

Optimal Design with OpenMDAO

Computational Design Laboratory

Department of Aerospace Engineering
Iowa State University

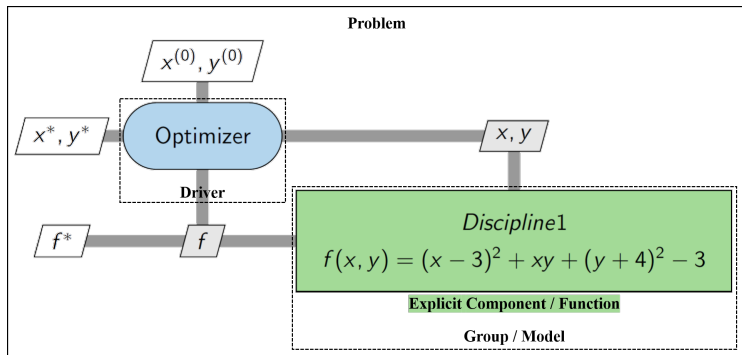
October 14, 2020

Outline

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

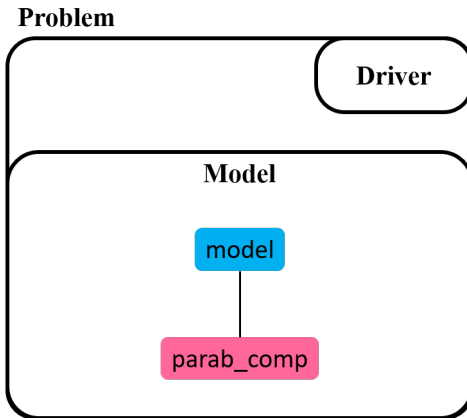
Problem 1: Single-discipline optimization

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



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

```
7 # Part 0: OpenMDAO and component imports
8 import openmdao.api as om
9
10 # Part 1: Create a new explicit components for f_xy
11 class Paraboloid(om.ExplicitComponent):
12     """
13     Evaluates the equation  $f(x,y) = (x-3)^2 + xy + (y+4)^2 - 3$ .
14     """
15
16     def setup(self):
17         self.add_input('x', val=0.0)
18         self.add_input('y', val=0.0)
19
20         self.add_output('f_xy', val=0.0)
21
22     def setup_partials(self):
23         # Finite difference all partials.
24         self.declare_partials('*', '*', method='fd')
25
26     def compute(self, inputs, outputs):
27         """
28          $f(x,y) = (x-3)^2 + xy + (y+4)^2 - 3$ 
29         Minimum at:  $x = 6.6667$ ;  $y = -7.3333$ 
30         """
31         x = inputs['x']
32         y = inputs['y']
33
34         outputs['f_xy'] = (x - 3.0)**2 + x * y + (y + 4.0)**2 - 3.0
```

Problem 1: Single-discipline optimization

```
37 if __name__ == "__main__":
38     # Part 2: Create a group and Paraboloid as subsystem of group
39     model = om.Group()
40     model.add_subsystem('parab_comp', Paraboloid())
41
42     # Part 3: Create problem from the group and setup the problem
43     prob = om.Problem(model)
44     prob.setup()
45
46     # Part 4: Provide x and y input to the problem
47     prob.set_val('parab_comp.x', 3.0)
48     prob.set_val('parab_comp.y', -4.0)
49
50     # Part 5: Run the problem
51     prob.run_model()
52
53     # Part 6: Print the input and output of the problem
54     print('x =', prob['parab_comp.x'])
55     print('y =', prob['parab_comp.y'])
56     print('f_xy =', prob.get_val('parab_comp.f_xy'))
57
58     print('\n-----\n')
59     # Part 7: Provide new input variables and print output
60     prob.set_val('parab_comp.x', 5.0)
61     prob.set_val('parab_comp.y', -2.0)
62     prob.run_model()
63     print('x =', prob['parab_comp.x'])
64     print('y =', prob['parab_comp.y'])
65     print('f_xy =', prob.get_val('parab_comp.f_xy'))
66     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

