = len(A)
    Brows = len(B)
    Acols = len(A[0])
    Bcols = len(B[0])


    if Acols != Brows:
        raise ValueError("Number of columns in A must equal the number of rows in B.")

    # Create an empty result matrix initialized with zeros
    result = [[0 for _ in range(Bcols)] for _ in range(Arows)]

    # Perform matrix multiplication
    for i in range(Arows):  # Iterate over rows of A
        for j in range(Bcols):  # Iterate over columns of B
            for k in range(Acols):  # Iterate over columns of A (and rows of B)
                result[i][j] += A[i][k] * B[k][j]

    return result
#YOUR CODE HERE


def matTrans(A):
#return the transpose of matrix A
    transpose = []

    for i in range(len(A[0])):
        newrow = []

        for j in range(len(A)):
            newrow.append(A[j][i])

        transpose.append(newrow)

    return transpose
```

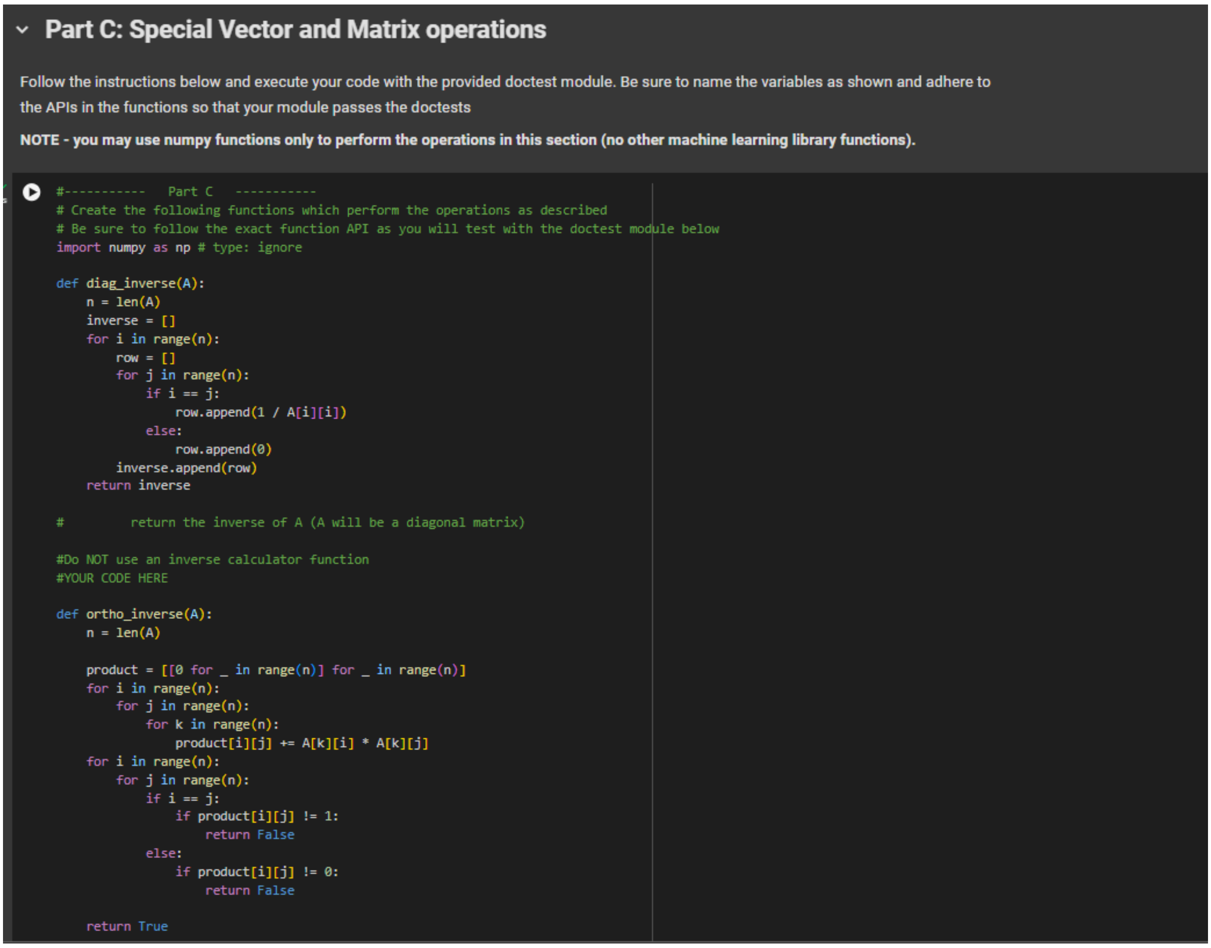


```
def matLinOp1(A,B):
#         A,B are matrices
#         return the transpose of the product of A and B
    product = matMult(A,B)
    return matTrans(product)
#YOUR CODE HERE


