MDO_architectures

December 17, 2016

0.1 SELLAR'S PROBLEM

This is a coupled disciplinary problem in OpenMDAO and has two disciplines. The problem can be stated as follows:

minimize $x_1^2 + z_2 + y_1 + e^{-y_2}$
w.r.t : z_1, z_2, x_1
subject to : $y_1 \overline{3.16-1\geq0\$}$
$1-y_2 \overline{24\geq0\$}$
$ -10 $\leq z_1 \leq 10$$
$ 0 $\leq z\_2 \leq 10$$
$ 0 $\leq x_1 \leq 10$$
Discipline 1 : $y_1(z_1, z_2, x_1, y_2) = z_1^2 + x_1 + z_2 - 0.2y_2$
Discipline 2 : $y_2 (z_1,z_2,y_1) = $\sqrt{y_1} + z\_1 + z\_2$$

z_1 and $ z_2 $ are the global design variables while x_1 is a local design variable. y_1 and y_2 are coupling variables between then two disciplines.

This coupling creates a non-linear system of equations which must be satisfied for valid solutions.

0.2 TYPES OF MDO ARCHITECTURES

- Multidisciplinary Design Feasible (MDF)

In this architecture the disciplines are directly coupled via some kind of solver, and the design variables are optimized all at the top level.

In [21]:
```
# sellar problem using MDF architecture
# For printing, use this import if you are running Python 2.x
from __future__ import print_function
from openmdao.api import ExecComp, IndepVarComp, Group, NLGaussSeidel, \
                         ScipyGMRES
from openmdao.api import Problem, ScipyOptimizer
import numpy as np

from openmdao.api import Component


class SellarDis1(Component):
    """Component containing Discipline 1."""
```

```python
    def __init__(self):
        super(SellarDis1, self).__init__()

        # Global Design Variable
        self.add_param('z1', val=0.0)
        self.add_param('z2', val=0.0)

        # Local Design Variable
        self.add_param('x', val=0.0)

        # Coupling parameter
        self.add_param('y2', val=1.0)

        # Coupling output
        self.add_output('y1', val=1.0)

    def solve_nonlinear(self, params, unknowns, resids):
        """Evaluates the equation
        y1 = z1**2 + z2 + x1 - 0.2*y2"""

        z1 = params['z1']
        z2 = params['z2']
        x1 = params['x']
        y2 = params['y2']

        unknowns['y1'] = z1**2 + z2 + x1 - 0.2*y2

   # def linearize(self, params, unknowns, resids):
        """ Jacobian for Sellar discipline 1."""
   #      J = {}

   #      J['y1','y2'] = -0.2
   #      J['y1','z'] = np.array([[2*params['z'][0], 1.0]])
   #      J['y1','x'] = 1.0

   #      return J


class SellarDis2(Component):
    """Component containing Discipline 2."""

    def __init__(self):
        super(SellarDis2, self).__init__()

        # Global Design Variable
        self.add_param('z1', val=0.0)
        self.add_param('z2', val=0.0)
```

```
        # Coupling parameter
        self.add_param('y1', val=1.0)

        # Coupling output
        self.add_output('y2', val=1.0)

    def solve_nonlinear(self, params, unknowns, resids):
        """Evaluates the equation
        y2 = y1**(.5) + z1 + z2"""

        z1 = params['z1']
        z2 = params['z2']
        y1 = params['y1']

# Note: this may cause some issues. However, y1 is constrained to be
# above 3.16, so lets just let it converge, and the optimizer will
# throw it out
        y1 = abs(y1)