```
69 # Part 8: Build the model for optimization
70 prob = om.Problem()
71 prob.model.add_subsystem('parab', Paraboloid(), promotes_inputs=['x', 'y'])
72
73 # Part 9: Provide initial values to x and y
74 prob.model.set_input_defaults('x', 3.0)
75 prob.model.set_input_defaults('y', -4.0)
76
77 # Part 10: Setup the optimizer
78 prob.driver = om.ScipyOptimizeDriver()
79 prob.driver.options['optimizer'] = 'COBYLA'
80
81 # Part 11: Provide bounds and objective function
82 prob.model.add_design_var('x', lower=-50, upper=50)
83 prob.model.add_design_var('y', lower=-50, upper=50)
84 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

```
87 # Part 12: Setup the problem and run
88 prob.setup()
89 prob.run_driver()
90
91 # Part 13: Print the results
92 # minimum value
93 print('f_xy=', prob.get_val('parab.f_xy'))
94 # Location of the minimum
95 print('x=', prob.get_val('x'))
96 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.]
```

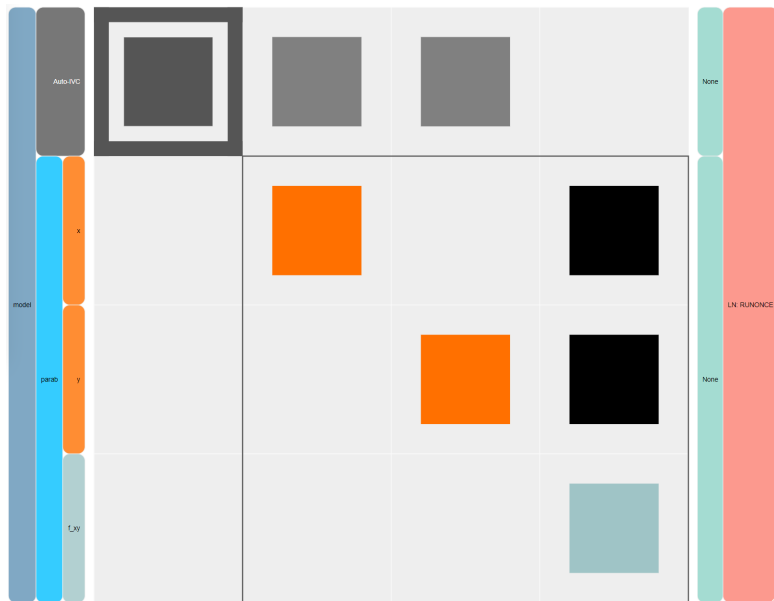
```
-----  
  
Optimization Complete
```

```
-----  
f_xy= [-27.33333333]  
x= [6.66666719]  
y= [-7.3333223]
```

```
In [2]:
```

IPython console History

Problem 1: Single-discipline N2 diagram



Problem 2: Two-discipline optimization

$$\text{Minimize } f = x^2 + z_2 + y_1 + \exp(-y_2)$$

$$\text{w.r.t. } x, z_1, z_2$$

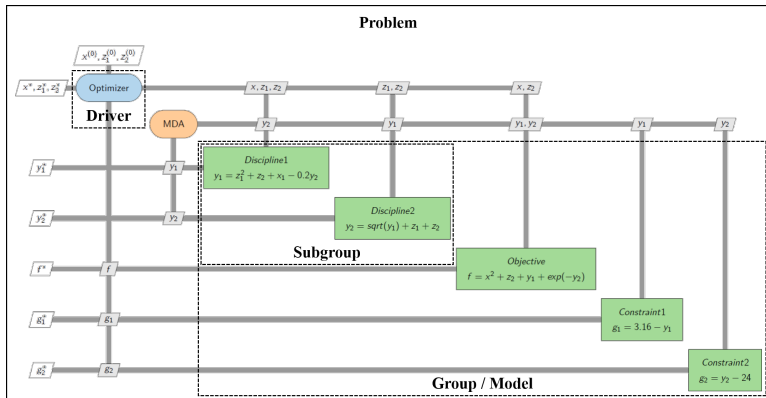
$$\text{s.t. } g_1 : 3.16 - y_1 \leq 0$$

$$g_2 : y_2 - 24.0 \leq 0$$

$$\text{Discipline 1 : } y_1 = z_1^2 + z_2 + x_1 - 0.2y_2$$

$$\text{Discipline 2 : } y_2 = \sqrt{y_1} + z_1 + z_2$$

Problem 2: Two-discipline optimization



The XDSM code for this figure: [▶ Link](#)

