Interfaçage C/C++ avec cython

Xavier Juvigny

ONERA/HPC

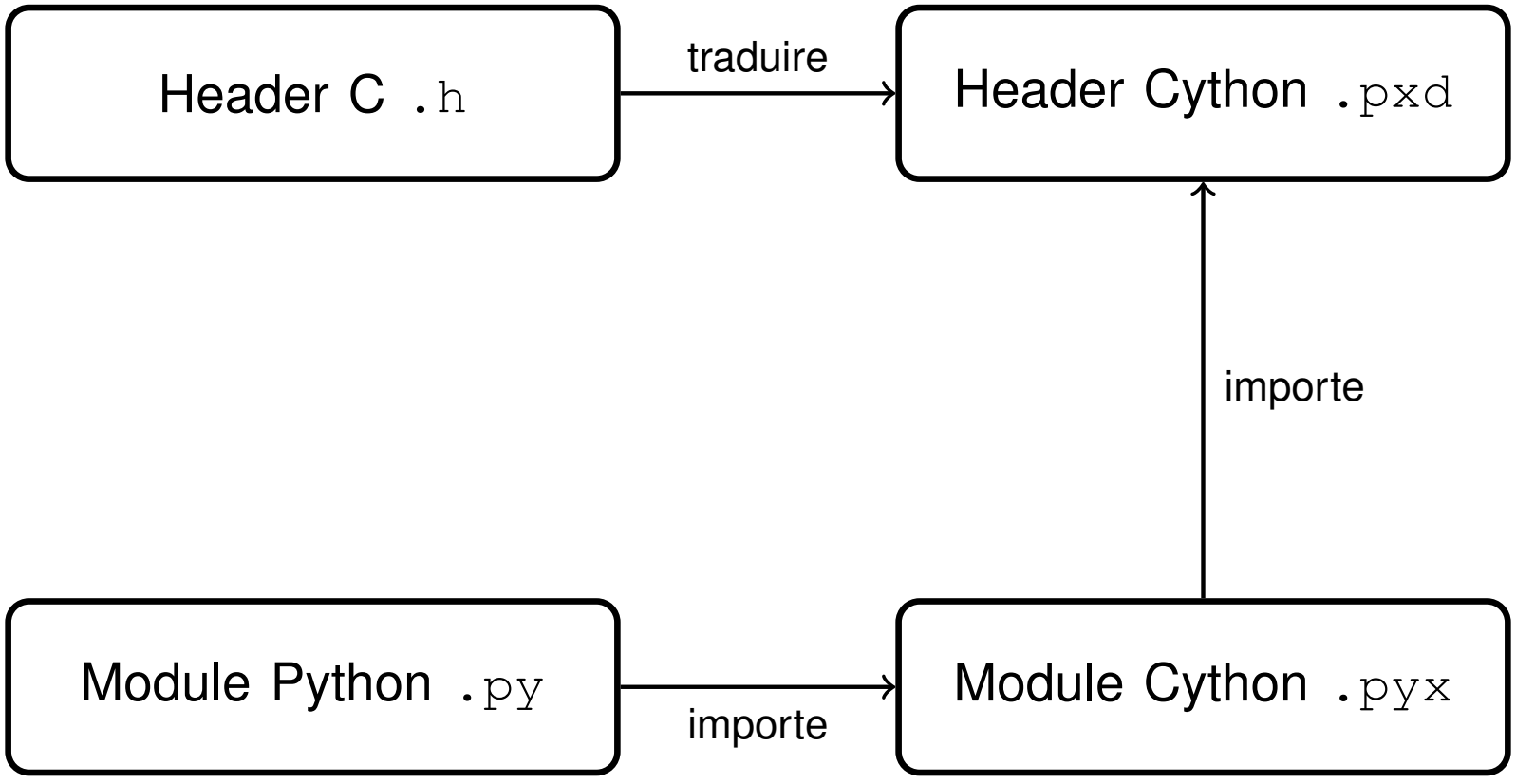
Décembre 2013

Interfaçage avec une librairie C existante

Motivations:

- Appeler des fonctions issues de librairies C existantes
- Rendre appelable par du C des fonctions python ou des fonctions C définies dans Cython.

Principe:



Interfacer une fonction C

Exemple : interfacer les fonctions `sin` et `cos` de `math.h` pour créer une fonction renvoyant le sinus et le cosinus

- Créer un header cython `math.pxd` dans un package approprié

```
cdef extern from "math.h":
    double sin(double angle)
    double cos(double angle)
```

- On peut appeler maintenant les deux fonctions dans un module cython `sincos.pyd` :

```
cimport math
def sincos(angle) :
  """
  Retourne le sinus et le cosinus de l'angle
  """
  return math.sin(angle),math.cos(angle)
```

- Script python utilisant le module cython :

```
from sincos import sincos
from math import pi
print sincos(0.25*pi)
```

Interfacer une structure C et ses fonctions dans une classe Python

Exemple : Interfacer une structure `Vecteur` avec ses fonctions associées

Le header C `Vecteur.h`

```c
typedef struct {
  int m_dim;
  double* m_arr_coefs;
} Vecteur;
Vecteur* vecteur_new(int dim);
void vecteur_free(Vecteur* vect);
int vecteur_getDim(const Vecteur* v);
int vecteur_is_null(const Vecteur* v);
double* vecteur_getCoefs(Vecteur* v);
double vecteur_normL2(const Vecteur* v);
void vecteur_add(const Vecteur* v1, const Vecteur* v2,
    Vecteur* v3);
```

Remarque : `is_null_vecteur` retourne en booléen sous forme d'entier.