def matLinOp2(A,B):
#         A,B are matrices
#         return the sum of -B and -A
    rows = len(A)
    columns = len(A[0])
    final = []
    for i in range(rows):
        newrow = []
        for j in range(columns):
            newrow.append(-A[i][j]-B[i][j])
        final.append(newrow)

    return final

#YOUR CODE HERE


def matLinOp3(A,B,a):
#         A,B are matrices, a is a scalar
#         return the product of A scaled by 'a' and B
    ascaled = matScale(A,a)
    product = matMult(ascaled,B)
    return product
#YOUR CODE HERE


def matLinOp4(A,B,C):
#         A,B,C are matrices
#         return the product of the transpose of C and the sum of A and B
    transposedc = matTrans(C)
    sumab = matAdd(A,B)
    matlinproduct = matMult(transposedc,sumab)
    return matlinproduct
#YOUR CODE HERE
```



```
#-----------------------------------------------------------------------------------------------
#Test with the following doctest test vectors.
#DO NOT EDIT THE TEST CODE!!!!
#Even changing the spacing can cause errors.
#The test code will automatically execute when you run the cell.
#You should test all your combination of outputs but your code at least must pass these exact tests.
#If your code fails, you will see a description in the console cell.
#If your code passes, you will see the message: "TestResults(failed=0, attempted=N)", where N is the number of tests
import doctest
"""
 >>> matScale(Ma, 4.8)
 [[14.399999999999999, 9.6, 43.68], [33.6, 48.0, 67.2], [14.399999999999999, 2.4, 82.55999999999999]]
 >>> matAdd(Ma, Mb)
 [[11, 19.3, 16.9], [9.9, 11, 15.4], [8, 1.0, 18.4]]
 >>> matSub(Ma, Mb)
 [[-5, -15.3, 1.2999999999999998], [4.1, 9, 12.6], [-2, 0.0, 16.0]]
 >>> matMult(Ma, Mc)
 [[50.3, 30.55], [133, 121.0], [64.1, 19.6]]
 >>> matTrans(Mc)
 [[3, 7, 3], [2, 10, 0.5]]
 >>> matLinOp1(Ma, Mb)
 [[75.3, 155.0, 111.45], [58.45, 138.10000000000002, 61.00000000000001], [37.12, 85.39999999999999, 44.739999999999995]]
 >>> matLinOp2(Ma, Mb)
 [[-11, -19.3, -16.9], [-9.9, -11, -15.4], [-8, -1.0, -18.4]]
 >>> matLinOp3(Ma, Mb, 3.7)
 [[278.61, 216.26500000000004, 137.34400000000002], [573.5, 510.97, 315.98], [412.365, 225.70000000000002, 165.538]]
 >>> matLinOp4(Ma, Mb, Mc)
 [[126.3, 137.9, 213.7], [125.0, 149.1, 197.0]]

