Optimal Design with OpenMDAO

Computational Design Laboratory

Department of Aerospace Engineering
Iowa State University

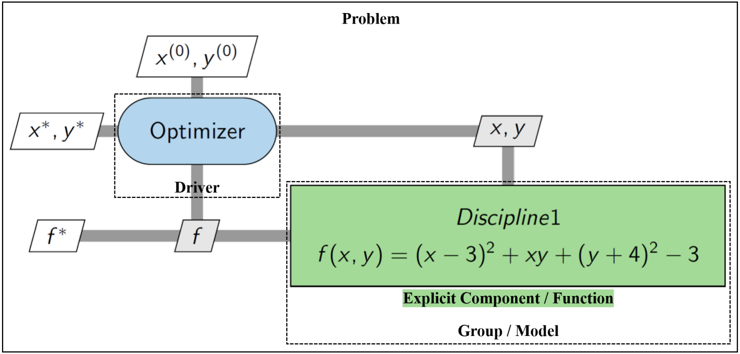
October 14, 2020

Outline

- Problem 1: Single-discipline optimization
- Problem 2: Two-discipline optimization
- Problem 3: Two-discipline optimization
- Problem 4: Airflow sensor system design
- Further reading

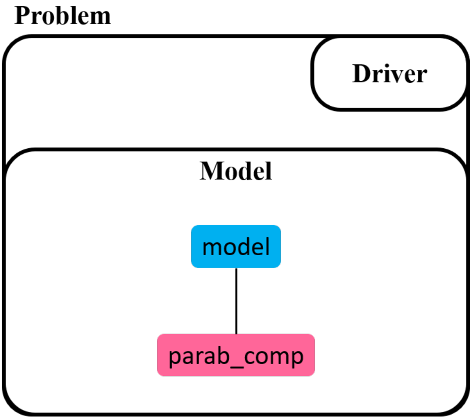
Problem 1: Single-discipline optimization

$$\begin{aligned} \text{Minimize} \quad & f(x,y) = (x-3)^2 + xy + (y+4)^2 - 3 \\ \text{w.r.t.} \quad & x, y \end{aligned}$$



The XDSM code for this figure: Link

Problem 1: Single-discipline optimization



Problem 1: Single-discipline optimization

- Download mdo_single_disp.py from Github: Link
- Part 0-7: Same as mda_single_disp.py

```python
# Part 0: OpenMDAO and component imports
import openmdao.api as om

# Part 1: Create a new explicit components for f_xy
class Paraboloid(om.ExplicitComponent):
    """
    Evaluates the equation f(x,y) = (x-3)^2 + xy + (y+4)^2 - 3.
    """

    def setup(self):
        self.add_input('x', val=0.0)
        self.add_input('y', val=0.0)

        self.add_output('f_xy', val=0.0)

    def setup_partials(self):
        # Finite difference all partials.
        self.declare_partials('*', '*', method='fd')

    def compute(self, inputs, outputs):
        """
        f(x,y) = (x-3)^2 + xy + (y+4)^2 - 3
        Minimum at: x = 6.6667; y = -7.3333
        """
        x = inputs['x']
        y = inputs['y']

        outputs['f_xy'] = (x - 3.0)**2 + x * y + (y + 4.0)**2 - 3.0
```

Problem 1: Single-discipline optimization

```python
if __name__ == "__main__":
    # Part 2: Create a group and Paraboloid as subsystem of group
    model = om.Group()
    model.add_subsystem('parab_comp', Paraboloid())

    # Part 3: Create problem from the group and setup the problem
    prob = om.Problem(model)
    prob.setup()

    # Part 4: Provide x and y input to the problem
    prob.set_val('parab_comp.x', 3.0)
    prob.set_val('parab_comp.y', -4.0)

    # Part 5: Run the problem
    prob.run_model()

    # Part 6: Print the input and output of the problem
    print('x =',prob['parab_comp.x'])
    print('y =',prob['parab_comp.y'])
    print('f_xy =',prob.get_val('parab_comp.f_xy'))

    print('\n---------------\n')
    # Part 7: Provide new input variables and print output
    prob.set_val('parab_comp.x', 5.0)
    prob.set_val('parab_comp.y', -2.0)
    prob.run_model()
    print('x =',prob['parab_comp.x'])
    print('y =',prob['parab_comp.y'])
    print('f_xy =', prob.get_val('parab_comp.f_xy'))
    print('\n---------------\n')
```

