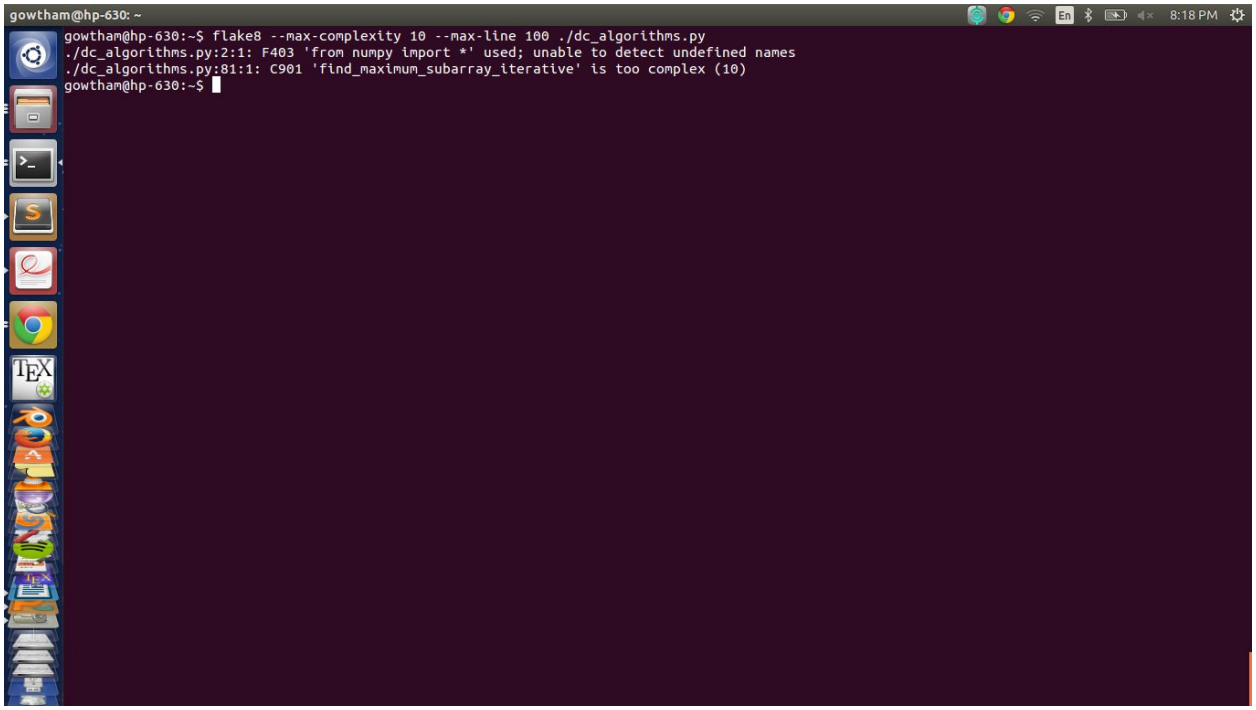
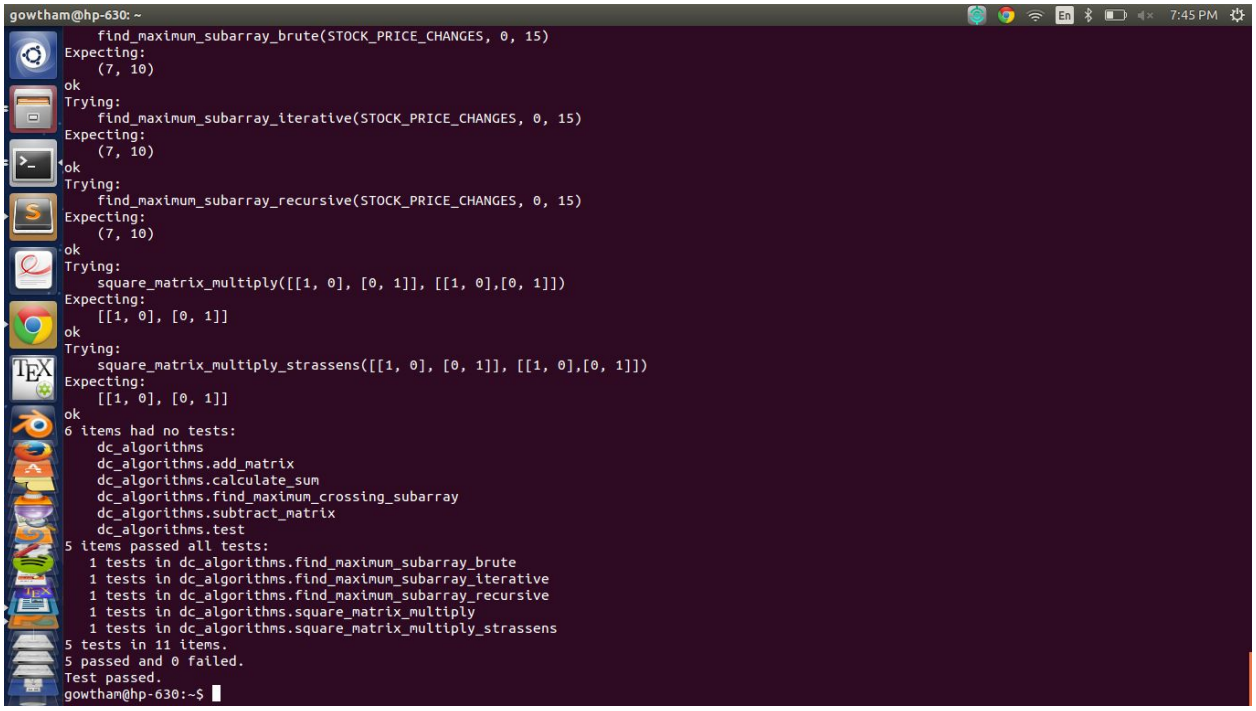
Flake8 Complexity



Doc Test:



Code for Assignment 1

```
# -*- coding: utf-8 -*-
from numpy import *  # analysis:ignore
# STOCK_PRICES  = [100,113,110,85,105,102,86,63,81,101,94,106,101,79,94,90,97]
STOCK_PRICE_CHANGES = [13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]


def find_maximum_subarray_brute(A, low=0, high=-1):
    maximum_left_index = -1
    maximum_right_index = -1
    maximum_sum = -999
    running_sum = 0

    array_length = len(A)

    # calculate all possible sums by iterating thorugh the arrary
    for i in range(array_length):
        for j in range(i, array_length):
            running_sum += A[j]
            if(running_sum > maximum_sum):
                maximum_sum = running_sum
                maximum_left_index = i
                maximum_right_index = j
        running_sum = 0

    return (maximum_left_index, maximum_right_index)


def find_maximum_crossing_subarray(A, low, mid,  high):
    running_sum = 0
    maximum_left_index = mid
    maximum_left_sum = -999

    for i in reversed(range(low, mid + 1)):  # to compensate for range(), using mid+1
        running_sum += A[i]
        if (maximum_left_sum < running_sum):
            maximum_left_sum = running_sum
            maximum_left_index = i

    running_sum = 0
    maximum_right_index = mid + 1
```

```
    maximum_right_sum = -999

    for j in range(mid+1, high + 1):  # to compensate for range() using high+1
        running_sum += A[j]
        if (maximum_right_sum < running_sum):
            maximum_right_sum = running_sum
            maximum_right_index = j

    return (maximum_left_index, maximum_right_index)


def find_maximum_subarray_recursive(A, low=0, high=-1):
    if (low == high):
        return (low, high)
    else:
        mid = (low + high) / 2

        (left_low, left_high) = find_maximum_subarray_recursive(A, low, mid)
        (right_low, right_high) = find_maximum_subarray_recursive(A, mid + 1, high)
        (cross_low, cross_high) = find_maximum_crossing_subarray(A, low, mid, high)

        left_sum = calculate_sum(STOCK_PRICE_CHANGES, left_low, left_high)
        right_sum = calculate_sum(STOCK_PRICE_CHANGES, right_low, right_high)
        cross_sum = calculate_sum(STOCK_PRICE_CHANGES, cross_low, cross_high)

        if (left_sum >= right_sum and left_sum >= cross_sum):
            return (left_low, left_high)
        elif (right_sum >= left_sum and right_sum >= cross_sum):
            return (right_low, right_high)
        else:
            return (cross_low, cross_high)


def calculate_sum(A, i, j):
    total_sum = 0
    for k in range(i, j + 1):
        total_sum += A[k]
