CNRS Autrans 2010

NumPy TP

Marc Poinot
Numerical Simulation
For
Aerodynamics and Aeroacoustics Dept.

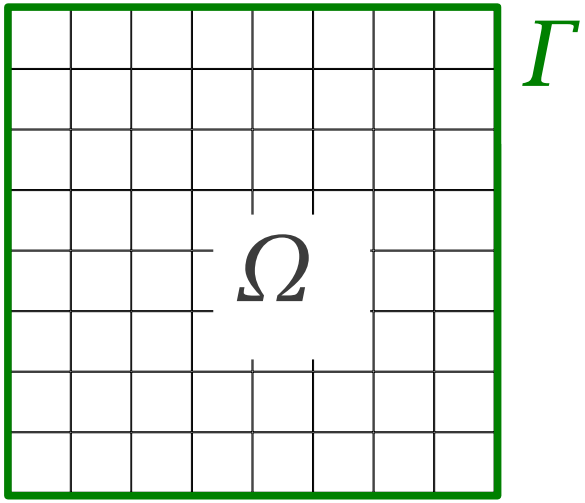
ONERA
THE FRENCH AEROSPACE LAB

retour sur innovation

Objectif

Calculer une propagation de chaleur

- Equation de la chaleur à l'équilibre
- Grille cartésienne



Problème

- u(x,y)
- $-\Delta u=0$ sur Ω
- $u=g(x,y)$ sur Γ
- Un rectangle chauffé sur 1 côté à une température T, température nulle sur les trois autres, trouver la répartition de température à l'équilibre
- Nous allons calculer le Laplacien de u :

$$\frac{(4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1})}{h^2}$$

Le module Heat.py

Un module est d'abord une interface

- L'interface est l'ensemble des services fournis par le module
- Faire le cas de test cible, celui-ci fixe une utilisation possible de notre interface, une fois établie une partie de l'interface, nous pouvons construire notre module, puis implémenter les fonctions

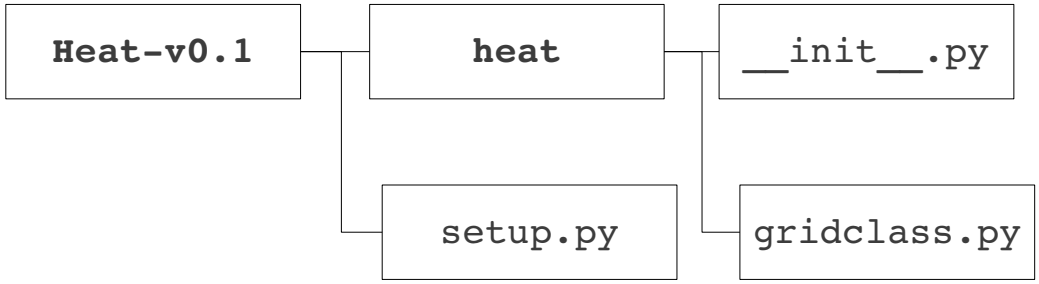
```
import heat

g=heat.Grid(17,23)
g.imin=180
g.imax=g.jmin=g.jmax=230
g.nextStep(1)
print g.nextStep()

h=g.copy()
h.imin=185
print h.nextStep()
```

- 1 - Faire un module Heat
- 2 - Faire une interface factice permettant d'exécuter le cas test

Correction - 1

- Créer arbre module
- Faire python **setup.py install –prefix=<MYPRIVATEDIR>**
- Recopier le chemin d'install
- Dans un autre shell **PYTHONPATH=<INSTALLDIR>**

```
from gridclass import Grid
```




```
from distutils.core import setup

setup(
name      = 'Heat library',
version   = 'v0.1',
packages = ['heat']
    )
```

```
import sys

class Grid:
  def __init__(self,xsize,ysize):
    self.xsize=xsize
    self.ysize=ysize
    self.imin=0
    self.jmin=0
    self.imax=0
    self.jmax=0
  def nextStep(self,step=sys.maxint):
    pass
  def copy(self):
    g=Grid(self.xsize,self.ysize)
    g.imin=self.imin
    g.jmin=self.jmin
    g.imax=self.imax
    g.jmax=self.jmax
    return g
```

Grille, initialisation et calcul

Classe Grid

- Nous implémentons les services de la classe sans toucher l'interface
  - Parfois il est possible de détecter des paramétres internes oubliés lors de la création du cas d'utilisation, il faut alors modifier l'interface et converger vers un compromis utilisateur/implémenteur.

- 1 – Constuire un tableau NumPy support de la grille
- 2 – Faire une fonction d'initialisation
- 3 – Faire un Laplacien avec une boucle Python
- 4 – Faire un Laplacien de type ufunc
- 5 – Implémenter *nextStep* qui résoud l'équation
- 6 – Ajouter un service print-like qui affiche la grille avec les températures par noeuds
- 7 – Ajouter un service de lecture/écriture de classe Grid
- 8 – Documenter le source
- 9 – Ajouter des exceptions: dimensions en I et j incorrectes, températures incorrectes...

- Pour les perfs, la visualisation, l'implémentation en fortran... attendre les cours suivant

Exemple d'implémentation - 1

```
import sys
import numpy
class Grid:
  def __init__(self,xsize,ysize):
    self.xsize=xsize
    self.ysize=ysize
    self.imin=0
    self.jmin=0
    self.imax=0
    self.jmax=0
    self.eps=numpy.finfo(numpy.float64).eps
    self._step=0
    self._last=0
    self._data=numpy.ones((xsize,ysize),dtype='d')*numpy.nan
    self._residual=numpy.finfo(numpy.float64).max
  def copy(self):
    g=Grid(self.xsize,self.ysize)
    g.imin=self.imin
    g.jmin=self.jmin
    g.imax=self.imax
    g.jmax=self.jmax
    return g
  def __str__(self):
    return str(self._data)
  def load(self,filename):
    self._data=numpy.load(filename)
  def save(self,filename):
    numpy.save(filename,self._data)
  def initialize(self):
    self._data[:,:]=0.0
    self._data[0,:]=self.imin
    self._data[-1,:]=self.imax
    self._data[:,0]=self.jmin
    self._data[:,-1]=self.jmax
```

Exemple d'implémentation - 2

```
def laplacien_boucle(self):
  self._residual=0
  a=self._data
  for i in range(1,self.xsize-1):
    for j in range(1,self.ysize-1):
      u_previous=a[i][j]
      a[i][j]=(a[i][j+1]+a[i+1][j]+a[i][j-1]+a[i-1][j])/4.0
      tmp=abs((u_previous-a[i][j])/a[i][j])
      if ((a[i][j]>0) and (self._residual<tmp)):
        self._residual=tmp
  return self._residual
def laplacien_ufunc(self):
    self._residual=0
  a=self._data
  u_previous=a[1:-1, 1:-1]
  u_next=(a[1:-1,2:]+a[2:,1:-1]+a[1:-1,:-2]+a[:-2,1:-1])/4.0
  u_mask=u_next>0.0
  self._residual=(numpy.repeat(numpy.fabs(u_previous-u_next).flat,u_mask.flat)\
                  /numpy.repeat(u_next.flat,u_mask.flat)).max()
  self._data[1:-1, 1:-1]=u_next
  return self._residual
def nextStep(self,step=sys.maxint):
  if (self._step==0): self.initialize()
  self._last=step
  while ((self._step<=self._last) and (self._residual>self.eps)):
    print 'Step # ',self._step
    self.laplacien_boucle()
    self._step+=1
  return self
```