Problem 1: Single-discipline optimization

- Part 8: Build the model for optimization
- Part 9: Provide initial guess to variables
- Part 10: Setup the optimizer
- Part 11: Provide bounds and objective function

```python
# Part 8: Build the model for optimization
prob = om.Problem()
prob.model.add_subsystem('parab', Paraboloid(), promotes_inputs=['x', 'y'])

# Part 9: Provide initial values to x and y
prob.model.set_input_defaults('x', 3.0)
prob.model.set_input_defaults('y', -4.0)

# Part 10: Setup the optimizer
prob.driver = om.ScipyOptimizeDriver()
prob.driver.options['optimizer'] = 'COBYLA'

# Part 11: Provide bounds and objective function
prob.model.add_design_var('x', lower=-50, upper=50)
prob.model.add_design_var('y', lower=-50, upper=50)
prob.model.add_objective('parab.f_xy')
```

Problem 1: Single-discipline optimization

- Part 12: Setup the problem and run optimization
- Part 13: Print the results of the problem

```python
# Part 12: Setup the problem and run
prob.setup()
prob.run_driver()

# Part 13: Print the results
# minimum value
print('f_xy=', prob.get_val('parab.f_xy'))
# location of the minimum
print('x=', prob.get_val('x'))
print('y=', prob.get_val('y'))
```

- For more detailed explanation of the setup: Link

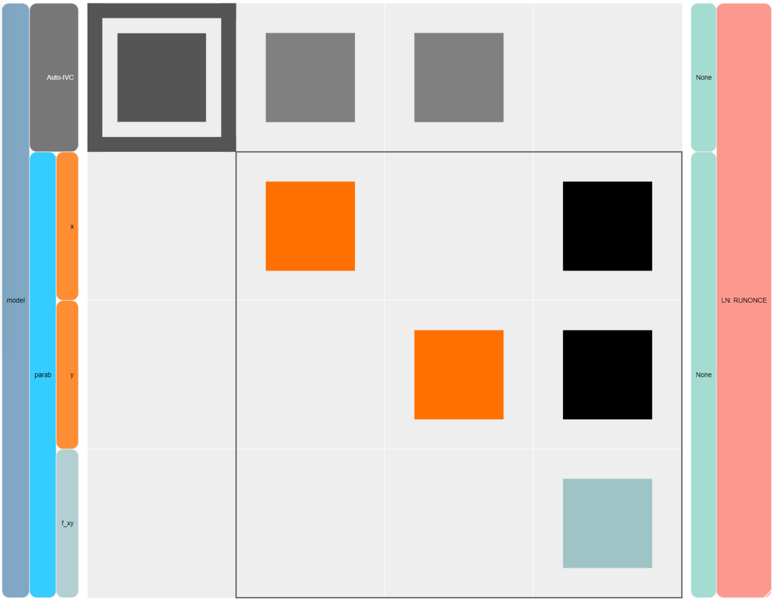
Problem 1: Single-discipline optimization

- Output

```
x = [3.]
y = [-4.]
f_xy = [-15.]

----------------

x = [5.]
y = [-2.]
f_xy = [-5.]

----------------

Optimization Complete
-----------------------------------
f_xy= [-27.33333333]
x= [6.66666719]
y= [-7.33333223]

In [2]:
```

IPython console History

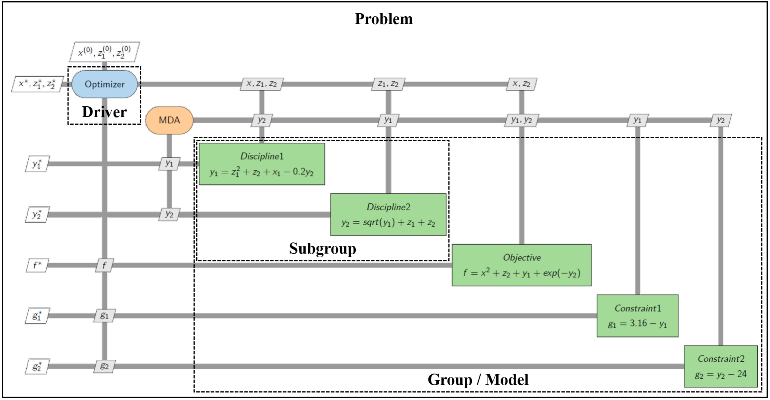
Problem 1: Single-discipline N2 diagram



Problem 2: Two-discipline optimization

$$\begin{aligned}
\text{Minimize} \quad & f = x^2 + z_2 + y_1 + \exp(-y_2) \\
\text{w.r.t.} \quad & x, z_1, z_2 \\
\text{s.t.} \quad & g_1 : 3.16 - y_1 \leq 0 \\
& g_2 : y_2 - 24.0 \leq 0 \\
\text{Discipline 1 :} \quad & y_1 = z_1^2 + z_2 + x_1 - 0.2y_2 \\
\text{Discipline 2 :} \quad & y_2 = \sqrt{y_1} + z_1 + z_2
\end{aligned}$$