        unknowns['y2'] = y1**.5 + z1 + z2

   #def linearize(self, params, unknowns, resids):
        """ Jacobian for Sellar discipline 2."""
#     J = {}
#     J['y2', 'y1'] = .5*params['y1']**-.5
#    #Extra set of brackets below ensure we have a 2D array instead of a 1
# for the Jacobian;  Note that Jacobian is 2D (num outputs x num inputs).
#   J['y2', 'z'] = np.array([[1.0, 1.0]])
# return J




class SellarDerivatives(Group):
    """ Group containing the Sellar MDA. This version uses the disciplines
    with derivatives."""

    def __init__(self):
        super(SellarDerivatives, self).__init__()


        self.deriv_options['type'] = 'fd'


        self.add('px', IndepVarComp('x', 1.0), promotes=['x'])
        self.add('pz1', IndepVarComp('z1', 5.0), promotes=['z1'])
        self.add('pz2', IndepVarComp('z2', 2.0), promotes=['z2'])
```

```
        self.add('d1', SellarDis1(),promotes=['z1','z2', 'x', 'y1', 'y2'])
        self.add('d2', SellarDis2(),promotes=['z1','z2', 'y1', 'y2'])

        self.add('obj_cmp', ExecComp('obj = x**2 + z2 + y1 + exp(-y2)',
                                     z2=0.0, x=0.0, y1=0.0, y2=0.0),
                 promotes=['obj','z2', 'x', 'y1', 'y2'])

        self.add('con_cmp1', ExecComp('con1 = 3.16 - y1'),
                 promotes=['y1', 'con1'])
        self.add('con_cmp2', ExecComp('con2 = y2 - 24.0'),
                 promotes=['con2', 'y2'])

        self.nl_solver = NLGaussSeidel()
        self.nl_solver.options['atol'] = 1.0e-12

        self.ln_solver = ScipyGMRES()



top = Problem()
top.root = SellarDerivatives()

top.driver = ScipyOptimizer()
top.driver.options['optimizer'] = 'SLSQP'
top.driver.options['tol'] = 1.0e-8


top.driver.add_desvar('z1', lower=-10.0, upper=10.0)
top.driver.add_desvar('z2', lower=0.0, upper=10.0)
top.driver.add_desvar('x', lower=0.0, upper=10.0)

top.driver.add_objective('obj')
top.driver.add_constraint('con1', upper=0.0)
top.driver.add_constraint('con2', upper=0.0)

top.setup()

# Setting initial values for design variables
top['x'] = 1.0
top['z1'] = 5.0
top['z2'] = 2.0

top.run()

print("\n")
print( "Minimum found at (%f, %f, %f)" % (top['z1'], \
                                         top['z2'], \
```

```
                                                    top['x']))
        print("Coupling vars: %f, %f" % (top['y1'], top['y2']))
        print("Minimum objective: ", top['obj'])
```

```
##############################################
Setup: Checking root problem for potential issues...

No recorders have been specified, so no data will be saved.
Group '' has the following cycles: [['d1', 'd2']]

The following params are connected to unknowns that are updated out of order, so th

Setup: Check of root problem complete.
##############################################

Optimization terminated successfully.    (Exit mode 0)
            Current function value: [ 3.18339395]
            Iterations: 7
            Function evaluations: 8
            Gradient evaluations: 7
Optimization Complete
-----------------------------------



Minimum found at (1.977639, 0.000000, 0.000000)
Coupling vars: 3.160000, 3.755278
Minimum objective:  3.18339395313
```

