ESE 402/542
Recitation 4

Intro to Python

A Few Notes Regarding CLT (not on exams)

- Using CLT with estimated variance
  - CLT says: if $X_i \sim \mathrm{dist}(\mu, \sigma^2)$ then $\frac{\sqrt{n}(\bar{X}_n - \mu)}{\sigma} \xrightarrow{d} \mathcal{N}(0,1)$
  - What if σ not known? Does CLT still work with an estimate of σ ?
  - If S_n^2 is a consistent estimator of σ^2 , then $\frac{\sqrt{n}(\bar{X}_n - \mu)}{S_n} \xrightarrow{d} \mathcal{N}(0,1)$
  - Unbiased and biased sample variance estimators are generally consistent

- Finite population sampling
  - Sampling with replacement → samples are i.i.d.
  - Sampling without replacement → no longer i.i.d.
  - Rule of thumb: if n (sample size) is approximately 10% of N, can assume i.i.d.
  - CLT still works as long as N is large relative to n

Jupyter Notebook

We'll use Jupyter Notebook to write and run Python code

- **Jupyter Notebook runs in a browser on your computer**
- **Includes interactive Python *interpreter* and *script editor***
- **To install, download Anaconda, a data science platform. This will install Python and Jupyter Notebook all at once:**
  https://www.anaconda.com/download (Download the latest version)
- **To run, open Terminal on Mac or Command Prompt on Windows and run:**
  ```
  jupyter notebook
  ```
- **Or launch from the Anaconda Navigator**

For reference: http://jupyter.org/install.html

Topics

- Basics and syntax
- Data Types
- Iterating
- Numpy Basics
- Example Problem

Topics

- Basics and syntax
- Data Types
- Iterating
- Numpy Basics
- Example Problem

Python Conventions

- `variables_and_functions`
- `CONSTANTS_LIKE_THIS`
- PEP 8 style guide

Python Objects

- All data as objects
- Do two variables refer to the same object in memory?: use `is`
- Do two variables have the same value?: use `==`

Operations

- Arithmetic: +, - *, /
- Power: **
- Modulus: %
- Comparison: <, >, <=, >=, ==, !=
- Assignment: +=, *=, /=, … (cannot use ++ and --)
- Boolean: and, or, not

Topics

- Basics and syntax
- Data Types
- Iterating
- Numpy Basics
- Example Problem

Lists

- Mutable
- Initialize empty with `list()` or `[]`
- Concatenation: `+`
- Resizable like Java ArrayList
- Check if list contains an element: `elt in my_list`
- Useful operations:
  - `l.append(new_val)`
  - `l.pop(index)`
  - `l.reverse()`

Tuples

- Immutable lists
- Use `(x, y, z)`

Strings

- Use double or single quotation
- Lots of useful methods
  - `s.join(str_list)`
  - `s.split(sep)`
  - `s.strip(sep)`

Dictionaries

- Python’s built-in hash map
- Initialize empty dictionary with `{}` or `dict()`
- `dict([(key1, val1), (key2, val2)])`
- `{key1: val1, key2: val2}`
- Access values with `d[key]`
- Assign values with `d[key] = new_value`
- Remove pair with `del`
- Views of dictionary with `d.keys()`, `d.values()`, and `d.items()`

Sets

- Hash set
- Initialize empty with `set()` or `{}`

Topics

- Basics and syntax
- Data Types
- Iterating
- Numpy Basics
- Example Problem

Iterables

- Includes lists, strings, tuples, dicts, sets and more

For loops

- No need for indexing

```
for elt in iterable:
    # do whatever
```

- Over range of integers

```
for i in range(100):
    # do whatever
```

If you need list indices

- Can usually use `enumerate()` instead of `range(len())`

```
for i, elt in enumerate(iterable):
    # do whatever
```

Iterating over two lists

- Built-in `zip()` function
- Two lists become a list of tuples

```
list_a = [3, 5, 7, 1, 20, 4]
list_b = [-4, 29, 90, 2, 1, -9]
for a, b in zip(list_a, list_b):
    print(a + b)
```

Iterating over dictionary views

- Use `d.keys()`, `d.values()`, and `d.items()`

```
for k, v in d.items():
    # do whatever

for k in d.keys():
    # do whatever

for v in d.values():
    # do whatever
```

List comprehensions

- Compact way to iterate
    - `[expr for elt in iterable]`
    - `[expr for elt in iterable if condition]`

List comprehensions

- For loop

```
example_list = [9, 2, 4, 9, -4]
res = []
for elt in example_list:
    res.append(elt * 2)
```

List comprehensions

- For loop

```
example_list = [9, 2, 4, 9, -4]
res = []
for elt in example_list:
    res.append(elt * 2)
```

- Comprehension equivalent

```
comp_res = [elt * 2 for elt in example_list]
```

Other comprehensions

- Works to create dictionaries and sets too!
- `{k: v for k, v in lst}`
- `{x for x in lst}`

Topics

- Basics and syntax
- Data Types
- Iterating
- Numpy Basics
- Example Problem

NumPy

- Efficient array representations for numerical work
- Matrix multiplication, find eigenvalues, and more

Importing modules

- At top of file

```
import numpy as np
from scipy import special, optimize
import matplotlib.pyplot as plt
```

- Can also import everything in a module with `*`

```
from module_name import *
```

NumPy arrays

- Multidimensional arrays
- `a = np.array([1, 2, 3], [4, 5, 6])`
- Useful attributes
  - ndim
  - shape
  - dtype
  - And more

Indexing and Slicing

- Use `a[index1, index2]` to selectelement
- Use `start_index:end_index` to select range (e.g. `a[0, 0:2]`)
- Can select columns with `:` (e.g. `a[0, :]`)

Vectorization

- Fast - under the hood implemented in C!
- Avoid for loops

Vectorization Example

- Want to exponentiate every element in a vector `x`

Vectorization Example

- Want to exponentiate every element in a vector `x`
- Slow

```
res = np.zeros(x.shape)
for i, elt in enumerate(x):
    res[i] = np.exp(elt)
```

Vectorization Example

- Want to exponentiate every element in a vector `x`
- Slow

```
res = np.zeros(x.shape)
for i, elt in enumerate(x):
    res[i] = np.exp(elt)
```

- Vectorized

```
np.exp(x)
```

Topics

- Basics and syntax
- Data Types
- Iterating
- Numpy Basics
- Example Problem

Practice Problem

- Without NumPy

```
def many_any(lst, k):
    ''' Returns True if at least k elements of list are > 0;
    otherwise False.
    '''
```

Practice Problem

- Without NumPy

```
def many_any(lst, k):
    ''' Returns True if at least k elements of list are > 0;
    otherwise False.
    '''
    elts = [x for x in lst if x > 0]
    return len(elts) >= k
```

Practice Problem

- With NumPy

```
def many_any_arr(arr, k):
    ''' Returns True if at least k elements of array are > 0;
    otherwise False.
    '''
```

Practice Problem

- With NumPy

```
def many_any_arr(arr, k):
    ''' Returns True if at least k elements of array are > 0;
    otherwise False.
    '''
    return (arr > 0).sum() >= k
```