Lab 3: Functions and matrix algebra

Data Science for Biologists • University of Washington • BIOL 419/519 • Winter 2019

Course design and lecture material by Bingni Brunton (https://github.com/bwbrunton) and Kameron Harris (https://github.com/kharris/). Lab design and materials by Eleanor Lutz (https://github.com/eleanorlutz/), with helpful comments and suggestions from Bing and Kam.

Table of Contents

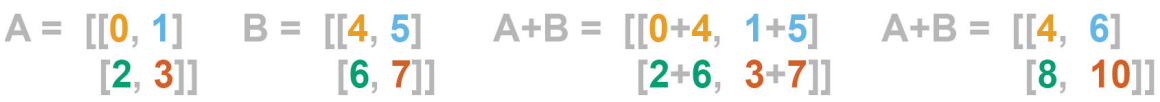
Helpful Resources

- A Primer on Matrices (https://see.stanford.edu/materials/lsoeldsee263/Additional1-notes-matrix-primer.pdf) by Stephen Boyd
- Python Data Science Handbook (http://shop.oreilly.com/product/0636920034919.do) by Jake VanderPlas
- Python Basics Cheat Sheet (https://datacamp-community-prod.s3.amazonaws.com/e30fbcd9-f595-4a9f-803d-05ca5bf84612) by Python for Data Science
- Jupyter Notebook Cheat Sheet (https://datacamp-community-prod.s3.amazonaws.com/48093c40-5303-45f4-bbf9-0c96c0133c40) by Python for Data Science
- Numpy Cheat Sheet (https://datacamp-community-prod.s3.amazonaws.com/e9f83f72-a81b-42c7-af44-4e35b48b20b7) by Python for Data Science

Lab 3 Part 1: Matrix algebra in Python

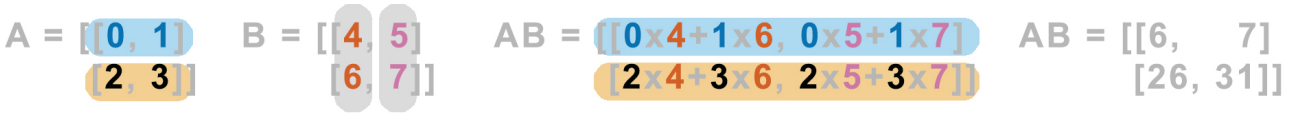
In Python we can do matrix algebra without having to calculate each value by hand. Given the numpy arrays `A` and `B`, `A+B` or `A-B` returns the elementwise addition or subtraction, and `A*B` returns the elementwise multiplication. *Elementwise* operations are performed on two matrices of the exact same shape, and produces an output matrix that also has the same dimensions. In elementwise addition, each value from matrix `A` is added to the single value in matrix `B` that is in the same position. Similarly, elementwise multiplication multiplies each value in `A` by the corresponding value in `B`.

The elementwise addition for the following two matrices `A` and `B`, (calculated in Python as `A+B`), looks like this:



A **matrix product** is different from elementwise matrix multiplication. A matrix product calculates the product of each value in the *rows* of the first matrix against the *columns* of the second matrix. Matrix products can only be calculated if the **number of columns in the first matrix equals the number of rows of the second matrix**.

The matrix product for `A` and `B` looks like this:



To calculate the matrix product for `A` and `B` in Python, we use the function `np.dot(A, B)`:

In [1]:
```
import numpy as np
```

In [2]:
```
A = np.array([[0, 1],
              [2, 3]])
B = np.array([[4, 5],
              [6, 7]])

print(np.dot(A,B))
```

```
[[ 6  7]
 [26 31]]
```

Although Python runs all of these calculations without requiring our input, we can also calculate the matrix product the long way in Python. This is probably not something you'll need to do in your research, but it's a good learning exercise to practice array indexing and matrix algebra.

To use array indexing to calculate each value in the `AB` matrix product, this is what we need to type in Python:

In [3]:
```
output = np.zeros([2, 2])
output[0, 0] = A[0, 0]*B[0, 0] + A[0, 1]*B[1, 0]
output[0, 1] = A[0, 0]*B[0, 1] + A[0, 1]*B[1, 1]
output[1, 0] = A[1, 0]*B[0, 0] + A[1, 1]*B[1, 0]
output[1, 1] = A[1, 0]*B[0, 1] + A[1, 1]*B[1, 1]
print(output)
```

```
[[  6.   7.]
 [ 26.  31.]]
```

Exercise 1: To practice indexing and matrix algebra, go through a similar process for the equation `A + B`. Your code should create a new matrix called `output` of the correct size, and then fill each value in the `output` matrix individually using indexing. Check that your answer matches the Numpy matrix addition `A+B`.

In [4]:
```
A = np.array([[0, 1],
              [2, 3]])
B = np.array([[4, 5],
              [6, 7]])

output = np.zeros([2, 2])
output[0, 0] = A[0, 0] + B[0, 0]
output[0, 1] = A[0, 1] + B[0, 1]
output[1, 0] = A[1, 0] + B[1, 0]
output[1, 1] = A[1, 1] + B[1, 1]
print(output)
```

```
[[  4.   6.]
 [  8.  10.]]
```

In [5]:
```
print(A + B)
```

```
[[ 4  6]
 [ 8 10]]
```

Lab 3 Part 2: Review of functions

Functions enclose a set of operations into a package that can be easily reused. We have actually already been using functions imported from other libraries many times throughout the quarter. For example, `np.mean(x)` is a function that returns the mean of variable `x` and `np.median(x)` is another function that returns the median. Functions can do many things, from calculating statistics to plotting figures. In the Matplotlib library, `plt.hist(x)` is a function that returns a histogram of values in variable `x`.

