How-to wrap hermes classes

Vladimir Cerny

October 21, 2011

Directory structure

Directory structure is the same as structure of hermes.

- hermes_common
  - include
    declaration files *.pxd. Files correspods with *.h files in hermes.
  - src
    definition files *.pxi. Files corresponds with *.cpp files in hermes.
- hermes2d
  - include
  - src

All files are included in corresponding setup.py and in hermes_common.pxd/hermes2d.pxd (pxd files) or hermes_common.pyx/hermes2d.pyx (pxi files)

Declaration files

Declaration file can be created from header file. This is example and it doesn't correspond to the real matrix.pxd file.

```
cdef extern from "matrix.h" namespace "Hermes::Algebra":
  enum EMatrixDumpFormat:
    DF_MATLAB_SPARSE
    DF_PLAIN_ASCII

  cdef cppclass Matrix[Scalar]: #abstract
    Scalar get(unsigned int m, unsigned int n)
    bool dump(FILE *file, char *var_name, EMatrixDumpFormat fmt)
    bool dump(FILE *file, char *var_name)

  cdef cppclass SparseMatrix[Scalar]:# public Matrix<Scalar>
    void prealloc(unsigned int n)

cdef class PyMatrixReal:
  cdef Matrix[double] * thisptr

cdef class PySparseMatrixReal
```

Line `cdef extern from "matrix.h" namespace "Hermes::Algebra":` contains link to header and namespace.

`enum EMatrixDumpFormat:` is enum definition

`cdef cppclass Matrix[Scalar]: #abstract` is templated class. I use "abstract" comment to indicate this class is abstract. There are no need to have constructor in abstract class.

`bool dump(FILE *file, char *var_name, EMatrixDumpFormat fmt)` is declaration of function. Function declaration in cython can't have default values so `bool dump(FILE *file, char *var_name)` is used as overloaded function. original function looks like this: `bool dump(FILE *file, const char *var_name, EMatrixDumpFormat fmt = DF_MATLAB_SPARSE)`

`cdef cppclass SparseMatrix[Scalar]:# public Matrix<Scalar>` after declaration of subclass I add comment of parent class as a reminder. It is not needed to wrap methods of subclasses inerhided from parent class.

`cdef class PyMatrixReal:` is declaration of pyhon class. Prefix Py is used in all python classes in hermes. Suffix Real/Complex is used for templated classes.

`cdef Matrix[double] * thisptr` is pointer to C++ class as a member of python class this pointer is not accesible from python. It exists only in class which is not subclass of other wrapped class.

Definition files

Definition files *.pxi (*.pxi are included files *.pyx is main file of module) contain python methods implemented in cython.

```
class PyEMatrixDumpFormat:
  DF_MATLAB_SPARSE, DF_PLAIN_ASCII, DF_HERMES_BIN, DF_NATIVE, DF_MATRIX_MARKET=range(5)

cdef class PyMatrixReal: #abstract
  def __dealloc__(self):
    del self.thisptr

  def get(self, unsigned int m, unsigned int n):
    self.thisptr.get(m, n)

  def dump(self, file, char *var_name, fmt=None):
    cdef FILE * f = PyFile_AsFile(file)
    if fmt:
      return self.thisptr.dump(f, var_name,fmt)
    else:
      return self.thisptr.dump(f, var_name)
```

Python does not have enums. So I use normal classes with static members. PyEMatrixDumpFormat.DF_MATLAB_SPARSE can be used in python and it returns number 0.

`__dealloc__` is method for memory freeing. It is neede only in class which is not subclass of other wrapped class. And it will allways look like this

Conversion from python objetcts to C++ types is done for calling C++ functions. Some types (numbers, char* to python string) is conversed automaticaly. Some need to be converted manualy. Function overloading need to be done manualy.

Constructors

Nonabstract methods have constructors in method `__cinit__`. Type of self is checked because `__cinit__` is called for all parent classes but constructor need to be called only one.

```
cdef class PyTimePeriod:
  def __cinit__(self,char * name=NULL):
    if (type(self)!=PyTimePeriod):
      return
    self.thisptr=new TimePeriod(name)
```

Type conversions

`std::complex<double>` is included as `cComplex[double]` and can be created from python complex `c=cComplex[double](p.real,p.imag)`. Analogicaly in the oposite direction `p=complex(c.real(),c.imag())`

`PyFile_AsFile(file)` converts python object created by python `open("path","mode")` to C type `FILE*`.

`std::string` can be converted from/to python string throught `char *`. `cdef string s.assing(pythonstr)` and `pythonstr=s.c_str()`

In place of `Hermes::vector` I use python list.

```
cdef vector[Mesh*] v
  cdef PyMesh m
    for m in mesh:
      v.push_back(m.thisptr)
```

In hermes_common/utils.px[d/i] are some methods for conversion python lists to C number arrays.