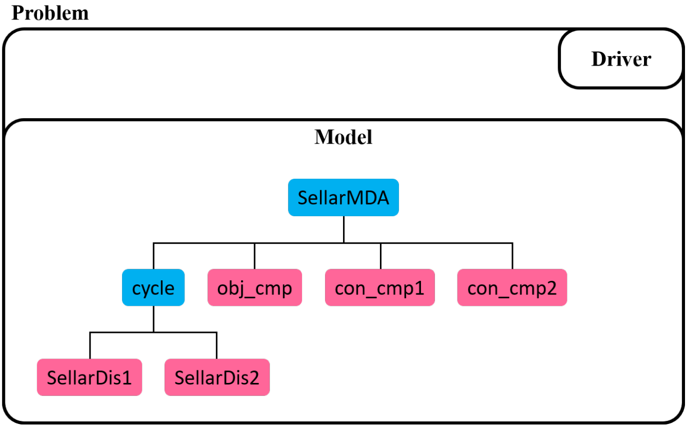
Problem 2: Two-discipline optimization



The XDSM code for this figure: Link

For more detailed explanation of the setup: Link

Problem 2: Two-discipline optimization



Problem 2: Two-discipline optimization

- Download mdo_sellar.py from Github: Link
- Part 1-6: Same as mda_sellar.py

```python
# Part 1: Import required packages
import openmdao.api as om
import numpy as np

# Part 2: Create new components for Discipline1 and 2
class SellarDis1(om.ExplicitComponent):
    """
    Component containing Discipline 1 -- no derivatives version.
    """
    def setup(self):
        # Global Design Variable
        self.add_input('z', val=np.zeros(2))

        # Local Design Variable
        self.add_input('x', val=0.)

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

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

        # Finite difference all partials.
        self.declare_partials('*', '*', method='fd')

    def compute(self, inputs, outputs):
        """
        Evaluates the equation
        y1 = z1**2 + z2 + x1 - 0.2*y2
        """
        z1 = inputs['z'][0]
        z2 = inputs['z'][1]
        x1 = inputs['x']
        y2 = inputs['y2']

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

class SellarDis2(om.ExplicitComponent):
    """
    Component containing Discipline 2 -- no derivatives version.
    """
```

Problem 2: Two-discipline optimization

```python
    def setup(self):
        # Global Design Variable
        self.add_input('z', val=np.zeros(2))

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

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

        # Finite difference all partials.
        self.declare_partials('*', '*', method='fd')

    def compute(self, inputs, outputs):
        """
        Evaluates the equation
        y2 = y1**(.5) + z1 + z2
        """
        z1 = inputs['z'][0]
        z2 = inputs['z'][1]
        y1 = inputs['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
        if y1.real < 0.0:
            y1 *= -1

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

# Part 3: Create group SellarMDA
class SellarMDA(om.Group):
    """
    Group containing the Sellar MDA.
    """
    def setup(self):
        indeps = self.add_subsystem('indeps', om.IndepVarComp(), promotes=['*'])
        indeps.add_output('x', 1.0)
        indeps.add_output('z', np.array([5.0, 2.0]))

        cycle = self.add_subsystem('cycle', om.Group(), promotes=['*'])
```