    return total_sum


def find_maximum_subarray_iterative(A, low=0, high=-1):
    flag = 0
    highest = A[0]
```

```
    index = 0

    # if all the values in the array are negative
    for i in range(low, high + 1):
        if (A[i] > highest):
            highest = A[i]
            index = i

        if (A[i] > 0):
            flag = 1

    if (flag != 1):
        return (index, index, highest)

    running_sum = 0
    running_index = -1
    maximum_sum = 0
    maximum_left_index = -1
    maximum_right_index = -1

    #keep on adding unless temp < 0 and if temp > max_sum update max_sum. Reset to 0
otherwise
    for i in range(low, high + 1):
        temp = running_sum + A[i]
        if (temp > 0):
            if (running_sum == 0):
                running_index = i

            running_sum = temp
        else:
            running_sum = 0

        if (running_sum > maximum_sum):
            maximum_sum = running_sum
            maximum_left_index = running_index
            maximum_right_index = i

    return (maximum_left_index, maximum_right_index)


def add_matrix(A, B):
    matrix_size = len(A)
    result = [[0 for j in range(matrix_size)] for i in range(matrix_size)]
```

```
    for i in range(matrix_size):
        for j in range(matrix_size):
            result[i][j] = A[i][j] + B[i][j]
    return result


def subtract_matrix(A, B):
    matrix_size = len(A)
    result = [[0 for j in range(matrix_size)] for i in range(matrix_size)]
    for i in range(matrix_size):
        for j in range(matrix_size):
            result[i][j] = A[i][j] - B[i][j]
    return result


def square_matrix_multiply(A, B):
    A = asarray(A)
    B = asarray(B)
    assert A.shape == B.shape
    assert A.shape == A.T.shape

    matrix_size = len(A)

    result = [[0 for i in range(matrix_size)] for j in range(matrix_size)]

    for i in range(matrix_size):
        for j in range(matrix_size):
            for k in range(matrix_size):
                result[i][j] += A[i][k] * B[k][j]
    return result


def square_matrix_multiply_strassens(A, B):
    A = asarray(A)
    B = asarray(B)
    assert A.shape == B.shape
    assert A.shape == A.T.shape
    assert (len(A) & (len(A) - 1)) == 0, "A is not a power of 2"

    matrix_size = len(A)

    result = [[0 for i in range(matrix_size)] for j in range(matrix_size)]
```

```
if (matrix_size == 1):
    result[0][0] = A[0][0]*B[0][0]
    return result
else:
    matrix_new_size = (matrix_size / 2)
    A11 = [[0 for j in range(matrix_new_size)] for i in range(matrix_new_size)]
    A12 = [[0 for j in range(matrix_new_size)] for i in range(matrix_new_size)]
    A21 = [[0 for j in range(matrix_new_size)] for i in range(matrix_new_size)]