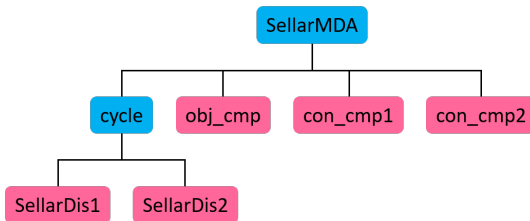
For more detailed explanation of the setup: [▶ Link](#)

Problem 2: Two-discipline optimization

Problem

Driver

Model



Problem 2: Two-discipline optimization

- Download mdo_sellar.py from Github: [▶ Link](#)
- Part 1-6: Same as mda_sellar.py

```
7 # Part 1: Import required packages
8 import openmdao.api as om
9 import numpy as np
10
11 # Part 2: Create new components for Discipline1 and 2
12 class SellarDis1(om.ExplicitComponent):
13     """
14     Component containing Discipline 1 -- no derivatives version.
15     """
16     def setup(self):
17         # Global Design Variable
18         self.add_input('z', val=np.zeros(2))
19
20         # Local Design Variable
21         self.add_input('x', val=0.)
22
23         # Coupling parameter
24         self.add_input('y2', val=1.0)
25
26         # Coupling output
27         self.add_output('y1', val=1.0)
28
29         # Finite difference all partials.
30         self.declare_partials('*', '*', method='fd')
31
32     def compute(self, inputs, outputs):
33         """
34         Evaluates the equation
35          $y1 = z1^2 + z2 + x1 - 0.2*y2$ 
36         """
37         z1 = inputs['z'][0]
38         z2 = inputs['z'][1]
39         x1 = inputs['x']
40         y2 = inputs['y2']
41
42         outputs['y1'] = z1**2 + z2 + x1 - 0.2*y2
43
44 class SellarDis2(om.ExplicitComponent):
45     """
46     Component containing Discipline 2 -- no derivatives version.
47     """
```

Problem 2: Two-discipline optimization

```
48 def setup(self):
49     # Global Design Variable
50     self.add_input('z', val=np.zeros(2))
51
52     # Coupling parameter
53     self.add_input('y1', val=1.0)
54
55     # Coupling output
56     self.add_output('y2', val=1.0)
57
58     # Finite difference all partials.
59     self.declare_partials('*', '*', method='fd')
60
61 def compute(self, inputs, outputs):
62     """
63     Evaluates the equation
64     y2 = y1*(.5) + z1 + z2
65     """
66     z1 = inputs['z'][0]
67     z2 = inputs['z'][1]
68     y1 = inputs['y1']
69
70     # Note: this may cause some issues. However, y1 is constrained to be
71     # above 3.16, so lets just let it converge, and the optimizer will
72     # throw it out
73     if y1.real < 0.0:
74         y1 *= -1
75
76     outputs['y2'] = y1*.5 + z1 + z2
77
78 # Part 3: Create group SellarMDA
79 class SellarMDA(om.Group):
80     """
81     Group containing the Sellar MDA.
82     """
83     def setup(self):
84         indeps = self.add_subsystem('indeps', om.IndepVarComp(), promotes=['*'])
85         indeps.add_output('x', 1.0)
86         indeps.add_output('z', np.array([5.0, 2.0]))
87
88         cycle = self.add_subsystem('cycle', om.Group(), promotes=['*'])
```

