第23回オープンCAE勉強会@関西
2013.7.6

OpenMDAOの最適化を試す

片山 達也

OpenMDAOって？

- 複合化領域の設計・解析・最適化ツール
  - いわゆるパラメトリックな最適化ツール
  - DAKOTA ?

- Pythoのフレームワーク
  - Pythonの豊富なモジュールが利用できる
  - 見渡せば、いろいろなツールでpythonが使われている
    （freeCAD,SALOME-MECA,PARAVIEW,pyFoam）
  - Plugin方式で拡張でき便利そう

    pyOpt_driver　ommodelwrapper　Dakota_driver

    excelwrapper　nastranwrapper　などなど
  - GUIもある

- オープンソース
  - 開発元：NASA Glenn Research Center
  - ここ数カ月毎月バージョンup(2013.7.6現在 version 0.7)

インストールについて

- 基本的にHP通り実施すればインストール可能

  URL：http://openmdao.org/releases/0.7.0/docs/

- インストール後のテスト[openmdao test]コマンドを行うためには各OSに別のコンパイラが必要

  URL：http://openmdao.org/releases/0.7.0/docs/dev-guide/intro.html#developer-requirements

  （Windowsでmingw32を使う場合pydistutils.cfgも忘れずに）

- Pluginのインストール方法

  URL：http://openmdao.org/releases/0.7.0/docs/getting-started/using_plugins.html

  Pluginは他のツールをOpenMDAOのフレームワークで使えるようにするものなので、他のツールは別途インストールする必要がある

pyOptのインストールについて

- pyOptのインストールの前にSwigのビルドが必要

  URL：
  http://www.swig.org/Doc2.0/SWIGDocumentation.html#Preface_installation

  ●windowsの場合

  mingw32,msys環境でビルドできる

  URL：http://www.swig.org/Doc2.0/Windows.html

  mysにてmingw/binにPATHを通して

  ./configure

  PCREがなんちゃらと言われた場合で不要であれば、
  ./configure --without-pcre

- pyOptのインストール

  URL：http://www.pyopt.org/install.html

  python setup.py install

  のコマンドでインストール可能

見ての通り、詳しいことは調べていません
まずは手探りでやってみました

T字型パイプの流れ解析

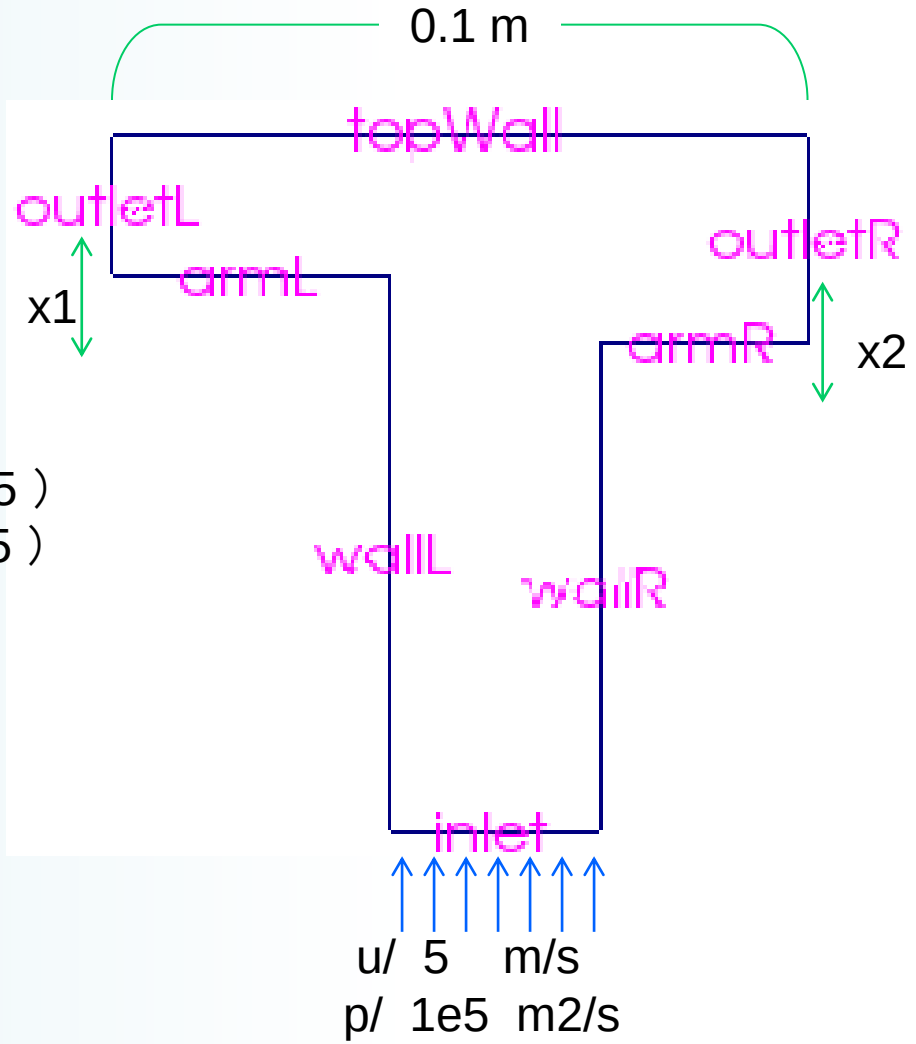
- 最適化問題
  - 目的関数：

    ①outLetLの最大流速：f1
    ②outLetRの最大流速：f2
    ③outLetLの流量：f3
    ④outLetRの流量：f4
    ⑤topWallが受ける流体力：f5

  - 設計変数：

    armLの寸法 ：x1 (-0.005～ 0.005 ）
    armRの寸法：x2 (-0.003～ 0.005 ）

    outLetL/Rの境界条件
    p/outLetInlet
    u/pressureInletOutletVelocity



主要なクラスの感覚的な説明

- Component
  - 計算や解析を行うクラス
  - 設計変数(入力)を設定してexecute()すれば出力に値が入る
  - CAE的には、モデルの変更～解析実行～値の取得(ポスト)を行う

- Driver
  - 運転手さん
  - Componentなどを使って最適化のためや近似モデル作成のための計算を行う

- MetaModel
  - 近似モデル
  - Componentを継承しているので、近似モデル完成後はComponentと同じように使われる

- Assembly
  - 場。データの引き渡しを司る

cfdComponent.py （1）

1. 形状変更
   - 0/ pointDisplacementの変更とmoveDyamicMeshを複数回実行して形状を変更