Interfacer une structure C et ses fonctions dans une classe Python

Le header Cython `cVecteur.pxd` correspondant à `Vecteur.h`

```
# file : cVecteur.pxd

cdef extern from "Vecteur.h":
  ctypedef struct Vecteur:
    int m_dim
    double* m_arr_coefs

  Vecteur* vecteur_new(int dim)
  void vecteur_free(Vecteur* vect)
  void vecteur_add(Vecteur* v1, Vecteur* v2, Vecteur* v3)
  int vecteur_getDim(Vecteur* v)
  bint vecteur_is_null(Vecteur* v);
  double vecteur_getCoef(Vecteur* v, int i)
  void vecteur_setCoef(Vecteur* v, int i, double coef)
  double* vecteur_getCoefs(Vecteur* v)
  double vecteur_normL2(Vecteur* v)
```

Remarque : Remarquer le `bint` pour l'entier considéré comme booléen.

Interfacer une structure C et ses fonctions dans une classe Python

Pointeur opaque : En fait, les fonctions associées à la structure font qu'on n'a pas besoin de connaître la représentation interne des données de la structure `Vecteur`
L'utilisation d'un pointeur opaque permet de s'affranchir de cette représentation, et d'être indépendant d'un changement de réalisation éventuel dans une version future de `Vecteur`.
On modifie le début du header Cython `Vecteur.pxd`

```
# file : cVecteur.pxd

cdef extern from "Vecteur.h":
  ctypedef struct Vecteur:
    pass

  Vecteur* vecteur_new(int dim)
  ...
```

Interfacer une structure C et ses fonctions dans une classe Python

Constitution d'une classe Python `Vecteur` dans `pyvecteur.pyx`

Initialisation d'une instance de la classe

```
# file : pyvecteur.pyx
cimport cVecteur

cdef class Vecteur:
  cdef cVecteur.Vecteur* _c_vecteur

  def __cinit__(self, dim=0):
    self._c_vecteur = cVecteur.vecteur_new(dim)
```

Remarques :

- `__cinit__` est toujours appelée avant `__init__`, immédiatement à la construction de l'objet.
- Les paramêtre de `__cinit__` doivent coïncider avec ceux de `__init__`

Interfacer une structure C et ses fonctions dans une classe Python

Gestion des erreurs

Dans notre fonction d'initialisation, deux erreurs possibles :

- Dimension en paramêtre a une valeurs négative
- Mémoire insuffisante pour la taille du vecteur demandée.

On utilise les exceptions `ValueError` et `MemoryError` de Python :

```
# file : vecteur.pyx
cimport cVecteur

cdef class Vecteur:
  cdef cVecteur.Vecteur* _c_vecteur

  def __cinit__(self, int dim=0):
    if dim < 0:
      raise ValueError("Negative dimension")
    self._c_vecteur = cVecteur.vecteur_new(dim)
    if self._c_vecteur is NULL:
      raise MemoryError()
```

Interfacer une structure C et ses fonctions dans une classe Python

Gestion mémoire : détruire le vecteur alloué du côté C
Cython propose une fonction spéciale `__dealloc__` appelée lors de la destruction de l'instance. Dans notre cas, il faut libérer la structure C uniquement si l'allocation s'est bien faite à l'initialisation :

```
# file : vecteur.pyx
cimport cVecteur

 cdef class Vecteur:
  cdef cVecteur.Vecteur* _c_vecteur

  ...

  def __dealloc__(self):
    if self._c_vecteur is not NULL:
      cVecteur.vecteur_free(self._c_vecteur)
```

Interfacer une structure C et ses fonctions dans une classe Python

Interfaçage des méthodes :

Retour des normes L2 :

```
# file : vecteur.pyx
cimport cVecteur

 cdef class Vecteur:
   cdef cVecteur.Vecteur* _c_vecteur

   ...

   def normL2( self ):
     if self._c_vecteur is not NULL:
       return cVecteur.vecteur_normL2(self._c_vecteur)
     return 0.
```

Interfacer une structure C et ses fonctions dans une classe Python

Accesseurs et modifieurs :
Cython donne un mot clef `property` permettant de définir des attributs dérivés ou virtuels en lecture (`__get__`) ou en écriture (`__set__`).

```
# file : vecteur.pyx
cimport cVecteur

 cdef class Vecteur:
   cdef cVecteur.Vecteur* _c_vecteur

   ...

 property dimension:
   """Dimension du vecteur"""
   def __get__(self):
     return cVecteur.vecteur_getDim(self._c_vecteur)
```