- Individual Design Feasible (IDF)

In IDF, the direct coupling between the disciplines is removed, and the input coupling variables are added to the optimizer's design variables. The algorithm calls for two new equality constraints that enforce the coupling between the disciplines.

- Collaborative Optimization (CO)

CO is a two-level architecture with three optimizer loops, one at each discipline, and one acting globally. The global optimizer drives the design and coupling variables towards an optimal solution that minimizes the objective while constraining to zero the sum of the squares of the residuals between the values commanded by the global optimizer and those set by the local optimizers. Each local optimizer operates on its own discipline, driving its design variables while minimizing the residual between the actual value of the design variables and the values commanded by the global optimizer.

- Simultaneous ANalysis and Design (SAND)

In SAND, the optimizer minimizes the problem by varying the design variables simultaneously with the coupling variables to achieve feasibility and drive the residual constraint to zero.

This means the residual needs to be expressed explicitly so we don't need any implicit components or a solver. The optimizer does it all.

Here is the python code for solving the Sellar problem using the SAND architecture in openMDAO.

```
In [33]: from __future__ import print_function

         import time

         import numpy as np

         from openmdao.api import Component, Group, Problem,\
             IndepVarComp, ExecComp, NLGaussSeidel, \
             ScipyGMRES, ScipyOptimizer

         class SellarDis1(Component):
             """Component containing Discipline 1."""

             def __init__(self):
                 super(SellarDis1, self).__init__()

                 # Design Variable
                 self.add_param('z1', val=0.0)
                 self.add_param('z2', val=0.0)
                 self.add_param('x', val=0.0)
                 self.add_param('y2', val=1.0)
                 self.add_param('y1', val=1.0)

                 self.add_output('resid1', val=1.0)

             def solve_nonlinear(self, params, unknowns, resids):
                 """Evaluates the equation
                 y1 = z1**2 + z2 + x1 - 0.2*y2"""

                 z1 = params['z1']
                 z2 = params['z2']
                 x1 = params['x']
                 y2 = params['y2']
                 y1 = params['y1']

                 unknowns['resid1'] = z1**2 + z2 + x1 - 0.2*y2 - y1


         class SellarDis2(Component):
             """Component containing Discipline 2."""

             def __init__(self):
                 super(SellarDis2, self).__init__()
```

```
        # Global Design Variable
        self.add_param('z1', val=0.0)
        self.add_param('z2', val=0.0)
        self.add_param('y1', val=1.0)
        self.add_param('y2', val=1.0)

        self.add_output('resid2', val=1.0)

    def solve_nonlinear(self, params, unknowns, resids):
        """Evaluates the equation
        y2 = y1**(.5) + z1 + z2"""

        z1 = params['z1']
        z2 = params['z2']
        y1 = params['y1']
        y1 = abs(y1)
        y2 = params['y2']

        unknowns['resid2'] = y1**.5 + z1 + z2 - y2




class SellarSAND(Group):
    """ Group containing the Sellar MDA. This version uses the disciplines
    with derivatives."""

    def __init__(self):
        super(SellarSAND, self).__init__()


        self.deriv_options['type'] = 'fd'


        self.add('px', IndepVarComp('x', 1.0), promotes=['x'])
        self.add('pz1', IndepVarComp('z1', 5.0), promotes=['z1'])
        self.add('pz2', IndepVarComp('z2', 2.0), promotes=['z2'])
        self.add('py1', IndepVarComp('y1', 1.0), promotes=['y1'])
        self.add('py2', IndepVarComp('y2', 1.0), promotes=['y2'])



        self.add('d1', SellarDis1(),
             promotes=['resid1', 'z1', 'z2' , 'x' , 'y1', 'y2'])
        self.add('d2', SellarDis2(),
             promotes=['resid2', 'z1', 'z2' , 'y1', 'y2'])
```

```
                self.add('obj_cmp', ExecComp('obj = x**2 + z2 + y1 + exp(-y2)',
                                             z2=0.0, x=0.0, y1=0.0, y2=0.0),
                         promotes=['obj','z2', 'x', 'y1', 'y2'])


                self.add('con_cmp1', ExecComp('con1 = 3.16 - y1'),
                     promotes=['con1', 'y1'])
                self.add('con_cmp2', ExecComp('con2 = y2 - 24.0'),
                     promotes=['con2', 'y2'])


        top = Problem()
        top.root = SellarSAND()

        top.driver = ScipyOptimizer()
        top.driver.options['optimizer'] = 'SLSQP'
        top.driver.options['tol'] = 1.0e-12

        top.driver.add_desvar('z1', lower=-10.0, upper=10.0)
        top.driver.add_desvar('z2', lower=0.0, upper=10.0)
        top.driver.add_desvar('x', lower=0.0, upper=10.0)
        top.driver.add_desvar('y1', lower=-10.0, upper=10.0)
        top.driver.add_desvar('y2', lower=-10.0, upper=10.0)

        top.driver.add_objective('obj')
        top.driver.add_constraint('con1', upper=0.0)
        top.driver.add_constraint('con2', upper=0.0)
        top.driver.add_constraint('resid1', equals=0.0)
        top.driver.add_constraint('resid2', equals=0.0)

        top.setup()
        tt = time.time()
        top.run()


        print("\n")
        print( "Minimum found at (%f, %f, %f)" % (top['z1'], \
                                                  top['z2'], \
                                                  top['x']))
        print("Coupling vars: %f, %f" % (top['d1.y1'], top['d1.y2']))
        print("Minimum objective: ", top['obj'])

##############################################
Setup: Checking root problem for potential issues...

No recorders have been specified, so no data will be saved.
```

```
Setup: Check of root problem complete.
##############################################

Optimization terminated successfully.    (Exit mode 0)
            Current function value: [ 3.18339395]
            Iterations: 6
            Function evaluations: 7
            Gradient evaluations: 6
Optimization Complete
-----------------------------------


Minimum found at (1.977639, 0.000000, 0.000000)
Coupling vars: 3.160000, 3.755278
Minimum objective:  3.18339395164
```

```
In [ ]:
```