Integrating Python and MATLAB

Kevin Williamson

November 28, 2017

Outline

1. Python
2. Getting Started
3. Python types in MATLAB
4. Using python modules

What is python?

- Python is a high-level programming language developed by Guido van Rossum in 1991
- It is an interpreted language, allowing for easy testing of code
- Can also be used as command-line scripts
- Name was inspired by Monty Python–documentation contains lots of Monty Python references (e.g., “spam”)
- Two flavors: python 2 and python 3
    - python 3 provides some optimizations and new features over python 2
    - not backward compatible: code written in python 2 will not often run in python 3

Why use python?

- Comes with “batteries included”: extensive standard library
- Lots of external libraries:
  - numeric libraries: numpy, scipy
  - Netgen/NGSolve
  - PyTrilinos
- Easily extensible
  - Most common implementation is CPython which is written in C and provides an API for writing C/C++ extension modules (with a slight learning curve)
  - tools like SWIG exist for easily wrapping C/C++ code
- Open source
- Portable (mostly)
- Well-documented:
  - python 2: `https://docs.python.org/2/`
  - python 3: `https://docs.python.org/3/`

Why use python in MATLAB?

- MATLAB provides some support for integrating with python
- Integrating MATLAB and python allows one to combine MATLAB code with the power of python

Using the right python verion

- MATLAB supports python 2.7, 3.4, 3.5, 3.6
- The command **pyversion** in MATLAB tells you which version of python is used
- To change version:
  - On Windows: **pyversion(VERSION)**
  - On Linux/Mac: **pyversion(EXECUTABLE)**

Opening up python interpreter

- The command **!python** opens up a python 2 interpreter
- The command **!python3** opens up a python 3 interpreter
- Leave either interpreter with **exit()**
- Can execute python commands within the interpreter, but they do not interact with MATLAB

Calling external python scripts

- Use **!python foo.py** (for python 2) or **!python3 foo.py** (for python 3) to execute the script 'foo.py'
- Can also use **system()** function: **system('python3 foo.py')**
- Still no direct interaction with MATLAB–will need to read/write data from disk
- Possible use case:
  1. MATLAB writes data to disk in a format that python script can read
  2. python script reads in data, processes, and writes new data for MATLAB to read

Working with python functions and types in MATLAB

- We can call python functions directly in MATLAB by prepending **py.**
- We can construct basic python data types and use directly in MATLAB:
  - list: **mylist = py.list({'Bob', 'Sue'});**
  - tuple: **mytuple = py.tuple({'Bob', 'Sue'});**
  - dictionary: **mydict = py.dict(pyargs('Bob',29,'Sue',25);**

Constructing python lists

- A python *list* is a mutable type in python for storing multiple objects of arbitrary type
- Python lists are similar to MATLAB cells
- Construct lists in python by enclosing data in brackets:
  - **list123 = [1,2,3]**
- Constructing lists in MATLAB (two ways):
  - **list123 = py.list([1, 2, 3]);**
  - **list123 = py.list({1, 2, 3});**
- Both produce the same lists, but what if we want a list of strings? These are different:
  - **liststr1 = py.list(['foo','bar']);**
  - **liststr2 = py.list({'foo', 'bar'});**
- Recommend always using { and } in MATLAB
- Can directly convert between python lists and MATLAB cells

Accessing list elements

- In python, we access elements of lists using 0-up indices in brackets. We can also access a *slice* of the list using a:b.
    - **list123[0]**
    - **list123[2]**
    - **list123[1:]**
- In MATLAB, we use 1-up indices. We use braces { and } to access individual elements and parentheses (and) for slices
    - **list123{1}**
    - **list123{2}**
    - **list123(2:end)**