Problem 2: Two-discipline optimization

```
        cycle.add_subsystem('d1', SellarDis1(), promotes_inputs=['x', 'z', 'y2'],
                            promotes_outputs=['y1'])
        cycle.add_subsystem('d2', SellarDis2(), promotes_inputs=['z', 'y1'],
                            promotes_outputs=['y2'])

        # Nonlinear Block Gauss Seidel is a gradient free solver
        cycle.nonlinear_solver = om.NonlinearBlockGS(iprint=1)  # try iprint=2

        self.add_subsystem('obj_cmp', om.ExecComp('obj = x**2 + z[1] + y1 + exp(-y2)',
                                                  z=np.array([0.0, 0.0]), x=0.0),
                           promotes=['x', 'z', 'y1', 'y2', 'obj'])

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


# Part 4: Setup model and problem
prob = om.Problem()
prob.model = SellarMDA()
prob.setup()

# Part 5: Provide input to the problem
prob['x'] = 2.
prob['z'] = [-1., -1.]

prob.run_model()

#  Part 6:  print details
print('\nInput ---')
print('x :',prob['x'])
print('z1 :',prob['z'][0])
print('z2 :',prob['z'][1])

print('\nDiscipline output ---')
print('y1 :',prob['y1'])
print('y2 :',prob['y2'])

print('\nObjective and constraints---')
print('obj :',prob['obj'])
print('con1 :',prob['con1'])
print('con2 :',prob['con2'])
print('\n')
```

Problem 2: Two-discipline optimization

- Part 7: Setup the optimization and print output

```
#  Part 7: Optimizing the Problem
prob.driver = om.ScipyOptimizeDriver()
prob.driver.options['optimizer'] = 'SLSQP'
# prob.driver.options['maxiter'] = 100
prob.driver.options['tol'] = 1e-8

prob.model.add_design_var('x', lower=0, upper=10)
prob.model.add_design_var('z', lower=0, upper=10)
prob.model.add_objective('obj')
prob.model.add_constraint('con1', upper=0)
prob.model.add_constraint('con2', upper=0)

# Ask OpenMDAO to finite-difference across the model to compute the gradients for the optimizer
prob.model.approx_totals()

prob.setup()
prob.set_solver_print(level=0)

prob.run_driver()

print('\nminimum found at')
print('x :',prob.get_val('x')[0])
print('z1 :',prob.get_val('z')[0])
print('z2 :',prob.get_val('z')[1])

print('')
print('y1 :',prob.get_val('y1'))
print('y2 :',prob.get_val('y2'))

print('\nminumum objective and constraints')
print('obj :',prob.get_val('obj')[0])
print('con1 :',prob.get_val('con1'))
print('con2 :',prob.get_val('con2'))
```

- For more detailed explanation of the setup: Link

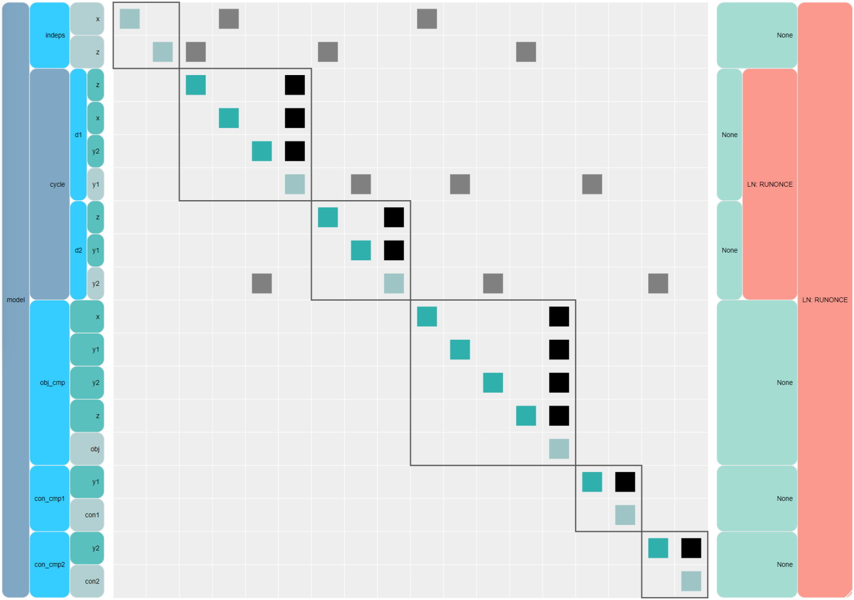
Problem 2: Two-discipline optimization

- Output

```
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 3.183393951729169
            Iterations: 6
            Function evaluations: 6
            Gradient evaluations: 6
Optimization Complete
-----------------------------------

minimum found at
x : 0.0
z1 : 1.977638883487764
z2 : 8.830566052859473e-15

y1 : [3.16]
y2 : [3.75527777]

minumum objective and constraints
obj : 3.183393951729169
con1 : [-8.97131258e-11]
con2 : [-20.24472223]

In [2]:
```