"""
doctest.testmod()
```

```
TestResults(failed=0, attempted=9)
```

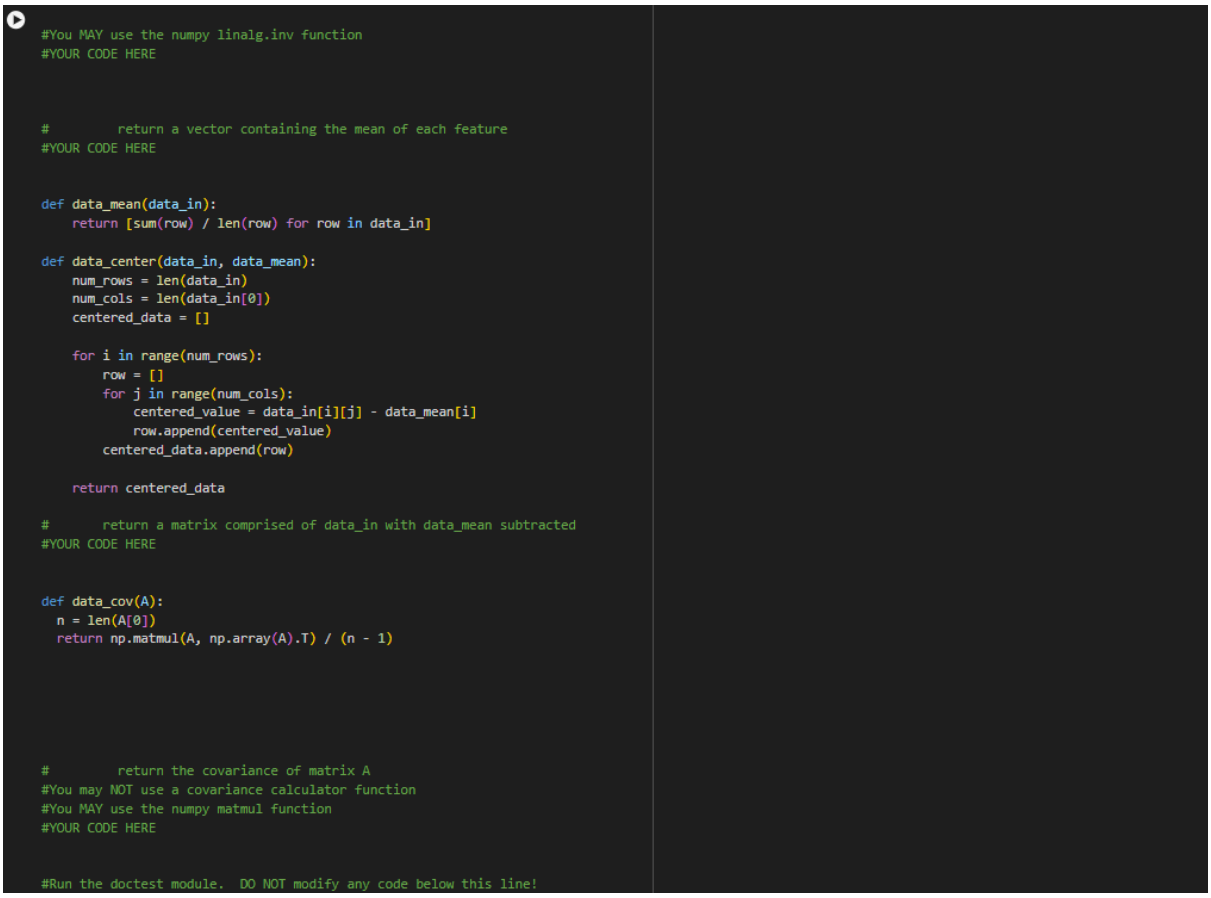


Part C: Special Vector and Matrix operations

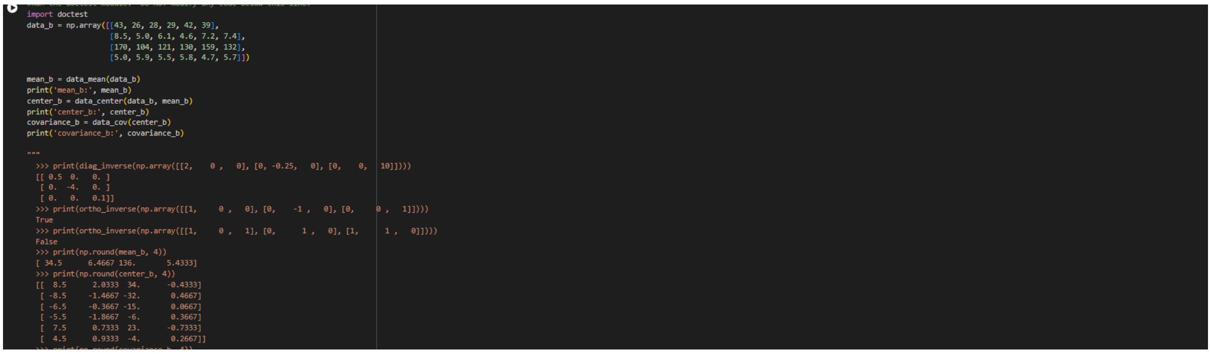
Follow the instructions below and execute your code with the provided doctest module. Be sure to name the variables as shown and adhere to the APIs in the functions so that your module passes the doctests

NOTE - you may use numpy functions only to perform the operations in this section (no other machine learning library functions).

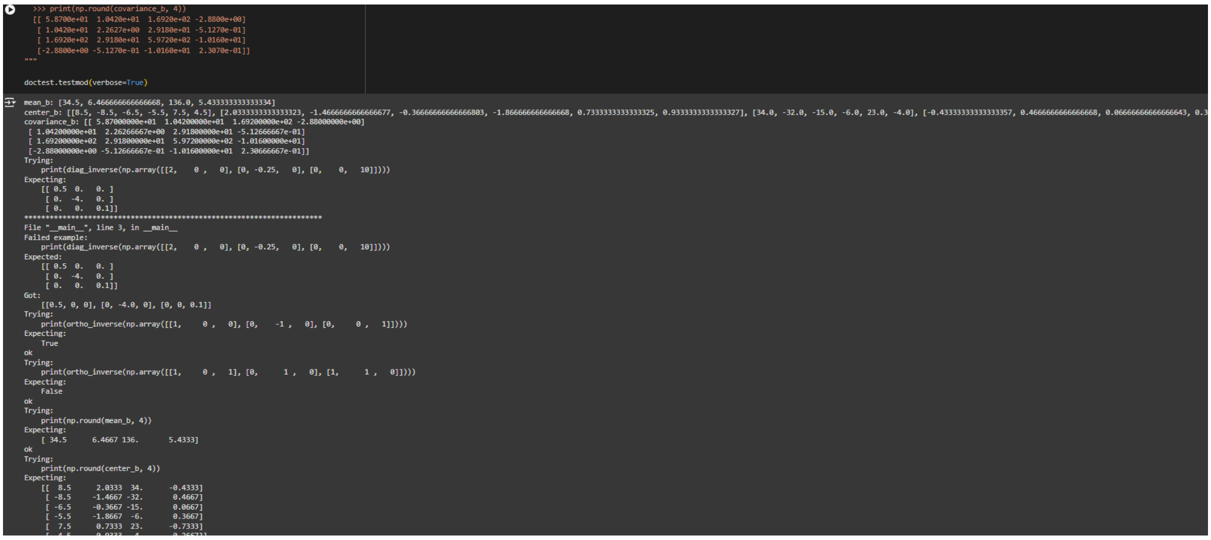
```
#----------   Part C   ----------
# Create the following functions which perform the operations as described
# Be sure to follow the exact function API as you will test with the doctest module below
import numpy as np # type: ignore