   - Windows版openfoamのコマンドをpython経由で実行できるようなクラスを作った

```
class tPipeCFD(Component):
    """ -0.005 <= x1 <=0.005
        -0.003 <= x2 <=0.005
    """
    # set up interface to the framework
    x1 = Float(0.0, iotype='in', desc='The variable of
    x2 = Float(0.0, iotype='in', desc='The variable of

    f1 = Float(iotype='out', desc='outletL_magU')
    f2 = Float(iotype='out', desc='outletR_magU')
    f3 = Float(iotype='out', desc='outletL_massFlow')
    f4 = Float(iotype='out', desc='outletR_massFlow')
    f5 = Float(iotype='out', desc='force at topWall')


    def execute(self):
```

```
    def execute(self):
        """CFD Analysis with OpenFOAM
            in C:\archive\OpenFOAM_win64\OpenFOAM\USERNAME-2.1.x\run\tPipe
        """
        print "calcate tPipeCFD ;x1=" + str(self.x1) +",x2="+str(self.x2)
        tPipe=foam4MDAO.foamCase(
            'C:\\archive\\OpenFOAM_win64\\OpenFOAM\\USERNAME-2.1.x\\run\\tPipe'
        x1 = self.x1
        x2 = self.x2
    #**preprocessing**
        #clean up
        tPipe.removeExecute(["0.org","constant","system"])
        tPipe.cp_r("0.org","0")
        tPipe.cp_r("constant/polyMesh/points.org","constant/polyMesh/points")
        tPipe.cp_r("system/controlDict.moveDynamicMesh","system/controlDict")

        # armL moves
        tPipe.cp_r("0/pointDisplacement.armL","0/pointDisplacement")
        tmpStr="        uniformValue    (0 " + str(x1) +" 0);"
        tPipe.replaceLine('0/pointDisplacement',55,tmpStr)
        tPipe.replaceLine('0/pointDisplacement',61,tmpStr)
        tPipe.replaceLine('0/pointDisplacement',68,tmpStr)
        tmpStr="        value   uniform (0 "+ str(x1) +" 0);"
        tPipe.replaceLine('0/pointDisplacement',56,tmpStr)
        tPipe.replaceLine('0/pointDisplacement',62,tmpStr)
        tPipe.replaceLine('0/pointDisplacement',69,tmpStr)
        ## run
        tPipe.runApplication("moveDynamicMesh  -noFunctionObjects")
        ## get point
        tPipe.cp_r("1/polyMesh/points","constant/polyMesh/points")
        ## clean up
        tPipe.removeExecute(["0.org","constant","system"])
```

