Outline of the Talk

1. Part 1: Preliminaries of Python
2. Part 2: Scientific Libraries
3. Part 3: Object Oriented Programming

Numerical Computation: `numpy`

- Goal: faster array processing

Numerical Computation: `numpy`

- Goal: faster array processing
  - Example: create an array with million entries and process them – add, compute average

Numerical Computation: `numpy`

- Goal: faster array processing
  - Example: create an array with million entries and process them – add, compute average
  - Classic python must use loop which is much slower than `C` or `Fortran`
  - `numpy` does precisely that – accesses the optimized `C` or `Fortran` code by sending the *relevant portion* of the python code

Numerical Computation: `numpy`

- Goal: faster array processing
  - Example: create an array with million entries and process them – add, compute average
  - Classic python must use loop which is much slower than `C` or `Fortran`
  - `numpy` does precisely that – accesses the optimized `C` or `Fortran` code by sending the *relevant portion* of the python code
  - the relevant portions are sent in batches – therefore your code should be *vectorized* as much as possible
  - and use `numpy` functions for them

Numerical Computation: `numpy`

- Goal: faster array processing
  - Example: create an array with million entries and process them – add, compute average
  - Classic python must use loop which is much slower than `C` or `Fortran`
  - `numpy` does precisely that – accesses the optimized `C` or `Fortran` code by sending the *relevant portion* of the python code
  - the relevant portions are sent in batches – therefore your code should be *vectorized* as much as possible
  - and use `numpy` functions for them
- A ton of useful functions on arrays
  - `linspace`, `amax`, `argmax`, `ones`, `zeros`, `sum`, `mean`, `var`, `std`, `cumsum`, `cumprod` and many more
  - `random` gives a bunch of random variables – `randn`, `beta`, `binomial`, `dirichlet`, `exponential`

Scientific Tools: `scipy`

- `numpy` $\subset$ `scipy`
- built on top of `numpy` for more focused scientific programming, e.g.,
  - linear algebra
  - numerical integration
  - interpolation
  - optimization
  - distributions and random number generation
  - signal processing
- similar ideas of using python over `C` and `Fortran` subroutines
- particularly useful for statistical methods – `scipy.stats`
- two specific functions – `bisect` and `newton`

Plotting: `matplotlib`

Features:

- high quality 2D and 3D plots
- output in all the usual formats (PDF, PNG, EPS, SVG etc.)
- LaTeX integration
- fine grained control over all aspects of presentation
- animation
- and many more

Some example usage of matplotlib

Data Handling: `pandas`

- Pandas is a package of fast, efficient data analysis tools for Python
- Similar to `numpy` that defines the basic array data type and fundamental operations on arrays
- `pandas` defines fundamental *structures* for working with data, and
- endows them with *methods* that facilitate operations such as
  - reading in data
  - adjusting indices
  - working with dates and time series
  - sorting, grouping, re-ordering, slicing, and general data manipulations
  - dealing with missing value
- Two fundamental *data types*: `series` and `dataframe`
  - `series`: an array of (possibly dissimilar) objects *with generalized index* – not as space consuming as dictionaries
  - `dataframe`: a matrix with generalized index – and various methods for data handling