def diag_inverse(A):
    n = len(A)
    inverse = []
    for i in range(n):
        row = []
        for j in range(n):
            if i == j:
                row.append(1 / A[i][i])
            else:
                row.append(0)
        inverse.append(row)
    return inverse

#        return the inverse of A (A will be a diagonal matrix)

#Do NOT use an inverse calculator function
#YOUR CODE HERE

def ortho_inverse(A):
    n = len(A)

    product = [[0 for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            for k in range(n):
                product[i][j] += A[k][i] * A[k][j]
    for i in range(n):
        for j in range(n):
            if i == j:
                if product[i][j] != 1:
                    return False
            else:
                if product[i][j] != 0:
                    return False

    return True
```



```
#You MAY use the numpy linalg.inv function
#YOUR CODE HERE



#         return a vector containing the mean of each feature
#YOUR CODE HERE


def data_mean(data_in):
    return [sum(row) / len(row) for row in data_in]

def data_center(data_in, data_mean):
    num_rows = len(data_in)
    num_cols = len(data_in[0])
    centered_data = []

    for i in range(num_rows):
        row = []
        for j in range(num_cols):
            centered_value = data_in[i][j] - data_mean[i]
            row.append(centered_value)
        centered_data.append(row)

    return centered_data

#       return a matrix comprised of data_in with data_mean subtracted
#YOUR CODE HERE


def data_cov(A):
  n = len(A[0])
  return np.matmul(A, np.array(A).T) / (n - 1)





#         return the covariance of matrix A
#You may NOT use a covariance calculator function
#You MAY use the numpy matmul function
#YOUR CODE HERE