Problem 2: Two-discipline optimization

```
89     cycle.add_subsystem('d1', SellarDis1(), promotes_inputs=['x', 'z', 'y2'],
90                        promotes_outputs=['y1'])
91     cycle.add_subsystem('d2', SellarDis2(), promotes_inputs=['z', 'y1'],
92                        promotes_outputs=['y2'])
93
94     # Nonlinear Block Gauss Seidel is a gradient free solver
95     cycle.nonlinear_solver = om.NonlinearBlockGS(iprint=1) # try iprint=2
96
97     self.add_subsystem('obj_cmp', om.ExecComp('obj = x**2 + z[1] + y1 + exp(-y2)',
98                        z=np.array([0.0, 0.0]), x=0.0),
99                        promotes=['x', 'z', 'y1', 'y2', 'obj'])
100
101     self.add_subsystem('con_cmp1', om.ExecComp('con1 = 3.16 - y1'), promotes=['con1', 'y1'])
102     self.add_subsystem('con_cmp2', om.ExecComp('con2 = y2 - 24.0'), promotes=['con2', 'y2'])
103
104
105 # Part 4: Setup model and problem
106 prob = om.Problem()
107 prob.model = SellarMDA()
108 prob.setup()
109
110 # Part 5: Provide input to the problem
111 prob['x'] = 2.
112 prob['z'] = [-1., -1.]
113
114 prob.run_model()
115
116 # Part 6: print details
117 print('\nInput ----')
118 print('x :',prob['x'])
119 print('z1 :',prob['z'][0])
120 print('z2 :',prob['z'][1])
121
122 print('\nDiscipline output ----')
123 print('y1 :',prob['y1'])
124 print('y2 :',prob['y2'])
125
126 print('\nObjective and constraints----')
127 print('obj :',prob['obj'])
128 print('con1 :',prob['con1'])
129 print('con2 :',prob['con2'])
130 print('\n')
```


Problem 2: Two-discipline optimization

- Part 7: Setup the optimization and print output

```
132# Part 7: Optimizing the Problem
133prob.driver = om.ScipyOptimizeDriver()
134prob.driver.options['optimizer'] = 'SLSQP'
135# prob.driver.options['maxiter'] = 100
136prob.driver.options['tol'] = 1e-8
137
138prob.model.add_design_var('x', lower=0, upper=10)
139prob.model.add_design_var('z', lower=0, upper=10)
140prob.model.add_objective('obj')
141prob.model.add_constraint('con1', upper=0)
142prob.model.add_constraint('con2', upper=0)
143
144# Ask OpenMDAO to finite-difference across the model to compute the gradients for the optimizer
145prob.model.approx_totals()
146
147prob.setup()
148prob.set_solver_print(level=0)
149
150prob.run_driver()
151
152print('\nminimum found at')
153print('x :',prob.get_val('x')[0])
154print('z1 :',prob.get_val('z')[0])
155print('z2 :',prob.get_val('z')[1])
156
157print('')
158print('y1 :',prob.get_val('y1'))
159print('y2 :',prob.get_val('y2'))
160
161print('\nminimum objective and constraints')
162print('obj :',prob.get_val('obj')[0])
163print('con1 :',prob.get_val('con1'))
164print('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

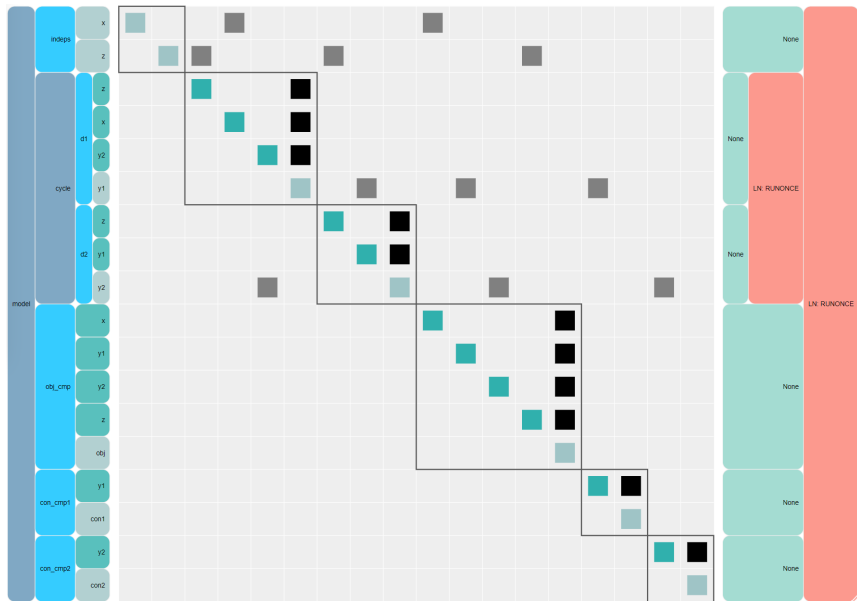
y1 : [3.16]
y2 : [3.75527777]

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:

$$\text{Minimize } f = x_1^2 + x_2^2 + x_3^2$$

$$\text{w.r.t. } x_1, x_2, x_3$$

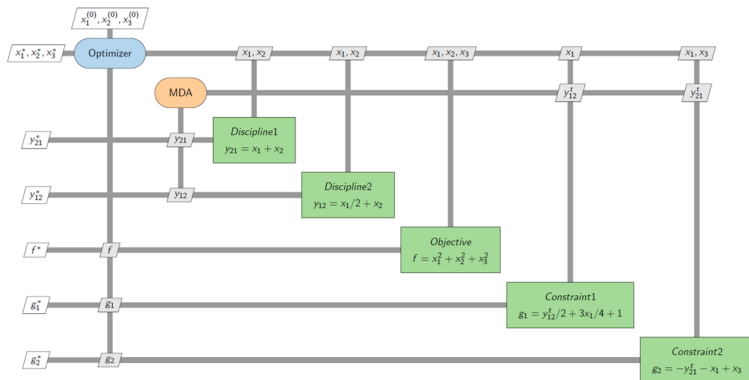
$$\text{s.t. } g_1 : \frac{y_{12}^t}{2} + \frac{3x_1}{4} + 1 \leq 0$$

$$g_2 : -y_{21}^t - x_1 + x_3 \leq 0$$

$$\text{Discipline 1 : } y_{21} = x_1 + x_2$$

$$\text{Discipline 2 : } y_{12} = \frac{x_1}{2} + x_2$$

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