If you were to write your own version of the `np.sum()` function, it might look something like this:

In [6]:
```
# Define a function that sums a list of numbers.
def sum_values(values):
    sum_of_values = 0

    for value in values:
        sum_of_values = sum_of_values + value

    return sum_of_values
```

After defining the function by running the code block above, we can use our function to calculate the sum of any list of values:

In [7]:
```
print(sum_values([0, 1, 2]))
print(sum_values([-2, -5, -6]))
print(sum_values([100, 100, 100]))
print(sum_values([2]))
```

```
3
-13
300
2
```

Similarly, we can write our own function to find the length of a list, instead of using the built-in `len()` function:

In [8]:
```
# Define a function that counts the number of values in a list of number
def count_values(values):
    count_of_values = 0

    for value in values:
        count_of_values = count_of_values + 1

    return count_of_values
```

In [9]:
```
print(count_values([0, 1, 2]))
print(count_values([-2, -5, -6]))
print(count_values([100, 100, 100]))
print(count_values([2]))
```

```
3
3
3
1
```

Exercise 2: Write a new function called `my_mean` that takes a list of numbers called `values` as an input, and returns the mean. Your function should use the previous two functions, `sum_values` and `count_values`. Test your function using the code block given below. All five statements should print `True` if you have written your function correctly.

In [10]:
```
# Define a function that returns the mean of values in a list of numbers
def my_mean(values):
    sum_of_values = sum_values(values)
    count_of_values = count_values(values)
    mean = sum_of_values / count_of_values
    return mean
```

In [11]:
```
print(my_mean([1, 3, 2, 0]) == 1.5)
print(my_mean([1, 3, 2]) == 2)
print(my_mean([-3, -4, -7, -8, -9, -10]) == -7)
print(my_mean([1]) == 1)
print(my_mean([1, -1]) == 0)
```

```
True
True
False
True
True
```

Exercise 3: Write a new function called `my_median` that takes a list of numbers as an input, and returns the median of the set of values. You should not use the builtin function `np.median()`. Test your function using the code block given below. All five statements should print `True` if you have written your function correctly.

In [12]:
```
# Define a function that returns the median of values in a list of numbe
def my_median(values):
    values.sort()
    count = count_values(values)

    # If there is an odd number of values,
    # the median is the center value in the sorted list.
    if count % 2 != 0:
        # np.floor returns the largest whole number
        # that can go into the decimal.
        # Since the length / 2 of an odd number of values is
        # x.5 (1.5, 2.5, etc), taking the floor will give us
        # the index number we want (1, 2, etc).
        index = int(np.floor(count/2))
        median = values[index]

    # If there is an even number of values,
    # the median is the mean of the two center values
    # in the sorted list.
    else:
        index1 = int(count/2 - 1)
        index2 = int(count/2)

        value1 = values[index1]
        value2 = values[index2]

        median = my_mean([value1, value2])

    return median
```

In [13]:
```
print(my_median([1, 3, 2, 0]) == 1.5)
print(my_median([1, 3, 2]) == 2)
print(my_median([-3, -4, -7, -8, -10, -10]) == -7.5)
print(my_median([1]) == 1)
print(my_median([1, -1]) == 0)
```

```
True
True
True
True
True
```

Lab 3 Bonus exercises

Bonus Exercise 1: Write a function that calculates the elementwise sum of any two Numpy arrays. Your function should also check that both matrices are the same size, and print an error message if this is not the case. It may be helpful to use the code you wrote in Exercise 1 for reference.

In [14]:
```
# Define a function that calculates the elementwise sum of any two Numpy
# The shape of the two arrays must be identical.

def elementwise_sum(array1, array2):

    # In Python, assertions create error messages if the logical stateme
    # contained in the assertion is false.
    # You can specify the error message by adding it as a string after
    # the assertion (but a custom error message is not required)
    assert array1.shape == array2.shape, "Arrays must be of identical sh

    rows, columns = array1.shape
    result = np.zeros([rows, columns])

    for row in range(rows):
        for col in range(columns):
            result[row, col] = array1[row, col] + array2[row, col]

    return result
```

In [15]:
```
A = np.array([[0, 1],
              [2, 3]])
B = np.array([[4, 5],
              [6, 7]])

print(elementwise_sum(A,B))
```

```
[[  4.   6.]
 [  8.  10.]]
```

Bonus Exercise 2: Write a function that calculates the matrix product of any two Numpy arrays. Your function should also check that multiplication is possible, and print an error message if the matrices are not the correct shape.

In [16]:
```
# Define a function that calculates the matrix product of any two Numpy

def matrix_product(array1, array2):

    assert array1.shape[1] == array2.shape[0], "# col in 1st matrix must

    rows, columns = array1.shape[0], array2.shape[1]
    result = np.zeros([rows, columns])

    for row in range(rows):
        for col in range(columns):
            # Each value in the result is equal to the sum of
            # the elementwise multiplication of the row from the first m
            # and the column of the second matrix.
            result[row, col] = sum(array1[row, :] * array2[:, col])

    return result
```

In [17]:
```
A = np.array([[0, 1],
              [2, 3]])
B = np.array([[4, 5],
              [6, 7]])

print(matrix_product(A,B))
```

```
[[  6.   7.]
 [ 26.  31.]]
```

In [ ]:
```

```