#Run the doctest module.  DO NOT modify any code below this line!
```

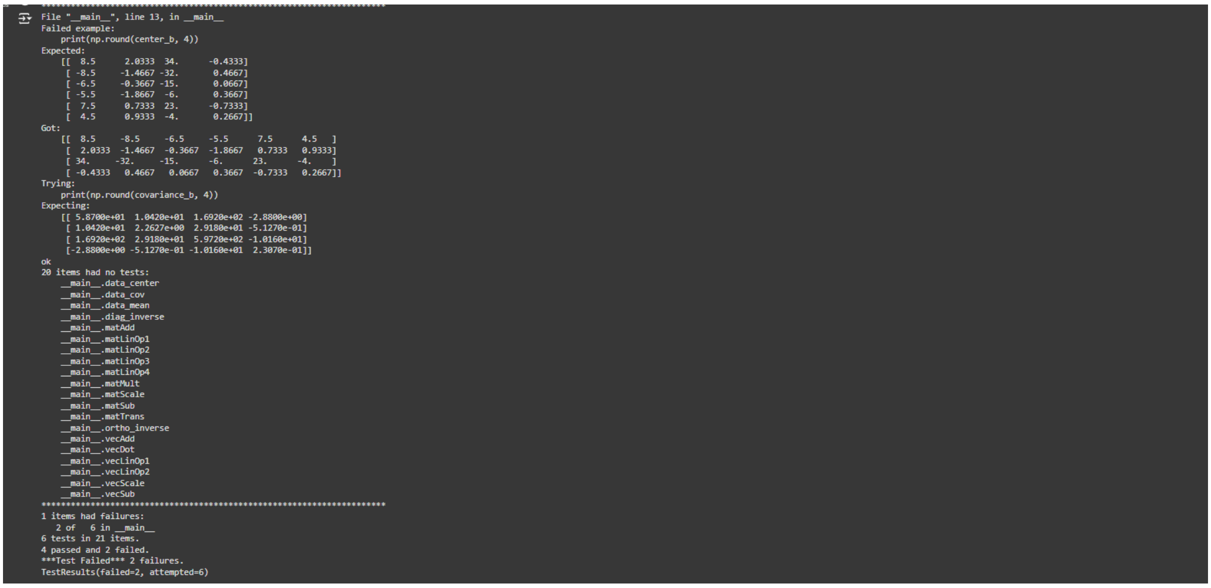


```
#Run the doctest module.  DO NOT modify any code below this line!
import doctest
data_b = np.array([[43, 26, 28, 29, 42, 39],
                   [8.5, 5.0, 6.1, 4.6, 7.2, 7.4],
                   [170, 104, 121, 130, 159, 132],
                   [5.0, 5.9, 5.5, 5.8, 4.7, 5.7]])

mean_b = data_mean(data_b)
print('mean_b:', mean_b)
center_b = data_center(data_b, mean_b)
print('center_b:', center_b)
covariance_b = data_cov(center_b)
print('covariance_b:', covariance_b)

"""
  >>> print(diag_inverse(np.array([[2,    0 ,   0], [0, -0.25,   0], [0,    0,   10]])))
  [[ 0.5  0.   0. ]
   [ 0.  -4.   0. ]
   [ 0.   0.   0.1]]
  >>> print(ortho_inverse(np.array([[1,     0 ,   0], [0,    -1 ,   0], [0,     0 ,   1]])))
  True
  >>> print(ortho_inverse(np.array([[1,     0 ,   1], [0,      1 ,   0], [1,      1 ,   0]])))
  False
  >>> print(np.round(mean_b, 4))
  [ 34.5      6.4667 136.       5.4333]
  >>> print(np.round(center_b, 4))
  [[  8.5      2.0333  34.      -0.4333]
   [ -8.5     -1.4667 -32.       0.4667]
   [ -6.5     -0.3667 -15.       0.0667]
   [ -5.5     -1.8667  -6.       0.3667]
   [  7.5      0.7333  23.      -0.7333]
   [  4.5      0.9333  -4.       0.2667]]
  >>> print(np.round(covariance_b, 4))
```



```
  >>> print(np.round(covariance_b, 4))
  [[ 5.8700e+01  1.0420e+01  1.6920e+02 -2.8800e+00]
   [ 1.0420e+01  2.2627e+00  2.9180e+01 -5.1270e-01]
   [ 1.6920e+02  2.9180e+01  5.9720e+02 -1.0160e+01]
   [-2.8800e+00 -5.1270e-01 -1.0160e+01  2.3070e-01]]
"""

