Vectorization

1405 Instructor: Ruiqing (Sam) Cao

Required Python Libraries for Today

Core

- NumPy, Pandas

```
import numpy as np
import pandas as pd
```

Visualization

- Matplotlib, Seaborn

```
from matplotlib import pyplot as plt
import seaborn as sns
```

Statistical learning

- scikit-learn, SciPy, statsmodels

```
import sklearn
import scipy
import statsmodel as sm
```

NumPy & Pandas Printing Format

It’s better to print only *the first few decimal digits* of large real numbers. Set the print options to keep 3 decimal digits and supress scientific notation (for NumPy arrays and Pandas DataFrames):

- **NumPy**

```
np.set_printoptions(precision=3,suppress=True)
```

- **Pandas**

```
pd.options.display.float_format = '{:.3f}'.format
```

Vectorization & Advanced Techniques

- Technically, any operation you can perform on a list can also be done on a NumPy array by *iterating through its elements*

- However, *loops are inefficient* for large data sets, and so this is where matrix algebra and vectorized operations can help

- NumPy provides built-in support for vectorized operations, making computations *faster* and the code *more concise*

Apply Functions Element-Wise to Array

- Often, you need to apply more complicated functions to each element of on a NumPy array

- Two options:

➢ Use built-in functions: **universal functions** (ufuncs)

➢ Write your own customized function and **vectorize** it

Universal Functions (ufuncs)

- Built-in functions that perform element-wise operations on a NumPy array without explicitly looping through an entire array

➢ A simple example: `np.sqrt(a)` returns a new array of the same size as `a` that contains the square root of each element of `a`

- ufuncs *are implemented in C (programming language)*
  - ➢ *Optimized for speed (e.g., avoids Python loops) and one reason NumPy is faster than plain-vanilla Python*

Commonly Used Universal Functions

ufunc Description	Python Implementation
Square root: x → sqrt(x)	`np.sqrt(a)`
Exponential: x → exp(x)	`np.exp(a)`
Natural log: x → ln(x)	`np.log(a)`
Largest integer no greater than: x→ floor(x)	`np.floor(a)`
Smallest integer no less than: x→ceil(x)	`np.ceil(a)`
Array maximum: (x1,..,xn)-> max{x1,...,xn}	`np.fmax(a1,a2,…,an)`
Array minimum: (x1,..,xn)-> max{x1,...,xn}	`np.fmin(a1,a2,…,an)`
Check for missing value	`np.isnan(a)`
Check for membership	`np.isin(a,master)`

Apply Functions Element-Wise to Array

- Often, you need to apply more complicated functions to each element of on a NumPy array

- Two options:

➢Use built-in functions: **universal functions** (`ufuncs`)

➢Write your own customized function and **vectorize** it

Vectorize a Function & Apply to Array

- `np.vectorize()` can apply *any function* element-wise to an array: it generalizes `ufuncs` but without speed improvements

<u>Step 1</u>: Define a function

```
def f(x):
       ...
return result
```

<u>Step 2</u>: Vectorize the function

```
vec_f = np.vectorize(f)
```

<u>Step 3</u>: Apply the vectorized function to an array

```
result = vec_f(a)
```