```
7 # Part 1: Import required packages
8 import openmdao.api as om
9
10 # Part 2: Create new components for Analysis1 and 2
11 class Analysis1(om.ExplicitComponent):
12     """
13     Component containing Discipline1 and Constraint1
14     """
15     def setup(self):
16         # Global Design Variable
17         self.add_input('x1', val=0.0)
18         self.add_input('x2', val=0.0)
19
20         # Coupling parameter
21         self.add_input('y12', val=1.0)
22
23         # Coupling output
24         self.add_output('y21', val=1.0)
25         self.add_output('g1', val=1.0)
26
27         # Finite difference all partials.
28         self.declare_partials('*', '*', method='fd')
29
30     def compute(self, inputs, outputs):
31         """
32         Evaluate y21, g1
33         """
34         x1 = inputs['x1']
35         x2 = inputs['x2']
36         y12 = inputs['y12']
37
38         outputs['y21'] = x1 + x2
39         outputs['g1'] = y12/2 + 3*x1/4 + 1
40
41 class Analysis2(om.ExplicitComponent):
42     """
43     Component containing Discipline2 and Constraint2
44     """
45     def setup(self):
46         # Global Design Variable
47         self.add_input('x1', val=0.0)
```

Problem 3: Two-discipline optimization

- Part 3: Create group ProcessMDA

```
73 # Part 3: Create group MDA
74 class ProcessMDA(om.Group):
75     """
76     Group containing MDA
77     """
78     def setup(self):
79         indeps = self.add_subsystem('indeps', om.IndepVarComp(), promotes=['*'])
80         indeps.add_output('x1', 1.0)
81         indeps.add_output('x2', 1.0)
82         indeps.add_output('x3', 1.0)
83
84         cycle = self.add_subsystem('cycle', om.Group(), promotes=['*'])
85         cycle.add_subsystem('d1', Analysis1(), promotes_inputs=['x1', 'x2', 'y12'], promotes_outputs=['y21', 'g1'])
86         cycle.add_subsystem('d2', Analysis2(), promotes_inputs=['x1', 'x2', 'x3', 'y21'], promotes_outputs=['y12', 'g2'])
87
88         # Nonlinear Block Gauss Seidel is a gradient free solver
89         cycle.nonlinear_solver = om.NonlinearBlockGS()
90
91         self.add_subsystem('obj_cmp', om.ExecComp('obj = x1**2 + x2**2 + x3**2',
92                                                    x1=0.0, x2=0.0, x3=0.0),
93                           promotes=['x1', 'x2', 'x3', 'obj'])
94
95         self.add_subsystem('con_cmp1', om.ExecComp('con1 = g1'), promotes=['con1', 'g1'])
96         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

```
99 # Part 4: Build the model and problem for optimization
100 prob = om.Problem()
101 prob.model = ProcessMDA()
102
103 # Part 5: Setup optimizer
104 prob.driver = om.ScipyOptimizeDriver()
105 prob.driver.options['optimizer'] = 'SLSQP'
106 # prob.driver.options['maxiter'] = 100
107 prob.driver.options['tol'] = 1e-8
108
109 # Part 6: Provide bounds and objective function
110 prob.model.add_design_var('x1', lower=-4, upper=4)
111 prob.model.add_design_var('x2', lower=-4, upper=4)
112 prob.model.add_design_var('x3', lower=-4, upper=4)
113 prob.model.add_objective('obj')
114 prob.model.add_constraint('con1', upper=0)
115 prob.model.add_constraint('con2', upper=0)
116
117 prob.setup()
118 prob.set_solver_print(level=0)
119
```