doctest.testmod(verbose=True)
```

```
mean_b: [34.5, 6.466666666666668, 136.0, 5.433333333333334]
center_b: [[8.5, -8.5, -6.5, -5.5, 7.5, 4.5], [2.0333333333333323, -1.4666666666666677, -0.3666666666666803, -1.866666666666668, 0.7333333333333325, 0.9333333333333327], [34.0, -32.0, -15.0, -6.0, 23.0, -4.0], [-0.43333333333333357, 0.4666666666666668, 0.06666666666666643, 0.3
covariance_b: [[ 5.87000000e+01  1.04200000e+01  1.69200000e+02 -2.88000000e+00]
 [ 1.04200000e+01  2.26266667e+00  2.91800000e+01 -5.12666667e-01]
 [ 1.69200000e+02  2.91800000e+01  5.97200000e+02 -1.01600000e+01]
 [-2.88000000e+00 -5.12666667e-01 -1.01600000e+01  2.30666667e-01]]
Trying:
    print(diag_inverse(np.array([[2,    0 ,   0], [0, -0.25,   0], [0,    0,   10]])))
Expecting:
    [[ 0.5  0.   0. ]
     [ 0.  -4.   0. ]
     [ 0.   0.   0.1]]
**********************************************************************
File "__main__", line 3, in __main__
Failed example:
    print(diag_inverse(np.array([[2,    0 ,   0], [0, -0.25,   0], [0,    0,   10]])))
Expected:
    [[ 0.5  0.   0. ]
     [ 0.  -4.   0. ]
     [ 0.   0.   0.1]]
Got:
    [[0.5, 0, 0], [0, -4.0, 0], [0, 0, 0.1]]
Trying:
    print(ortho_inverse(np.array([[1,     0 ,   0], [0,    -1 ,   0], [0,     0 ,   1]])))
Expecting:
    True
ok
Trying:
    print(ortho_inverse(np.array([[1,     0 ,   1], [0,      1 ,   0], [1,      1 ,   0]])))
Expecting:
    False
ok
Trying:
    print(np.round(mean_b, 4))
Expecting:
    [ 34.5      6.4667 136.       5.4333]
ok
Trying:
    print(np.round(center_b, 4))
Expecting:
    [[  8.5      2.0333  34.      -0.4333]
     [ -8.5     -1.4667 -32.       0.4667]
     [ -6.5     -0.3667 -15.       0.0667]
     [ -5.5     -1.8667  -6.       0.3667]
     [  7.5      0.7333  23.      -0.7333]
     [  4.5      0.9333  -4.       0.2667]]
```



```
**********************************************************************
File "__main__", line 13, in __main__
Failed example:
    print(np.round(center_b, 4))
Expected:
    [[  8.5      2.0333  34.      -0.4333]
     [ -8.5     -1.4667 -32.       0.4667]
     [ -6.5     -0.3667 -15.       0.0667]
     [ -5.5     -1.8667  -6.       0.3667]
     [  7.5      0.7333  23.      -0.7333]
     [  4.5      0.9333  -4.       0.2667]]
Got:
    [[  8.5     -8.5     -6.5     -5.5      7.5      4.5   ]
     [  2.0333  -1.4667  -0.3667  -1.8667   0.7333   0.9333]
     [ 34.     -32.     -15.      -6.      23.      -4.    ]
     [ -0.4333   0.4667   0.0667   0.3667  -0.7333   0.2667]]
Trying:
    print(np.round(covariance_b, 4))
Expecting:
    [[ 5.8700e+01  1.0420e+01  1.6920e+02 -2.8800e+00]
     [ 1.0420e+01  2.2627e+00  2.9180e+01 -5.1270e-01]
     [ 1.6920e+02  2.9180e+01  5.9720e+02 -1.0160e+01]
     [-2.8800e+00 -5.1270e-01 -1.0160e+01  2.3070e-01]]
ok
20 items had no tests:
    __main__.data_center
    __main__.data_cov
    __main__.data_mean
    __main__.diag_inverse
    __main__.matAdd
    __main__.matLinOp1
    __main__.matLinOp2
    __main__.matLinOp3
    __main__.matLinOp4
    __main__.matMult
    __main__.matScale
    __main__.matSub
    __main__.matTrans
    __main__.ortho_inverse
    __main__.vecAdd
    __main__.vecDot
    __main__.vecLinOp1
    __main__.vecLinOp2
    __main__.vecScale
    __main__.vecSub
**********************************************************************
1 items had failures:
   2 of   6 in __main__
6 tests in 21 items.
4 passed and 2 failed.
***Test Failed*** 2 failures.
TestResults(failed=2, attempted=6)
```

Passed, only failures were minor differences in formatting