Using boost::python and Eigen to replace Python code with C++

Johannes Nix

Talk at UK Astronomy Techology Centre, June 2018

Replacing Numerical Python code with C++

- We have complex algorithms, developed in Numerical Python
- Our deliverables still need to be written in C++
- Comprehensive tests have been developed in Python
- We want to use this test code
- **Goal: an efficient way to convert Python to C++, and test the result**

Replacing Numerical Python code with C++

- We have complex algorithms, developed in Numerical Python
- Our deliverables still need to be written in C++
- Comprehensive tests have been developed in Python
- We want to use this test code
- **Goal: an efficient way to convert Python to C++, and test the result**

Replacing Numerical Python code with C++

- We have complex algorithms, developed in Numerical Python
- Our deliverables still need to be written in C++
- Comprehensive tests have been developed in Python
- We want to use this test code
- **Goal: an efficient way to convert Python to C++, and test the result**

Replacing Python code with C++: Overview

- Translate code to C/C++
- Use the Eigen library to translate Numpy code
- Wrap C++ classes into Python extension modules, using boost::python
- Test results using the existing Python tests

Translating Numpy code using Eigen

- Numerical Python: Vectors and arrays as primary objects
- Makes possible to write concise numerical code
- Also, a large range of vector operations, linear algebra, etc
- The Eigen library[1] matches much of Numpy's capabilities
- It is a advancement of Todd Veldhuizen's blitz++ library, which introduced template expressions to write compact array code in C++[2]

[1] http://eigen.tuxfamily.org/

[2] http://blitz.sourceforge.net/resources/blitz-0.9.pdf

Translating Numpy code using Eigen

- Numerical Python: Vectors and arrays as primary objects
- Makes possible to write concise numerical code
- Also, a large range of vector operations, linear algebra, etc
- The Eigen library[1] matches much of Numpy's capabilities
- It is a advancement of Todd Veldhuizen's blitz++ library, which introduced template expressions to write compact array code in C++[2]

[1] http://eigen.tuxfamily.org/

[2] http://blitz.sourceforge.net/resources/blitz-0.9.pdf

Example: Computing a surface normal vector

- we have three points defining a plane: x_0, x_1, x_2
- we want to compute the surface normal of the plane

$$\begin{aligned} \vec{n_u} &= (\vec{x_1} - \vec{x_0}) \times (\vec{x_2} - \vec{x_0}) \\ \vec{n} &= \frac{1}{||\vec{n_u}||} \vec{n_u} \end{aligned}$$

Example : Computing a surface normal vector in Numpy

```
1  from numpy.linalg import norm
2
3  def getNormalPlane(x0, x1, x2):
4      a = x1 - x0
5      b = x2 - x0
6      axb = cross(a,b)
7
8      return axb / norm(axb);
```

Example : C++ function computing a surface normal using Eigen

```
1   Vector3d getNormalPlane(const Vector3d &x0,
2                           const Vector3d &x1,
3                           const Vector3d &x2)
4   {
5       Vector3d a = x1 - x0;
6       Vector3d b = x2 - x0;
7       Vector3d axb = a.cross(b);
8
9       double norm = axb.norm();
10
11      Vector3d n = axb * (1.0 / norm);
12      return n;
13  }
```

Example : Eliminating intermediate objects

```
1   Vector3d getNormalPlane(const Vector3d &x0,
2                           const Vector3d &x1,
3                           const Vector3d &x2)
4   {
5       Vector3d axb = (x1 - x0).cross(x2 - x0);
6       return axb / axb.norm();
7   }
```

Example : C++ Error handling

```
1   Vector3d getNormalPlane(const Vector3d &x0,
2                           const Vector3d &x1,
3                           const Vector3d &x2)
4   {
5       Vector3d axb = (x1 - x0).cross(x2 - x0);
6
7       double norm = axb.norm();
8       if (norm <= 0) {
9         throw UndefinedPlaneException(
10              "the three points are on a line");
11      }
12      return axb / norm;
13  }
```

Wrapping C++ classes as Python extension types

The boost::python library automates the tedious task of exposing C/C++ extension modules to Python.