    A22 = [[0 for j in range(matrix_new_size)] for i in range(matrix_new_size)]
    B11 = [[0 for j in range(matrix_new_size)] for i in range(matrix_new_size)]
    B12 = [[0 for j in range(matrix_new_size)] for i in range(matrix_new_size)]
    B21 = [[0 for j in range(matrix_new_size)] for i in range(matrix_new_size)]
    B22 = [[0 for j in range(matrix_new_size)] for i in range(matrix_new_size)]

    for i in range(matrix_new_size):
        for j in range(matrix_new_size):
            A11[i][j] = A[i][j]
            A12[i][j] = A[i][j + matrix_new_size]
            A21[i][j] = A[i + matrix_new_size][j]
            A22[i][j] = A[i + matrix_new_size][j + matrix_new_size]
            B11[i][j] = B[i][j]
            B12[i][j] = B[i][j + matrix_new_size]
            B21[i][j] = B[i + matrix_new_size][j]
            B22[i][j] = B[i + matrix_new_size][j + matrix_new_size]

    S1 = subtract_matrix(B12, B22)
    S2 = add_matrix(A11, A12)
    S3 = add_matrix(A21, A22)
    S4 = subtract_matrix(B21, B11)
    S5 = add_matrix(A11, A22)
    S6 = add_matrix(B11, B22)
    S7 = subtract_matrix(A12, A22)
    S8 = add_matrix(B21, B22)
    S9 = subtract_matrix(A11, A21)
    S10 = add_matrix(B11, B12)

    P1 = square_matrix_multiply_strassens(A11, S1)
    P2 = square_matrix_multiply_strassens(S2, B22)
    P3 = square_matrix_multiply_strassens(S3, B11)
    P4 = square_matrix_multiply_strassens(A22, S4)
    P5 = square_matrix_multiply_strassens(S5, S6)
    P6 = square_matrix_multiply_strassens(S7, S8)
    P7 = square_matrix_multiply_strassens(S9, S10)
```

```
        C11 = add_matrix(subtract_matrix(add_matrix(P5, P4), P2), P6)
        C12 = add_matrix(P1, P2)
        C21 = add_matrix(P3, P4)
        C22 = subtract_matrix(subtract_matrix(add_matrix(P5, P1), P3), P7)

        result = [[0 for j in range(matrix_size)] for i in range(matrix_size)]

        for i in range(matrix_new_size):
            for j in range(matrix_new_size):
                result[i][j] = C11[i][j]
                result[i][j + matrix_new_size] = C12[i][j]
                result[i + matrix_new_size][j] = C21[i][j]
                result[i + matrix_new_size][j + matrix_new_size] = C22[i][j]
        return result


def test():
    """
    >>> find_maximum_subarray_brute(STOCK_PRICE_CHANGES, 0, 15)
    (7, 10)
    >>> find_maximum_subarray_iterative(STOCK_PRICE_CHANGES, 0, 15)
    (7, 10)
    >>> find_maximum_subarray_recursive(STOCK_PRICE_CHANGES, 0, 15)
    (7, 10)
    >>> square_matrix_multiply([[1, 0], [0, 1]], [[1, 0],[0, 1]])
    [[1, 0], [0, 1]]
    >>> square_matrix_multiply_strassens([[1, 0], [0, 1]], [[1, 0],[0, 1]])
    [[1, 0], [0, 1]]
    """
    pass

if __name__ == '__main__':
    import doctest
    doctest.testmod()
    test()
```