Lecture 16

Nothing says autumn like some fresh apple pie.
Lecture 16

- Numpy
- Pandas
Motivation

Numerical Analysis and Computation on Large Datasets

• Python is incredibly powerful and essentially comes with functionality to do anything we could want

• However, when doing computations with a lot of numerical data, there’s an external library which provides us with even more power (faster operations and more convenient/succinct code): Numpy and PANDAS!
Numpy

Numpy (numb-pie)

• Python lists do not have the notion of addition, subtraction, multiplying, and dividing lists of numbers. We’d have to manually write these functions and iterate through every number.

• Numpy provides this functionality, and it can compute such very fast

• Numpy was designed to facilitate computing on large amounts of numerical data in python.
REAL-TIME CODING

Numpy examples
Numpy

• Numpy is an external library, so we must import it if we want to use it.

    import numpy as np

• We didn’t have to call it “np” but that’s common practice. Whatever name we gave that variable, that’s what we use to access the numpy functions
Numpy — Data Structures

• Python’s `list` data structure allows us to store a list of data. Numpy provides such too and calls their data structure an `array`.

• The basic data structures in numpy are `array` and `matrices`
You can construct a numpy array from a list:

```python
import numpy as np
weights = [121, 130, 220, 170, 148]
ages = np.array([21, 18, 19, 20, 19])
new_weights = np.array(weights)
```
numpy array’s do not support different data types in the same array; if you try to do such, numpy will guess a type and convert the entire array to that type

```python
import numpy as np

# the array below will be casted to contain strings
np.array([1, 2.5, 'hello'])

# the array below will be casted to contain floats
np.array([1, 2.5])
```
Numpy — Array

• We can do many operations with array’s:
  • `np.min()`
  • `np.max()`
  • `np.sum()`
  • `np.mean()`
  • `np.median()`
  • `np.var()`
  • `np.std()`
  • `np.sin()` or `np.cos()` or `np.tan()`
  • etc
• `np.arange()` is similar to the range function we’ve been using except it allows us to use floats for the start, stop, and step.

• and, we can see all of the values ahead of time:

```python
np.arange(10)
```

```python
np.arange(0, 1, 0.1)
```
Matrix is a two dimensional array of data, a la a list of lists, but array of arrays.

All arrays it contains must be of the same length.

Can create a matrix of all zeros via:

```python
np.zeros((num_rows, num_columns))
```
Numpy

Numpy — Matrices

- Indexing into a matrix is similar to an array, but we add now have to specify the row and column
- Now we have $m[\text{row, column}]$
- It supports the same start, stop, step syntax we used with a python list and a numpy array, just that the row and column are separated by a comma

$m[0:10, 0]$
Numpy — Matrices

- Will get the first column of data. We can shorten this with just a single colon

- Similarly, to get the first row of data, we can switch them:

  \[
  m[0, :]
  \]
• One of the cool things that numpy allows us to do is assign a value or a list to a slice of values. For example, if we wanted the first row of the matrix to be 1s we can say:

\[ m[0, :] = 1 \]
For more info on Numpy, check out their website:
https://docs.scipy.org/doc/numpy/user/quickstart.html
Lecture 16

- Numpy
- Pandas
Motivation

pandas
Motivation

• **pandas** is a data analysis library built on top of **numpy** and many other libraries.
• It extends Numpy in cool and interesting ways, and it comprises your homework
Motivation

pandas

• As with numpy, the first thing we must do is import pandas. Again we use the as syntax to shorten our commands.

• You’ll also often find examples online referring just to commands with pd or np. That implies that pandas and numpy were imported in this manner.

    import pandas as pd
The pandas Series is similar to the numpy array in that it’s a one-dimensional data array.

It even supports the Numpy functions, like mean, sum, max etc.

Creating a pandas Series is simply:

```python
series = pd.Series(np.arange(10))
```
Motivation

pandas — Series

• pandas supports giving a customized index label to each value

```python
series = pd.Series(np.arange(5), index=['zero','one','two','three','four'])
```

• Now we can refer to the value in the series either by its numerical index or its label index

```python
series[0]  or
series['zero']
```
Motivation

**pandas — Series**

- Can even make a series from a Python dictionary

```python
dict = {'b': 1, 'a': 0, 'c': 2}
series = pd.series(dict)
```
Motivation

pandas — DataFrame

• The **DataFrame** is a more extensible version of the Numpy 2-d matrix. It allows columns to be different types, which the numpy matrix didn’t.

• This makes it very convenient to read DataFrame directly from a `.csv` file!

```python
data_frame = pd.read_csv("data.csv")
```
pandas — DataFrame

- Can take a quick look at examples of the data by calling:

  ```python
  data_frame.head()
  ```

- You can get a quick summary of the data by calling:

  ```python
  data_frame.describe()
  ```
Motivation

pandas — DataFrame

• If our .csv file includes a header row of labels, then we can refer to the columns by their header label!

    `data_frame[‘happiness’]`

• To refer to specific rows, we can address a range of them:

    `data_frame[0:3]`
Motivation

pandas — DataFrame

• To get a specific row, we can use the following (which creates a very readable look at the row of data)

    `data_frame.iloc[0]`

• Can address a single data point:

    `data_frame.at[0, 'Generosity']`

• You can get the column labels by looking at

    `data_frame.columns`
Unlike numpy data structures, you can add columns to pandas data frames like you would a Python dictionary

data_frame[‘Unhappiness Score’] = 10.0 - data_frame[‘Happiness Score’]
Like a numpy array or matrix, you can use a filter syntax

```
data_frame[data_frame[‘happiness’] > 7]
```

Then if you wanted just the list of countries:

```
data_frame[data_frame[‘happiness’] > 7][‘Country’]
```
Motivation

**pandas — DataFrame**

Summarizing operations by default only occur on a single axis

```python
data_frame.mean()
```

If you want to summarize by the other axis

```python
data_frame.max(1)
```
Motivation

pandas — DataFrame

After creating a data frame, you can save it to a csv file with simply
`data_frame.to_csv('output.csv')`

pandas even supports reading and writing to excel files!

`df.to_excel('foo.xlsx', sheet_name='Sheet1')`
`pd.read_excel('foo.xlsx', 'Sheet1', index_col=None, na_values=['NA'])`
LAB TIME