- In detail, quite complex, but works well
- Error messages sometimes intimidating, but harmless
- One certainly does not need to understand all the details

Example : Python class

```
1  class SimpleClass:
2      def init(self, offset):
3          self.offset = offset
4
5      def addOffset(x):
6          return x + self.offset
```

Example : Equivalent C++ class

```
1  class SimpleClass {
2  private:
3
4  public:
5
6      double myoffset;
7
8      SimpleClass(double offset);
9
10     double addOffset(double x);
11
12     double getOffset();
13 };
```

Example : boost::python wrapper

```
1  BOOST_PYTHON_MODULE(boost_simple)
2  {
3      using namespace boost::python;
4
5      class_<SimpleClass>("SimpleClass", init<double>())
6      .def("addOffset", &SimpleClass::addOffset)
7      .def("getOffset", &SimpleClass::getOffset)
8      .def_readwrite("offset", &SimpleClass::_offset)
9      ;
10 }
```

Example 2: Python class processing a list

```
1  class Statistics:
2
3      getMedian(my_numbers):
4          my_numbers = list(my_numbers)
5          my_numbers.sort()
6          m = len(my_numbers)
7          return my_numbers[m/2]
8
9  };
```

C++ class processing a python list

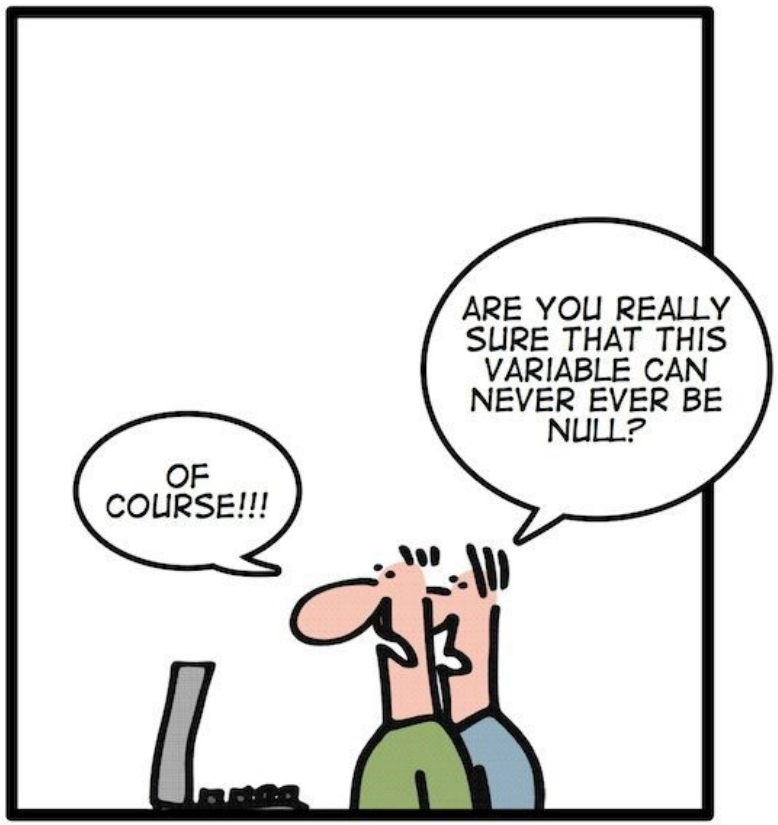
```
1  class SimpleStatistics {
2  public:
3      SimpleStatistics();
4
5      double getMedian(boost::list &my_numbers) {
6          std::vector<double> double_numbers;
7
8          const int len_list = len(my_numbers);
9
10         for (int i= 0; i < len_list; i++)
11         {
12             double next_num = boost::extract<double>(my_numbers[i]);
13             double_numbers.push_back(next_num);
14
15         }
16         std::sort(double_numbers.begin(), double_numbers.end());
17
18         return double_numbers[len_list/2];
19     }
20 };
```

Testing the list example...

Did you spot the bug?

We need to test for a list length of zero!

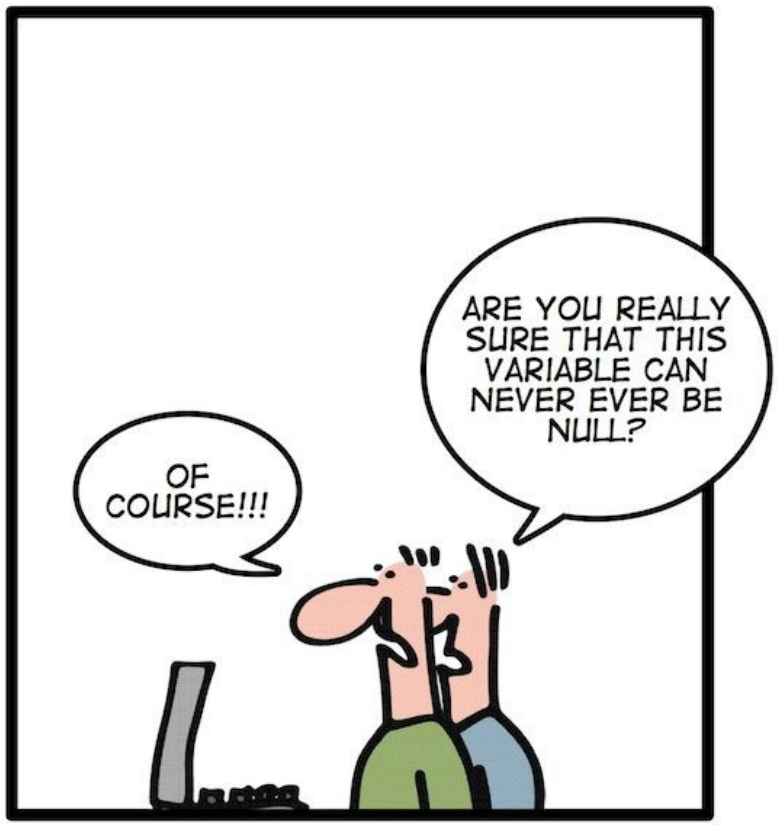
Testing the list example...