IPython console History log

Problem 2: Two-discipline N2 diagram

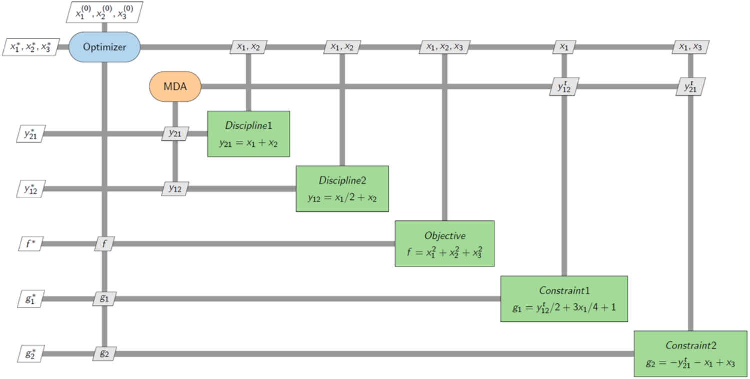


Problem 3: Two-discipline optimization

Problem formulation:

$$
\begin{aligned}
\text{Minimize} \quad & f = x_1^2 + x_2^2 + x_3^2 \\
\text{w.r.t.} \quad & x_1, x_2, x_3 \\
\text{s.t.} \quad & g_1 : \frac{y_{12}^t}{2} + \frac{3x_1}{4} + 1 \leq 0 \\
& g_2 : -y_{21}^t - x_1 + x_3 \leq 0 \\
\text{Discipline 1 :} \quad & y_{21} = x_1 + x_2 \\
\text{Discipline 2 :} \quad & y_{12} = \frac{x_1}{2} + x_2
\end{aligned}
$$

Problem 3: XDSM of MDF formulation



The XDSM code for this figure: Link

Problem 3: Two-discipline optimization

- Download mdo_analytical_mdf.py from Github: Link
- Part 1: Import required packages
- Part 2: Create new components for Analysis1 and 2

```python
# Part 1: Import required packages
import openmdao.api as om

# Part 2: Create new components for Analysis1 and 2
class Analysis1(om.ExplicitComponent):
    """
    Component containing Discipline1 and Constraint1
    """
    def setup(self):
        # Global Design Variable
        self.add_input('x1', val=0.0)
        self.add_input('x2', val=0.0)

        # Coupling parameter
        self.add_input('y12', val=1.0)

        # Coupling output
        self.add_output('y21', val=1.0)
        self.add_output('g1', val=1.0)

        # Finite difference all partials.
        self.declare_partials('*', '*', method='fd')

    def compute(self, inputs, outputs):
        """
        Evaluate y21, g1
        """
        x1 = inputs['x1']
        x2 = inputs['x2']
        y12 = inputs['y12']

        outputs['y21'] = x1 + x2
        outputs['g1']  = y12/2 + 3*x1/4  +1

class Analysis2(om.ExplicitComponent):
    """
    Component containing Discipline2 and Constraint2
    """
    def setup(self):
        # Global Design Variable
        self.add_input('x1', val=0.0)
```

Problem 3: Two-discipline optimization

- Part 3: Create group ProcessMDA

```python
# Part 3: Create group MDA
class ProcessMDA(om.Group):
    """
    Group containing MDA
    """
    def setup(self):
        indeps = self.add_subsystem('indeps', om.IndepVarComp(), promotes=['*'])
        indeps.add_output('x1', 1.0)
        indeps.add_output('x2', 1.0)
        indeps.add_output('x3', 1.0)

        cycle = self.add_subsystem('cycle', om.Group(), promotes=['*'])
        cycle.add_subsystem('d1', Analysis1(), promotes_inputs=['x1','x2','y12'],promotes_outputs=['y21','g1'])
        cycle.add_subsystem('d2', Analysis2(), promotes_inputs=['x1','x2','x3','y21'],promotes_outputs=['y12','g2'])

        # Nonlinear Block Gauss Seidel is a gradient free solver
        cycle.nonlinear_solver = om.NonlinearBlockGS()

        self.add_subsystem('obj_cmp', om.ExecComp('obj = x1**2 + x2**2 + x3**2 ',
                                                  x1=0.0, x2=0.0, x3=0.0),
                           promotes=['x1','x2','x3','obj'])

        self.add_subsystem('con_cmp1', om.ExecComp('con1 = g1'), promotes=['con1', 'g1'])
        self.add_subsystem('con_cmp2', om.ExecComp('con2 = g2'), promotes=['con2', 'g2'])
```

