10. Using C++ in Python

Ján Dugáček

November 21, 2018

Table of Contents

1. Using SWIG
   - Motivation
   - Some class
   - Configuration file for SWIG
   - Using SWIG
   - Using distutils to have Python compile it for itself
   - Usage
   - Exercise
   - More info
2. Numpy
   - A note about C arrays
   - A function
   - SWIG file
   - Result
   - Exercise
   - More info
3. Homework

Motivation

- You will run into a lot of code written in Python in your life
- You might need to integrate C++ into it because cycles are not exactly fast in Python
- Using C++ is the only way to add new builtin types into Python

Some class

```
#ifndef PSEUDOFUNC_H
#define PSEUDOFUNC_H
#include <map>
class pseudofunc {
        std::map<float, float> data;
public:
        void add(float x, float fx);
        float at(float x);
};
#endif
```

- The parts with `#ifndef` and others are preventing the file from being included more than once, all you have to do is to give it a unique name
- Name is supposed to be `pseudofunc.h`
- Declarations of methods will be in another file

Some class #2

```
#include "pseudofunc.h"
void pseudofunc::add(float x, float fx) {
        data[x] = fx;
}
float pseudofunc::at(float x) {
  if (data.empty()) return 0;
  if (x<data.begin()->first) return data.begin()->second;
  if (x>data.rbegin()->first) return data.rbegin()->second;
  auto more = data.upper_bound(x);
  auto less = std::prev(more);
  float part=(x - less->first)/(more->first - less->first);
  return part * more->second + (1 - part) * less->second;
}
```

- Now we define the methods, in a file that should be named `pseudofunc.cpp`

Configuration file for SWIG

```
%module pseudofunc
%{
#include "pseudofunc.h"
%}
%include "pseudofunc.h"
```

- It tells SWIG how to compile the file, it allows a lot of tricks
- Make sure you don't mistake the % for the #
- If using some other python-bound types, `%include` them as `std_string.i`
- These slides assume this file is named `pseudofunc.i`

Using SWIG

```
swig -c++ -python pseudofunc.i
```

- You need to have SWIG installed to do this in the command line
- You have to specify the code is in C++ (and not in C) and that you want to generate python code (and not for other programming languages)

Using distutils to have Python compile it for itself

```
#!/usr/bin/env python3
from distutils.core import setup, Extension
pseudofunc_module = Extension('_pseudofunc',
       sources=['pseudofunc_wrap.cxx', 'pseudofunc.cpp'], )
setup (name = 'pseudofunc', version = '1', author = "Dugi",
       description = """A pseudofunction""",
       ext_modules = [pseudofunc_module],
       py_modules = ["pseudofunc"], )
```

- If it's called `makePseudofunc.py`, call `python3 makePseudofunc.py build` in command line

Usage

```
>>> import pseudofunc
>>> p = pseudofunc.pseudofunc()
>>> p.add(3,15)
>>> p.add(4,20)
>>> p.add(5,30)
>>> p.at(4.1)
21.0
```

- If you have the compiled file and the generated python file available (for example in the same folder), you can import it and use the class in Python

Exercise

1. Use C++ to create a `mumpy` Python module that will tell a random `yer mum` joke (out of at least 3)
2. Use C++ to create a simulacrum of `std::vector<double>` in Pyton (or `easy::vector<double>`
3. Use C++ to create some sort of 2D vector in Python
4. Use C++ to create a function in Python that computes the derivative of data in a form that fits you

More info

- A short guide to using SWIG is at `https://www.cs.ubc.ca/~gberseth/blog/using-swig-to-wrap-c-for-python.html`
- A very detailed description of this functionality is at `http://www.swig.org/Doc1.3/Python.html`

A note about C arrays

```
void arrayDemoC(int* array) {
        int a = *array;
        int b = array[0];
        int c = array[1];
}
```

- Arrays in C are actually naked pointers to the first element in an array
- Naked pointers, the only kind of pointers in C, are thus used in the same way as arrays
- The array's size must be in another variable

A function

```
#include "unnormalise.h"
void unnormalise(double* in, int inS, double* out, int outS)
{
        if (inS == 0 || inS != outS) return;
        double min = in[0];
        for (int i = 0; i < inS; i++)
                if (in[i] < min) min = in[i];
        min = 1 / min;
        for (int i = 0; i < inS; i++)
                out[i] = in[i] * min;
}
```

- Numpy arrays are accessed as 2 variables, data and size
- Can you guess what this does?

A function #2

```
#ifndef UNNORMALISE_H
#define UUNORMALISE_H
void unnormalise(double* in, int inS, double* out, int outS);
#endif
```

- We make it a header

SWIG file

```
%module unnormalise
%{
#define SWIG_FILE_WITH_INIT
#include "unnormalise.h"
%}
%include "numpy.i"
%init %{
    import_array();
%}
%apply (double* IN_ARRAY1, int DIM1) {(double* in, int inS)}
%apply (double* INPLACE_ARRAY1, int DIM1) {(double* out, int
%include "unnormalise.h"
```

- We need to enable the swig initialisation function `import_array()` with `SWIG_FILE_WITH_INIT`
- We need to map the input types to the argument names we are using

Result

```
>>> import unnormalise
>>> result = numpy.empty_like(x)
>>> unnormalise.unnormalise(x, y)
```

- It's compiled like before
- It's not possible to return an array (numpy's SWIG interface is not great)
  - It's faster without returning, as new arrays aren't created
  - You can add a Python function that creates the result array and returns it
- To bind more values, use:

```
%apply (double* IN_ARRAY1, int DIM1) {(double* in1,
                int inS1), (double* in2, int inS2)}
```

Exercise

1. Write a C++ function that can find the median of a numpy array
2. Write a C++ function that can find the highest common denominator of all numbers in a numpy array
3. Write a C++ function that interpolates a two-variable function given by a 2D numpy array (assuming the indexes are x and y)
4. Write a C++ function that computes the divergence of a function given by a 3D numpy array
5. Write a C++ function that computes the curl of a function given by a 3D numpy array

```
%apply (double* IN_ARRAY2, int DIM1, int DIM2)
                {(double* in, int in1, int in2)}
```

Exercise for those who don't know numpy

1. Use C++ to add a function to Python that will sort a vector so that the highest number is the first
2. Use C++ to add a function to Python that prints a smiley of a given size made of spaces and some letter
3. Use C++ to create a Python class `vector3D` that supports addition and vector product
4. Use C++ to create a Python function to compute the mean root-mean-square deviation of given data
5. Use C++ to create a Python function to compute the covariance of two sets of given data

More info

- A description of using numpy with C++ is at `https://www.scipy-lectures.org/advanced/interfacing_with_c/interfacing_with_c.html#swig`
- In-depth information can be found here `https://docs.scipy.org/doc/numpy/reference/swig.interface-file.html`

Homework

- Use C++ to create a Python class that represents analytical functions composed of addition, subtraction, multiplication, division, power and logarithm
- You have two weeks to do it
- It's recommended to compose it in a tree structure, starting with function $f(x) = 1$ and function $f(x) = x$ and using operators to compose it, then `operator()` to call it
- For the ambitious: Give it a method to compute its derivative (still in analytical form)
- For lunatics: Give it a method to compute its primitive function (still in analytical form)