Matlab vs Python

Data Structures

Structure	Python	Matlab
Scalars	0-dimensional	1-dimensional
Collections	Lists (1-d)	Cell arrays
Key:Value maps	Dictionaries	Structures
Arrays	`numpy` arrays *	Matrices

Code organization

- In Python, code is organized in **packages** (Toolboxes in Matlab).
- A single Python file is a **module**.
- A folder of Python files with a special `__init__.py` file (which may be empty) is a **package**.
- Python programmers use **virtual environments** to isolate and keep track of the dependencies.

Syntax (Python)

```python
import numpy as np
import matplotlib.pyplot as plt

fs = [1, 2, 4]
all_time = np.linspace(0, 2, 200)
t = all_time[:100]

for f in fs:
    y = np.sin(2 * np.pi * f * t)
    plt.plot(t, y, label='{} Hz'.format(f))

plt.legend()
plt.savefig('basics_python.pdf')
```

Syntax (Matlab)

```matlab



fs = [1 2 4];
allTime = linspace(0, 2, 200);
t = allTime(1:100);
hold('on')
for f = fs
      y = sin(2 * pi * f * t);
      plot(t, y, 'DisplayName', sprintf('%d Hz', f));
end
legend('show')
saveas(gcf, 'basics_matlab.pdf');
```

Any differences?

Some differences

- Matlab does not need imports, as long as the file is on the right path.
- No `;` needed in Python (usually).
- Items are comma-separated in Python.
- Different use of `[` and `(`.
- Indentation is needed for the `for` loop in Python, but no `end` keyword.
- Keyword arguments in Python.

Slicing

- In Python, it starts at zero:

```python
a = [1,2,3,4,5,6,7]
low, high = 2, 4
a == a[:low] + a[low:high] + a[high:]
```

- In Matlab, at one:

```matlab
a = [1 2 3 4 5 6 7];
low = 2;
high = 4;
all(a == [a(1:low), a(low+1:high), a(high+1:end)])
```

Arrays

- Let A be a 2D array with r rows and c columns.
    - Matlab: `shape(A) => (r,c)`
    - Numpy: `A.shape => (r,c)`
- Operations are always **element-wise** in Python (no need for `.*`, `./` as in Matlab).
- For matrix product in Python:
    - `@`, e.g. `A@A`. (Python 3+)
    - `np.dot(A,A)` or `np.matmul(A,A)`

Arrays (cont.)

- Memory storage of arrays is different: Numpy uses *row-major* order, Matlab *column-major*.
- **Example:** If $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ then:

Python	Matlab
A[0,0] = a	A(1,1) = a
A[0,1] = b	A(2,1) = c
A[1,0] = c	A(1,2) = b
A[1,1] = d	A(2,2) = d

Why do we care?

- Memory layout can have significant impact on performance.
- We do not care if we vectorize our code to avoid loops because this is taken care of under the hood.
- When we *do* have to loop in Python, start on the inner-most dimension (over 10x improvement).

OOP

- Code can be organized in two important (not the only) categories:
  - **Procedural**: code is organized in functions.
  - **Object-oriented**: Data and code are coupled together.
- Python and Matlab support both, with Python leaning more on the OOP side.

Example

- Same code,different paradigms:

Object-oriented	Procedural
`a = np.arange(6)`	`a = np.arange(6)`
`a.max(axis=0)`	`np.max(a, axis=0)`

- Chaining methods vs function nesting:

Python	Matlab
`txt = "Python is fun!!  "`	`txt = "Python is fun!!  ";`
`txt.strip().upper()`	`upper(strip(sentence))`

References

- Matlab to Python migration guide
- Numpy for Matlab Users