ファイルの変更、コピー、削除、コマンドの実行を行う独自クラスを作成

cfdComponent.py （２）

2. 解析実行

- 前頁と同様にして解析を行う

3. 値の取得(ポスト処理)

- ポスト値はfunctionObjectで定めておく

```
122     #**cfd processing**
123         tPipe.cp_r("system/controlDict.simpleFoam","system/controlDict")
124         tPipe.runApplication("simpleFoam")
125 
126         #inletP=tPipe.getCell(".\\inletPressure\\0\\faceSource.dat",)
127         uL=tPipe.getCell("outletLMaxU/0/faceSource.dat",4,3)[1:-2].split()
128         uR=tPipe.getCell("outletRMaxU/0/faceSource.dat",4,3)[1:-2].split()
129 
130         self.f1=math.sqrt(float(uL[0])**2+float(uL[1])**2+float(uL[2])**2)
131         self.f2=math.sqrt(float(uR[0])**2+float(uR[1])**2+float(uR[2])**2)
132         self.f3=float(tPipe.getCell("outletLMassFlow/0/faceSource.dat",4,3))
133         self.f4=float(tPipe.getCell("outletRMassFlow/0/faceSource.dat",4,3))
134         self.f5=float(tPipe.getCell("forces/0/forces.dat",2,2," "))
135 
136 if __name__ == '__main__':
137     sim=tPipeCFD()
138     sim.x1=-0.005
139     sim.x2=0.005
140     sim.execute()
141     print sim.f1,sim.f2,sim.f3,sim.f4,sim.f5
142 
```

creatMetamodel.py

1. 近似モデルを作成するアセンブリ
   - ２設計変数について4水準作成し5x5のフルマトリクスの実験計画
   - 近似モデルについては、クリギング法のクラスを使用。
     （ただし通常のクリギング法のクラスではなくFloatKrigingSurrogate ）

```
14 from openmdao.main.api import Assembly, Component, SequentialWorkflow, set_as_t
15 from math import sin, cos
16
17 from openmdao.lib.datatypes.api import Float
18 from openmdao.lib.drivers.api import DOEdriver
19 from openmdao.lib.doegenerators.api import FullFactorial, Uniform
20 from openmdao.lib.components.api import MetaModel
21 from openmdao.lib.casehandlers.api import DBCaseRecorder
22 from openmdao.lib.surrogatemodels.api import LogisticRegression, FloatKrigingSu
23
24 import cfdComponent
25 reload(cfdComponent)
26
27 class Simulation(Assembly):
28
29     def configure(self):
30
31         #Components
32         self.add("tPipeMeta",MetaModel())
33         self.tPipeMeta.model = cfdComponent.tPipeCFD()
34
35         self.tPipeMeta.sur_f1 = FloatKrigingSurrogate()
36         self.tPipeMeta.sur_f2 = FloatKrigingSurrogate()
37         self.tPipeMeta.sur_f3 = FloatKrigingSurrogate()
38         self.tPipeMeta.sur_f4 = FloatKrigingSurrogate()
39         self.tPipeMeta.sur_f5 = FloatKrigingSurrogate()
40
41         self.tPipeMeta.recorder = DBCaseRecorder()
```

```
41         self.tPipeMeta.recorder = DBCaseRecorder()
42
43         #Training the MetaModel
44         self.add("DOE_Trainer",DOEdriver())
45         self.DOE_Trainer.DOEgenerator = FullFactorial()
46         self.DOE_Trainer.DOEgenerator.num_levels = 5
47         self.DOE_Trainer.add_parameter("tPipeMeta.x1",low=-0.005,high=0.005)
48         self.DOE_Trainer.add_parameter("tPipeMeta.x2",low=-0.003,high=0.005)
49         self.DOE_Trainer.case_outputs = ["tPipeMeta.f1","tPipeMeta.f2",
50                                          "tPipeMeta.f3","tPipeMeta.f4","tPipeMeta.f5"]
51         self.DOE_Trainer.add_event("tPipeMeta.train_next")
52         self.DOE_Trainer.recorders = [DBCaseRecorder()]
53
54         #Iteration Hierarchy
55         self.driver.workflow = SequentialWorkflow()
56         self.driver.workflow.add(['DOE_Trainer'])
57         self.DOE_Trainer.workflow.add('tPipeMeta')
58
59
60 if __name__ == '__main__':
61
62     sim = set_as_top(Simulation())
63     sim.run()
64
65     sim.tPipeMeta.save_to_egg("tPipeMeta","0.3")
66
```