Problem 3: Two-discipline optimization

- Part 4: Build the model and problem
- Part 5: Setup the optimizer
- Part 6: Provide bounds and objective function

```
# Part 4: Build the model and problem for optimization
prob = om.Problem()
prob.model = ProcessMDA()

# Part 5: Setup optimizer
prob.driver = om.ScipyOptimizeDriver()
prob.driver.options['optimizer'] = 'SLSQP'
# prob.driver.options['maxiter'] = 100
prob.driver.options['tol'] = 1e-8

# Part 6: Provide bounds and objective function
prob.model.add_design_var('x1', lower=-4, upper=4)
prob.model.add_design_var('x2', lower=-4, upper=4)
prob.model.add_design_var('x3', lower=-4, upper=4)
prob.model.add_objective('obj')
prob.model.add_constraint('con1', upper=0)
prob.model.add_constraint('con2', upper=0)

prob.setup()
prob.set_solver_print(level=0)
```

Problem 3: Two-discipline optimization

- Part 7: Run the model with initial values
- Part 8: Run optimization and print the results

```python
# Part 7: Run model with initial values
print('\nSingle evaluation')
prob['x1'] = 2.
prob['x2'] = 2.
prob['x3'] = 2.
prob.run_model()
print('x1 :',prob['x1'])
print('x2 :',prob['x2'])
print('x3 :',prob['x3'])
print('g1 :',prob['g1'])
print('g2 :',prob['g2'])
print('obj :',prob['obj'][0])
print('\n')

# Part 8: Run optimization and print outputs
# Ask OpenMDAO to finite-difference across the model to compute the gradients for the optimizer
prob.model.approx_totals()
prob.run_driver()
# --------------------------
print('minimum found at')
print('x1 :',prob['x1'])
print('x2 :',prob['x2'])
print('x3 :',prob['x3'])
print('g1 :',prob['g1'])
print('g2 :',prob['g2'])
print('minumum objective')
print('obj :',prob['obj'][0])
```

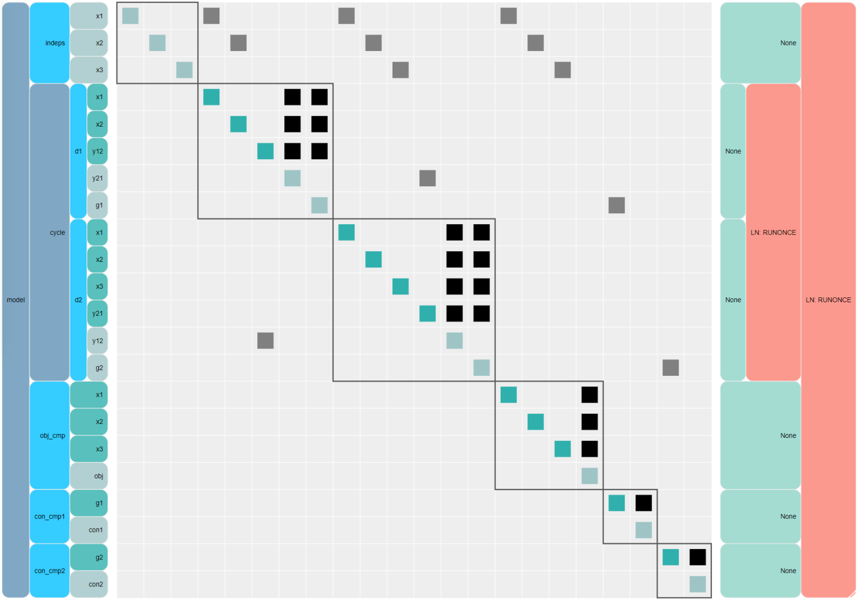
Problem 3: Two-discipline optimization

- Output

```
Single evaluation
x1 : [2.]
x2 : [2.]
x3 : [2.]
g1 : [4.]
g2 : [-4.]
obj : 12.0


Optimization terminated successfully    (Exit mode 0)
            Current function value: [4.8]
            Iterations: 5
            Function evaluations: 6
            Gradient evaluations: 5
Optimization Complete
-----------------------------------
minimum found at
x1 : [-0.7999999]
x2 : [-0.4000002]
x3 : [-2.]
g1 : [1.76069159e-11]
g2 : [1.02140518e-14]
minumum objective
obj : 4.799999999830984

In [2]:
```