Did you spot the bug?

We need to test for a list length of zero!

Testing the list example...



Did you spot the bug?

We need to test for a list length of zero!

Wrapping functions with Eigen array parameters

- Classes with methods which have Eigen vectors as arguments need to be wrapped in a derived class
- Derived class accepts Numpy arrays and...
- translates the Numpy arrays into Eigen array objects.

Just a short glance at it...
(For details, see demo and code sample package...)

Wrapping functions with Eigen array parameters

- Classes with methods which have Eigen vectors as arguments need to be wrapped in a derived class
- Derived class accepts Numpy arrays and...
- translates the Numpy arrays into Eigen array objects.

Just a short glance at it...
(For details, see demo and code sample package...)

Detailed Example : wrapping Eigen arrays

```
1  void py_getNormalPlane(PyObject * x0, PyObject * x1, PyObject * x2,
2                         PyObject * n)
3  {
4
5    /* check for right shape of input arrays
6       (not shown)*/
7
8      // convert input argument
9      Map<Vector3d> vec_x0 ((double*) PyArray_DATA(x0), 3);
10     Map<Vector3d> vec_x1 ((double*) PyArray_DATA(x1), 3);
11     Map<Vector3d> vec_x2 ((double*) PyArray_DATA(x2), 3);
12     // convert output argument
13     Map<Vector3d> vec_n ((double*) PyArray_DATA(n), 3);
14     // call implementation using Eigen arrays
15     vec_n = getNormalPlane(vec_x0, vec_x1, vec_x2);
16 }
```

How does this look? / 1

```
jnix@nippes:~/MOONS/boost-sample$ make
"g++" -shared -I -std=c++11 -Wall -Wextra -pedantic  -fPIC -DDEBUG -g
"g++" -shared -I -std=c++11 -Wall -Wextra -pedantic  -fPIC -DDEBUG -g
jnix@nippes:~/MOONS/boost-sample$ python
Python 2.7.13 (default, Nov 24 2017, 17:33:09)
[GCC 6.3.0 20170516] on linux2
Type "help", "copyright", "credits" or "license" for more information
>>> import boost_simple
>>> s = boost_simple.SimpleClass(3)
>>> s.addOffset(10)
13.0
>>> s.offset
3.0
```

How does this look? / 2

```
jnix@nippes:~/MOONS/boost-sample$ python
Python 2.7.13 (default, Nov 24 2017, 17:33:09)
Type "help", "copyright", "credits" or "license" for more information
>>> import boost_eigen
>>> be = boost_eigen.SimpleGeometry()
>>> from numpy import *
>>> x0 = array([0., 0., 0.])
>>> x1 = array([1., 0., 0.])
>>> x2 = array([0., 1., 0.])
>>> result = zeros(3,dtype=float)
>>> be.getNormalPlane(x0,x1,x2,result)
>>> result
array([ 0.,  0.,  1.])
>>> be.getNormalPlane(x0,x1,x1,result)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  boost_eigen.UndefinedPlaneError:  the three points
  are on a line, and do not define a plane
>>>
```

Testing

Use Python unit tests as usual. . .

Summary

- The Eigen library allows to replace Numerical Python code with C++
- boost::python makes C++ code accessible as Python extension modules
- Most testing can still be done in Python

Example code

The example code shown can be copied or cloned from

`file://dalriada.roe.ac.uk/home/jnix/MOONS/boost-sample/`

It contains:

- some more examples
- a recipe for accessing Numpy arrays as Eigen objects
- a template for exception handling.

Thanks for the attention.

References

1. http://eigen.tuxfamily.org
2. http://www.lassp.cornell.edu/sethna/DM/Documentation/numpy.pdf, chapter 12 and 13
3. https://docs.python.org/3/extending/
4. https://www.boost.org/doc/libs/1_67_0/libs/python/doc/html/tutorial
5. "Python Scripting for Computational Science", Hans-Petter Langtangen, Chapter 10, Springer Berlin Heidelberg, 2006, ISBN-13 978-3-540-29415-3
6. http://blitz.sourceforge.net/resources/blitz-0.9.pdf