Calling list member functions

- In python, calling **dir(listname)** will return all of the members of the list named **listname** (in MATLAB, use **py.dir(listname)**)
- There are various member functions, including **append()** and **index()**
- We may call these functions using dot notation:
    - **list123.append(7)**
    - **list123.index(2)**
- Syntax is the same for both python in MATLAB–since **list123** is a python type, we do not need to prepend **py.**
- In MATLAB, we need only be careful that we are passing the correct types (arguments that are MATLAB types may need to be converted to corresponding python types before calling the function)

More about lists

- Lists are *mutable*, i.e., we can change the elements:
  - python: **list123[1] = 7**
  - MATLAB: **list123{2} = 7;**
- In python, we can construct lists using list comprehension, but I am unsure how to do this in MATLAB:
  - python only: **mylist = [x**2 for x in range(1,10)]**

Tuples

- Python *tuples* are like lists except they are *immutable*, i.e., they cannot be changed
- Construction:
  - python: **tuple123 = (1,2,3)**
  - MATLAB: **tuple123 = py.tuple({1,2,3});**
- Access elements and slices in the same way as lists:
  - python (element): **tuple123[0]**
  - python (slice): **tuple123[:2]**
  - MATLAB (element): **tuple123{1}**
  - MATLAB (slice): **tuple123(1:2)**
- Access member functions directly:
  - both: **tuple123.index(2)**

Dictionaries

- A python *dictionary* is a data structure containing key-value pairs, with keys and values being arbitrary types. Each key is associated with a unique value.
- Construction:
  - python: **mydict = {'key1':'val1', 'key2':'val2'}**
  - MATLAB: **mydict = py.dict(py.args('key1','val1','key2','val2'));**
- In MATLAB, use **pyargs** function to alternate between keys and values, but this requires keys to be strings
- If all keys are strings, can directly convert between python dictionaries and MATLAB structs:
  - **d = py.dict(s);**
  - **s = struct(d);**

More on dictionaries

- Access elements in python using [and]; access elements in MATLAB using braces { and }:
    - python: **mydict['key1']**
    - MATLAB: **mydict{'key1'}**
- Can add new key-value pairs at any time:
    - python: **mydict['key3'] = 'val3'**
    - MATLAB: **mydict{'key3'} = 'val3'**
- Dictionaries are mutable, so we can modify any key-value pair:
    - python: **mydict['key1'] = 'newval1'**
    - MATLAB: **mydict{'key1'} = 'newval1'**

pyargs

- Some python functions take keyword-value arguments
- Suppose there is a function **foo** that has a keyword argument named **bar**. Then this function can be called as follows:
  - python: **x = foo(bar=7)**
  - MATLAB: **x = py.foo(pyargs('bar',7));**
- **pyargs** can also be used to construct dictionaries, but can only use keys that are strings

Python modules

- A python module is a collection of various data types and functions
- To use a module in python, we need to import it:
  1. **import glob**
  2. **import numpy as np**
  3. **from os import path**
  4. **from sys import ***
- For first import statement, we can use anything within the **glob** module by prefixing with **glob.**
- Second import statement is similar to first, except to use anything from **numpy** we now prefix with **np.**
- Third import statement only imports the **path** submodule of **os**. Use **path.** as prefix
- Fourth import statement imports everything from **sys** and no prefix is needed

Using python modules in MATLAB

- In MATLAB, use **py.module.name** to access member **name** from the module **module**. For submodules, use full module name
    - **filelist = py.glob.glob('*');**
    - **py.os.path.sep**
- No import statements

Benefits of modules

- Organizes code, allowing for simplified scripting and code reuse
- Since python modules are pure python, can run python code that can't be directly run in MATLAB
  1. Convert MATLAB data to python objects
  2. Call python module function to operate on python objects
  3. Return python objects that MATLAB can use
  4. Convert new objects to MATLAB data

Creating and using a module

- To create a module named “spam”, place all of the desired functions in a .py file named “spam.py”
- Within python, import with **import spam**, and prefix all calls with **spam.**
- Within MATLAB, prefix all calls with **py.spam.**