IPython console History

Problem 3: N2 diagram



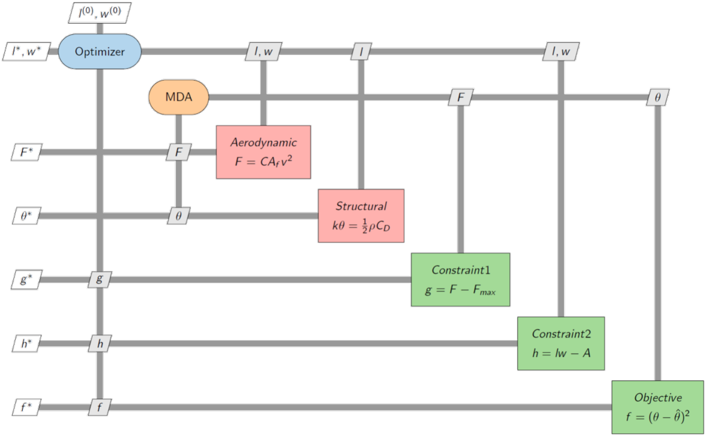
Problem 4: Airflow sensor system design

Problem formulation:

$$
\begin{aligned}
\text{Minimize} \quad & f = (\theta - \hat{\theta})^2 \\
\text{w.r.t.} \quad & \mathbf{x} = (l, w)^T \\
\text{s.t.} \quad & g : F - F_{max} \leq 0 \\
& h : lw - A = 0 \\
\text{Aerodynamic analysis :} \quad & F = CA_f v^2 \\
\text{Structural analysis :} \quad & k\theta = \frac{1}{2}\rho C_D
\end{aligned}
$$

where $C = \frac{1}{2}\rho C_d$, $A_f = lw\cos\theta$, $\rho = 1$ kg/s, $C_d = 2.0$, $\hat{\theta} = 0.250$ rad, $A = 0.01\ m^2$, $F_{max} = 7.0$ N, and $v = 40.0$ m/s

Problem 4: Airflow sensor system design



The XDSM code for this figure: Link

Problem 4: Airflow sensor system design

- Download mdo_airflow_sensor_mdf.py from Github: Link
- Part 1: Import required packages
- Part 2: Create new components for structures and aerodynamics

```python
# Part 1: Import required packages
import openmdao.api as om
import numpy as np

# Part 2: Create new components
class Structures(om.ImplicitComponent):
    """
    Structures Component
    """
    def setup(self):
        # Global Design Variable
        self.add_input('l', val=0.1)

        # Coupling parameter
        self.add_input('F', val=0.1)

        # Coupling output
        self.add_output('theta', val=0.1)

        # Finite difference all partials.
        self.declare_partials('*', '*', method='fd')

    def apply_nonlinear(self, inputs, outputs, residuals):
        """
        Evaluates theta
        """
        l = inputs['l']
        F = inputs['F']
        theta = outputs['theta']
        k = 0.05  #constant
        residuals['theta'] = k*theta - 1/2*F*l*np.cos(theta)
        # print('residuals',residuals['theta'])

class Aerodynamics(om.ExplicitComponent):
    """
    Aerodynamics Component
    """
    def setup(self):
        # Global Design Variable
        self.add_input('l', val=0.1)
```

Problem 4: Airflow sensor system design

- Part 3: Create group ProcessMDA with Newton solver

```python
# Part 3: Create group MDA
class ProcessMDA(om.Group):

    def setup(self):
        indeps = self.add_subsystem('indeps', om.IndepVarComp(), promotes=['*'])
        indeps.add_output('l', 0.01)
        indeps.add_output('w', 0.01)

        cycle = self.add_subsystem('cycle', om.Group(), promotes=['*'])
        cycle.add_subsystem('d1', Structures(), promotes_inputs=['l', 'F'], promotes_outputs=['theta'])
        cycle.add_subsystem('d2', Aerodynamics(), promotes_inputs=['l', 'w','theta'], promotes_outputs=['F'])

        ns = cycle.nonlinear_solver = om.NewtonSolver(solve_subsystems=True)
        ns.options['maxiter'] = 500

        self.add_subsystem('obj_cmp', om.ExecComp('obj = (theta - 0.250)**2'), promotes=['theta','obj'])
        self.add_subsystem('con_cmp1', om.ExecComp('con1 = F - 7'), promotes=['con1', 'F'])
        self.add_subsystem('con_cmp2', om.ExecComp('con2 = l*w - 0.01'), promotes=['con2', 'l','w'])
```