validateMetaModel.py

1. 先に作成した近似モデルの評価を行う
   - ランダム抽出した16点のサンプリング点における
   - 近似モデルについては、クリギング法のクラスを使用。
     （ただし通常のクリギング法のクラスではなくFloatKrigingSurrogate ）

```
27 class Simulation(Assembly):
28
29     def configure(self):
30
31         #Components
32         self.add("tPipeMeta",MetaModel())
33         self.tPipeMeta.model = cfdComponent.tPipeCFD()
34
35         self.tPipeMeta.sur_f1 = FloatKrigingSurrogate()
36         self.tPipeMeta.sur_f2 = FloatKrigingSurrogate()
37         self.tPipeMeta.sur_f3 = FloatKrigingSurrogate()
38         self.tPipeMeta.sur_f4 = FloatKrigingSurrogate()
39         self.tPipeMeta.sur_f5 = FloatKrigingSurrogate()
40
41         self.tPipeMeta.recorder = DBCaseRecorder()
42
43         #Training the MetaModel
44         self.add("DOE_Trainer",DOEdriver())
45         self.DOE_Trainer.DOEgenerator = FullFactorial()
46         self.DOE_Trainer.DOEgenerator.num_levels = 5
47         self.DOE_Trainer.add_parameter("tPipeMeta.x1",low=-0.005,high=0.005)
48         self.DOE_Trainer.add_parameter("tPipeMeta.x2",low=-0.003,high=0.005)
49         self.DOE_Trainer.case_outputs = ["tPipeMeta.f1","tPipeMeta.f2",
50                                          "tPipeMeta.f3","tPipeMeta.f4","tPipeMeta.f5"]
51         self.DOE_Trainer.add_event("tPipeMeta.train_next")
52         self.DOE_Trainer.recorders = [DBCaseRecorder()]
53
54         #Iteration Hierarchy
```