Interfacer une structure C et ses fonctions dans une classe Python

Interfacer avec numpy : Cython a déjà un header pour numpy.

```
# file : vecteur.pyx
cimport cVecteur
cimport numpy as cnp
import numpy  as  np

cdef class Vecteur:
 cdef cVecteur.Vecteur* _c_vecteur
   ...
 property coefs:
   """Coefficients du vecteur"""
   def __get__(self):
     cdef cnp.npy_intp  dim[1]
     dim[0] = self.dimension
     return cnp.PyArray_SimpleNewFromData(1, dim,
            cnp.NPY_DOUBLE,cVecteur.vecteur_getCoefs(self.
                _c_vecteur))
   def __set__(self, array):
     cdef int dim = cnp.PyArray_DIM(array,0)
     cVecteur.vecteur_setDimension(self._c_vecteur, dim)
     cVecteur.vecteur_setCoefs(self._c_vecteur,
                    <double*>cnp.PyArray_DATA(array))
```

Interfacer une structure C et ses fonctions dans une classe Python

Opérateurs arithmétiques

```
# file : vecteur.pyx
cimport cVecteur
cimport numpy as cNumpy
import numpy

cdef class Vecteur:
 cdef cVecteur.Vecteur* _c_vecteur
   ...
 def __add__(Vecteur self, Vecteur v):
     if (self._c_vecteur is NULL) or (v._c_vecteur is NULL):
         return None
     if self.dimension != v.dimension :
         raise ValueError("Incompatible dimensions")
     w = Vecteur(self.dimension)
     cVecteur.vecteur_add(self._c_vecteur, v._c_vecteur, w.
         _c_vecteur)
     return w
```

Interface du C++ avec Cython

```
template<typename K>
class Vecteur
{
public:
  Vecteur(size_t dim);
  Vecteur(size_t dim, const K* coefs);
  Vecteur(const Vecteur& u);
  ~Vecteur();

  size_t getDimension() const { return m_dim; }
  void setDimension(size_t dim);
  const K& operator [](size_t i) const;
  K& operator [] (size_t i);
  K* getCoefs();
  const K* getCoefs() const;
  void setCoefs( const K* coefs);

  bool is_null() const;
  Vecteur<K> operator+ (const Vecteur<K>& u) const;
  Vecteur<K>& operator = (const Vecteur<K>& u);
  K normL2() const;
private:
  size_t m_dim;
  K* m_arr_coefs;
};
```

Interface du C++ avec Cython

```
cdef extern from "Vecteur.hpp" namespace "Algebra":
  cdef cppclass Vecteur[K]:
    Vecteur(size_t dim)
    Vecteur(size_t dim, K* data)
    Vecteur(Vecteur[K] u)

    size_t getDimension()
    void setDimension( size_t dim ) except +
    K& operator [] ( size_t i )
    K* getCoefs()
    void setCoefs( K* coefs )

    bint is_null()
    Vecteur[K] operator+( Vecteur[K] u ) nogil
    # Pas encore supporte
    #  Vecteur[K] operator=( Vecteur[K] u )
    K normL2()
```

Seules les classes C++ peuvent être déclarées comme template en Python.

Interface du C++ avec Cython

```
cdef class Vecteur:
  cdef cVecteur.Vecteur[double]* _c_vecteur

  def __cinit__(self, cnp.ndarray coefs = None):
      cdef int dim
      cdef cnp.npy_intp shape[1]
      if coefs is not None:
          shape[0] = cnp.PyArray_DIM(coefs,0)
          dim = <int>shape[0]
          self._c_vecteur = new cVecteur.Vecteur[double](dim,<
              double*>cnp.PyArray_DATA(coefs))

  def __dealloc__(self):
      if self._c_vecteur:
          del self._c_cvecteur
```

Une classe Cython ne peut être template !

Interface du C++ avec Cython

Appel aux méthodes d'une classe C++ :

```
cdef class Vecteur:
  ...
  def normL2(self):
       if self._c_vecteur is not NULL:
          return cVecteur._c_vecteur.normL2()
       return 0.
```

Interface du C++ avec Cython

```
cdef class Vecteur:
  ...
  property dimension:
    """Dimension du vecteur"""
    def __get__(self):
        return <int>self._c_vecteur.getDimension()
    def __set__(self, int dim):
        self._c_vecteur.setDimension(<size_t>dim)

  property coefs:
    """Coefficients de la matrice"""
    def __get__(self):
        cdef cnp.npy_intp shape[1]
        cdef double* array
        array = self._c_vecteur.getCoefs()
        shape[0] = <cnp.npy_intp>self.dimension
        ndarray = cnp.PyArray_SimpleNewFromData(1,shape,cnp.
            NPY_DOUBLE,<void*>array)
        return ndarray
    def __set__(self, cnp.ndarray array) :
        cdef int dim = cnp.PyArray_DIM(array,0)
        self._c_vecteur.setDimension(dim)
        # Attrape exception
        self._c_vecteur.setCoefs(<double*>cnp.PyArray_DATA(array
            ))
```