Problem 4: Airflow sensor system design

- Part 4: Build the model and problem
- Part 5: Setup the optimizer
- Part 6: Provide bounds and objective function

```python
# Part 4: Build the model and problem for optimization
prob = om.Problem()
prob.model = ProcessMDA()

# Part 5: Setup optimizer
prob.driver = om.ScipyOptimizeDriver()
prob.driver.options['optimizer'] ='SLSQP'  #'COBYLA' 'SLSQP'
prob.driver.options['maxiter'] = 100
prob.driver.options['tol'] = 1e-5
# prob.driver.options['disp'] = True

# Part 6: Provide bounds and objective function
prob.model.add_design_var('l', lower=0.01, upper=1)
prob.model.add_design_var('w', lower=0.01, upper=1)
prob.model.add_objective('obj')
prob.model.add_constraint('con1', lower=-1e-5, upper=0)
prob.model.add_constraint('con2', equals=0)

prob.setup()
prob.set_solver_print(level=0)
```

Problem 4: Airflow sensor system design

- Part 7: Run the model with initial values
- Part 8: Run optimization and print the results

```python
# Part 7: Run model with initial values
print('\nSingle evaluation')
prob['l'] = 0.1
prob['w'] = 0.1
prob.run_model()
print('l=',prob['l'])
print('w=',prob['w'])
print('theta=',prob['theta'])
print('F=',prob['F'])
print('f=',prob['obj'])
print('\n')

# Part 8: Run optimization and print outputs
prob.model.approx_totals()
prob.run_driver()
# -------------------------
print('minimum found at')
print('l=',prob['l'])
print('w=',prob['w'])
print('theta=',prob['theta'])
print('F=',prob['F'])
print('con1=',prob['con1'])
print('con2=',prob['con2'])
print('minumum objective')
print('f=',prob['obj'])
```

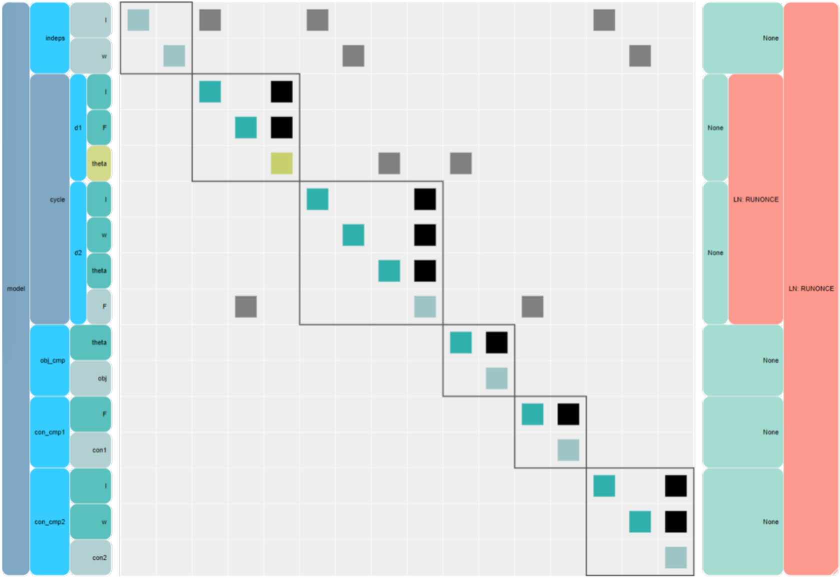
Problem 4: Airflow sensor system design

- Output

```
Single evaluation
l= [0.1]
w= [0.1]
theta= [1.28362274]
F= [4.53188304]
f= [1.06837598]


Optimization terminated successfully    (Exit mode 0)
            Current function value: [0.75338995]
            Iterations: 8
            Function evaluations: 11
            Gradient evaluations: 8
Optimization Complete
-----------------------------------
minimum found at
l= [0.03650555]
w= [0.27393115]
theta= [1.11798038]
F= [6.99999597]
con1= [-4.03007659e-06]
con2= [7.65094243e-09]
minumum objective
f= [0.75338995]

In [2]:
```

IPython console History

Problem 4: Airflow sensor system N2 diagram



Further reading

- Optimization of Paraboloid: Link
- Multidisciplinary Optimization: Link