関数	相関係数
f1	0.962
f2	0.994
f3	0.993
f4	0.993
f5	0.883

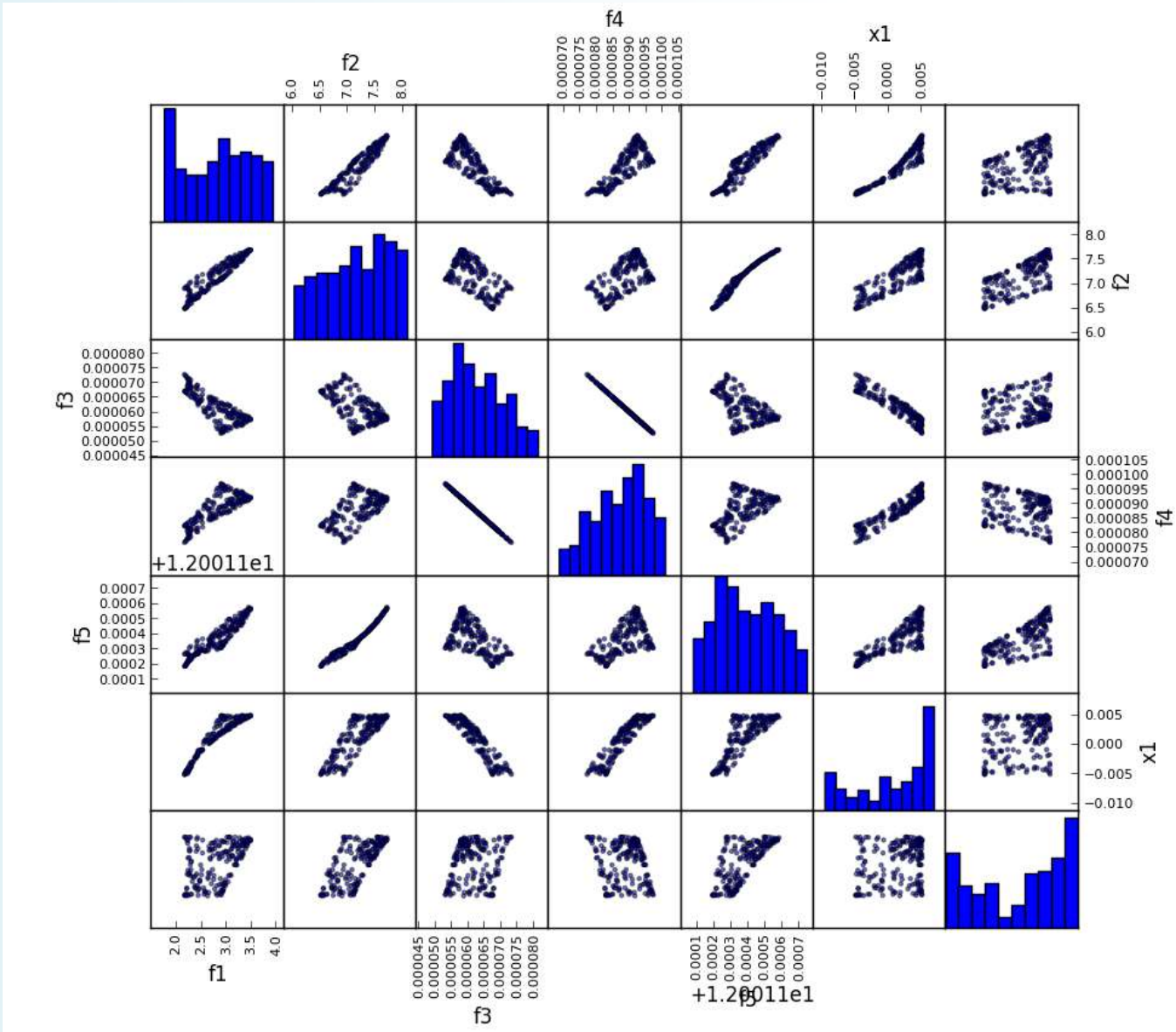
f5の相関係数が低い
理由は設計変数の変化
に対する感度が悪いため

OptMetaModel.py

1. メタモデルを用いた多目的最適化
   - pyOptのNSGA2を用いて多目的最適化を行う
   - 母集団200,10世代にて計算

```
class MultiObjectiveOptimization(Assembly):
    """Multi Objective optimization of the  with NSGA2."""

    def configure(self):
        """ Creates a new Assembly containing a MultiFunction and an optimizer""

        # pylint: disable-msg=E1101

        # Create MultiFunction component instances
        import os,fnmatch
        a=os.listdir(os.getcwd())
        fname=fnmatch.filter(a,"tPipeMeta-0.3*.egg")[0]

        mm=MetaModel.load_from_eggfile(fname)
        self.add(mm.name,mm)

        # Create NSGA2 Optimizer instance
        self.add('driver', pyOptDriver())

#        self.driver.recorders=[DBCaseRecorder()]

        # Driver process definition
        self.driver.workflow.add(mm.name)

        self.driver.print_results = True

        # NSGA2 Objective
        self.driver.add_objective(mm.name +'.f1')
        self.driver.add_objective(mm.name +'.f2')
        self.driver.add_objective(mm.name +'.f3')
        self.driver.add_objective(mm.name +'.f4')
        self.driver.add_objective(mm.name +'.f5')

        # NSGA2 Design Variable
        self.driver.add_parameter(mm.name +'.x1',low=-0.005,high=0.005)
        self.driver.add_parameter(mm.name +'.x2',low=-0.003,high=0.005)

        # NSGA2 Constraints
#        self.driver.add_constraint('multifunction.g1_x  >= 6.0')
#        self.driver.add_constraint('multifunction.g2_x  >= 1.0')


def main():

        try:
            from pyopt_driver.pyopt_driver import pyOptDriver
        except ImportError:
            pass

        sim = MultiObjectiveOptimization()
        set_as_top(sim)

        try:
            sim.driver.optimizer = 'NSGA2'
        except ValueError:
            raise SkipTest("NSGA2 not present on this system")

        # PyOpt Flags
        sim.driver.title='multi-Objective Opt with tPipeMeta'
        optdict = {}
        optdict['PopSize'] = 200     #   a multiple of 4
        optdict['maxGen'] = 10
        optdict['pCross_real'] = 0.6 #prob of crossover of design variables in r
        optdict['pMut_real'] = 0.5   #prob of mutation of (1/design varaibles)
        optdict['eta_c'] = 10.0      #distribution index for crossover in range
```

OptMetaModel.py



まとめ