Solutions Lecture 11 (Sections 11.3-11.4)

Make sure to import Numpy to be able to complete all the exercises.

```
import numpy as np

# Display numerical values in NumPy arrays only up to three decimals,
# and suppress scientific notation
np.set_printoptions(precision=3, suppress=True)
```

Question 1

Suppose we are given a three-dimensional $m \times n \times p$ array A where $m, n, p \in \mathbb{N}$. The interpretation of these parameters is that there are m students that all did n assignments, and each assignment consisted of p questions. The elements of this array represent grades that students have obtained. For every question of every assignment, a student has received a (real-valued) grade in the interval $[1, 10]$.

The element A_{ijk} is the grade that student i obtained for question k of assignment j, where $i \in \{0, \dots, m-1\}, j \in \{0, \dots, n-1\}$ and $k \in \{0, \dots, p-1\}$.

We will write some functions to compute averages of the grades in the array A. You should not use for-loops in the questions below.

a) Write a function `average()` that takes as input the array A and outputs a one-dimensional array with on position i the average grade that student i obtained of all questions of all assignments together. You may only use the function `np.mean()` (possibly multiple times) for this.

```
def average(A):
    return np.mean(np.mean(A,axis=2),axis=1)
```

Your function should give the following output on the input below.

```
A = np.array([
[[1,1,1,1],
[2,3,2,3],
[9,5,2,4]],
[[1,1,1,1],
[2,3,2,3],
```

```
[7,7,3,8]
]])

B = average(A)
print(B)
```

```
[2.833 3.25 ]
```

b) Again answer question a), but now by using `np.mean()` at most one time by first reshaping A. Call your function `average_v2()`.

```
def average_v2(A):
    m, n, p = np.shape(A)
    B = A.reshape(m,n*p)
    return np.mean(B,axis=1)
```

Your function should give the following output on the input below.

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

B = average_v2(A)
print(B)
```

```
[2.833 3.25 ]
```

Suppose next that for every question k of assignment j, there is a weight w_{jk} determining the importance of the question.

c) Write a function `weighted_average()` that takes as input the array A and a two-dimensional array with weights $W = (w_{jk}) \in \mathbb{R}^{n \times p}$. The function first computes the weighted grade per assignment j of every student i, i.e.,

$$\frac{1}{\sum_{k=0}^{p-1} w_{jk}} \sum_{k=0}^{p-1} w_{jk} A_{ijk}$$

and afterwards the unweighted average per assignment over all students, i.e.,

$$\frac{1}{m} \sum_{i=0}^{m-1} A_{ij}.$$

The output should therefore be a one-dimensional array of size n.

```python
def weighted_average(A,W):
    total_weights = np.sum(W,axis=1)
    return np.mean(np.sum(A*W,axis=2)/total_weights,axis=0)
```

Your function should give the following output on the input below.

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

W = np.array([
[2,1,2,1],
[2,3,3,3],
[1,1,1,1]
])

B = weighted_average(A,W)
print(B)
```

```
[1.    2.545 5.625]
```

Finally, we will write a function that can round grades in $[1, 10]$ to the closest half-integral number in $\{1, 1.5, \ldots, 4.5, 5, 6, 6.5, \ldots, 9, 9.5, 10\}$. Note that the grade 5.5 is not included; every grade in the interval $[5, 6]$ has to be rounded to the closest integer (either 5 or 6).

d) Write a function `rounded_grades()` that takes as input a three-dimensional array A with elements in $[1, 10]$ and rounds these numbers according to the procedure described above. Test your function on a $2 \times 3 \times 4$ array with (real-valued) numbers randomly generated from the interval $[1, 10]$ using `np.random.uniform()`. Using NumPy random seed $s = 3$; you can round a real-valued scalar to its nearest integer value using `np.round()`.

```python
# Function
def rounded_grades(A):
    mask_outside = (A <= 5) |  (A >= 6)
    mask_inside = (A > 5) &  (A < 6)
    A[mask_outside] = np.round(2*A[mask_outside])/2
    A[mask_inside] = np.round(A[mask_inside])
    return A
```

```python
# Fix randomness
np.random.seed(3)

# Data generation
m, n, p = 2, 3, 4

A = np.random.uniform(1,10,size=(m,n,p))
```

Your function should give the output below on the specified input.

```python
print("Array A = \n",A)
```

```
Array A =
 [[[5.957 7.373 3.618 5.597]
  [9.037 9.067 2.13  2.865]
  [1.463 4.967 1.269 5.111]]

 [[6.842 3.506 7.086 6.318]
  [1.216 6.03  3.333 4.736]
  [3.552 7.238 4.964 2.412]]]
```

```python
print("Rounded grades are \n",rounded_grades(A))
```

```
Rounded grades are
 [[[6.  7.5 3.5 6. ]
  [9.  9.  2.  3. ]
  [1.5 5.  1.5 5. ]]

 [[7.  3.5 7.  6.5]
  [1.  6.  3.5 4.5]
  [3.5 7.  5.  2.5]]]
```

Question 2

In this exercise we will create functions that can count the number of integer solutions to a linear equation.

a) Write a function `sum_count` that takes as input two numbers k and n. It should return the total number of integer points $x = [x_0, \dots, x_{n-1}] \in \{0, 1, 2, \dots, k\}^n$ for which

$$\sum_{i=0}^{n-1} x_i = k.$$

You do not have to return the integer points themselves, only the number of points. Do not use for-loops. Hint: Generate arrays representing the grid $\{0, 1, \dots, k\}^n$ with `np.meshgrid()` and add them up.

```python
def sum_count(k,n):
    r = np.tile(np.arange(0,k+1),(n,1))
    grid = np.meshgrid(*r)
    sums = np.sum(grid,axis=0)
    mask = (sums == k)
    return np.sum(mask)
```

Your function should give the following output on the input below.

```python
k = 2
n = 3

# Integer points returned as tuples
print(sum_count(k,n))
```

```
6
```

b) Write a function `equation_count` that takes as input an array $a = [a_0, \dots, a_{n-1}] \in \mathbb{N}^n$ and a scalar $b \in \mathbb{N}$. It should return the total number of integer points $x = [x_0, \dots, x_{n-1}] \in \mathbb{N}^n$ for which

$$\sum_{i=0}^{n-1} a_i x_i = b.$$

You do not have to return the integer points themselves, only the number of points satisfying the equation. Do not use for-loops. Choose an appropriate grid to search over based on the array a and scalar b.

```python
def equation_count(a,b):
    # Number of variables
    n = np.size(a)

    # Boundary on the grid size;
    # largest value an individual x_i can have
    bound = np.floor(b/np.min(a))

    # Determine search grid
    r = np.tile(np.arange(0,bound+1),(n,1))
    x = np.meshgrid(*r)

    # Reshape a so that we can multiply with x
    new_shape = np.hstack((n,np.ones(n, dtype=int)))
    a = a.reshape(*new_shape)

    mask = (np.sum(x*a,axis=0) == b)
    return np.sum(mask)
```

Your function should give the following output on the input below.

```
a = np.array([1,2,2,2,2])
b = 5

# Integer points returned as tuples
print(equation_count(a,b))
```

```
15
```

Question 3

Suppose that, for given array $r = [r_0, \dots, r_{n-1}]$, we have a polynomial of the form

$$g(x_0, \dots, x_{n-1}) = \prod_{i=0}^{n-1}(x_i - r_i) = (x_0 - r_0)(x_1 - r_1)\cdots(x_{n-1} - r_{n-1}).$$

Write a function `g()` that takes as input the array r and arrays $a = [a_0, \dots, a_{n-1}]$ and $b = [b_0, \dots, b_{n-1}]$, with $a_i \leq b_i$ for all $i = 0, \dots, n-1$. It should compute the optimal value of g over an integer-valued n-dimensional box, i.e.,

$$\min\{g(x_0, \dots, x_{n-1}) : [x_0, \dots, x_{n-1}] \in B\}$$

where

$$B = \{a_0, a_0+1, \dots, b_0\} \times \{a_1, a_1+1, \dots, b_1\} \times \cdots \times \{a_{n-1}, a_{n-1}+1, \dots, b_{n-1}\} \subseteq \mathbb{N}^n.$$

You can do this by computing all function values in the integer box and computing the minimum. You may use one for-loop, but not to iterate over all possibilities x in the box. Test your function on the input below where $B = \{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}^3$.

```
def g(r,a,b):
    # Determine box points
    n = np.size(a)
    ranges = [np.arange(a[i],b[i]+1) for i in range(n)]
    x = np.meshgrid(*ranges)

    # Reshape r so that we can substract from x
    new_shape = np.hstack((n,np.ones(n, dtype=int)))
    r = r.reshape(*new_shape)

    # Determine function values
    g_values = np.prod(x-r,axis=0)

    return np.min(g_values)
```

```python
#Test the function here
r = np.array([3/2,21/8,-4/3])
a = np.array([-5,-5,-5])
b = np.array([5,5,5])

print(g(r,a,b))
```

```
-181.72916666666669
```