Problem 3: Two-discipline optimization

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

```
121 # Part 7: Run model with initial values
122 print('\nSingle evaluation')
123 prob['x1'] = 2.
124 prob['x2'] = 2.
125 prob['x3'] = 2.
126 prob.run_model()
127 print('x1 :',prob['x1'])
128 print('x2 :',prob['x2'])
129 print('x3 :',prob['x3'])
130 print('g1 :',prob['g1'])
131 print('g2 :',prob['g2'])
132 print('obj :',prob['obj'][0])
133 print('\n')
134
135 # Part 8: Run optimization and print outputs
136 # Ask OpenMDAO to finite-difference across the model to compute the gradients for the optimizer
137 prob.model.approx_totals()
138 prob.run_driver()
139 # -----
140 print('minimum found at')
141 print('x1 :',prob['x1'])
142 print('x2 :',prob['x2'])
143 print('x3 :',prob['x3'])
144 print('g1 :',prob['g1'])
145 print('g2 :',prob['g2'])
146 print('minimum objective')
147 print('obj :',prob['obj'][0])
```

Problem 3: Two-discipline optimization

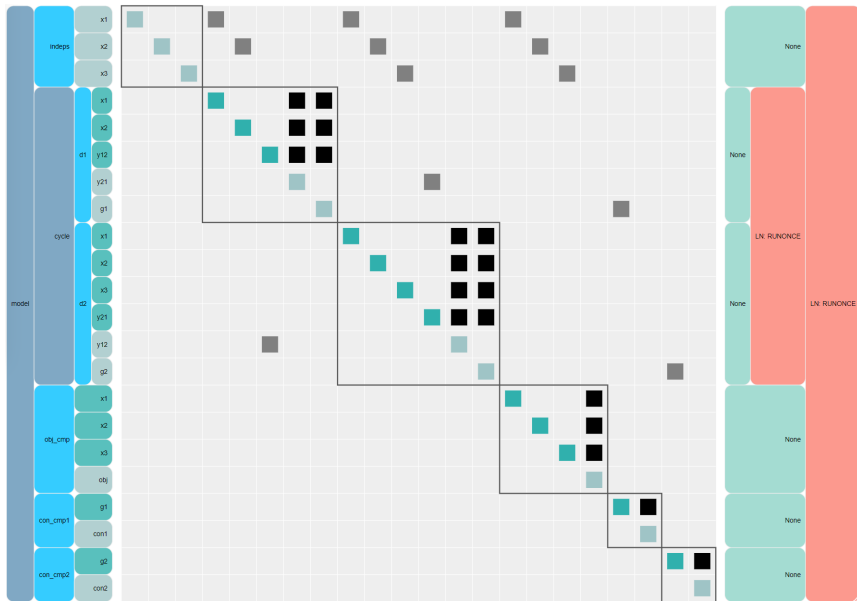
- Output

```
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]:
```

Problem 3: N2 diagram



Problem 4: Airflow sensor system design

Problem formulation:

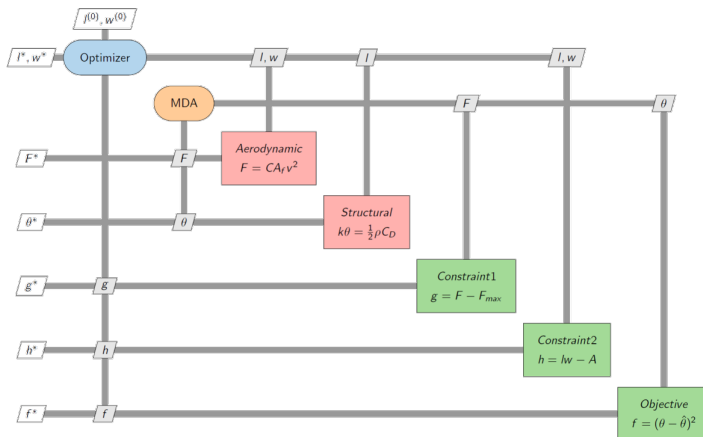
$$\begin{aligned} \text{Minimize} \quad & f = (\theta - \hat{\theta})^2 \\ \text{w.r.t.} \quad & \mathbf{x} = (l, w)^T \\ \text{s.t.} \quad & g : F - F_{\max} \leq 0 \\ & h : lw - A = 0 \end{aligned}$$

$$\text{Aerodynamic analysis : } F = CA_f v^2$$

$$\text{Structural analysis : } k\theta = \frac{1}{2}\rho C_D$$

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

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

```
7 # Part 1: Import required packages
8 import openmdao.api as om
9 import numpy as np
10
11 # Part 2: Create new components
12 class Structures(om.ImplicitComponent):
13     """
14     Structures Component
15     """
16     def setup(self):
17         # Global Design Variable
18         self.add_input('l', val=0.1)
19
20         # Coupling parameter
21         self.add_input('F', val=0.1)
22
23         # Coupling output
24         self.add_output('theta', val=0.1)
25
26         # Finite difference all partials.
27         self.declare_partials('*', '*', method='fd')
28
29     def apply_nonlinear(self, inputs, outputs, residuals):
30         """
31         Evaluates theta
32         """
33         l = inputs['l']
34         F = inputs['F']
35         theta = outputs['theta']
36         k = 0.05 #constant
37         residuals['theta'] = k*theta - 1/2*F*1*np.cos(theta)
38         # print("residuals",residuals['theta'])
39
40 class Aerodynamics(om.ExplicitComponent):
41     """
42     Aerodynamics Component
43     """
44     def setup(self):
45         # Global Design Variable
46         self.add_input('l', val=0.1)
```

Problem 4: Airflow sensor system design

- Part 3: Create group ProcessMDA with Newton solver

```
73 # Part 3: Create group MDA
74 class ProcessMDA(om.Group):
75
76     def setup(self):
77         indeps = self.add_subsystem('indeps', om.IndepVarComp(), promotes=['*'])
78         indeps.add_output('l', 0.01)
79         indeps.add_output('w', 0.01)
80
81         cycle = self.add_subsystem('cycle', om.Group(), promotes=['*'])
82         cycle.add_subsystem('d1', Structures(), promotes_inputs=['l', 'F'], promotes_outputs=['theta'])
83         cycle.add_subsystem('d2', Aerodynamics(), promotes_inputs=['l', 'w', 'theta'], promotes_outputs=['F'])
84
85         ns = cycle.nonlinear_solver = om.NewtonSolver(solve_subsystems=True)
86         ns.options['maxiter'] = 500
87
88         self.add_subsystem('obj_cmp', om.ExecComp('obj = (theta - 0.250)**2'), promotes=['theta', 'obj'])
89         self.add_subsystem('con_cmp1', om.ExecComp('con1 = F - 7'), promotes=['con1', 'F'])
90         self.add_subsystem('con_cmp2', om.ExecComp('con2 = 1*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

```
92 # Part 4: Build the model and problem for optimization
93 prob = om.Problem()
94 prob.model = ProcessMDA()
95
96 # Part 5: Setup optimizer
97 prob.driver = om.ScipyOptimizeDriver()
98 prob.driver.options['optimizer'] = 'SLSQP' # 'COBYLA' 'SLSQP'
99 prob.driver.options['maxiter'] = 100
100 prob.driver.options['tol'] = 1e-5
101 # prob.driver.options['disp'] = True
102
103 # Part 6: Provide bounds and objective function
104 prob.model.add_design_var('l', lower=0.01, upper=1)
105 prob.model.add_design_var('w', lower=0.01, upper=1)
106 prob.model.add_objective('obj')
107 prob.model.add_constraint('con1', lower=-1e-5, upper=0)
108 prob.model.add_constraint('con2', equals=0)
109
110 prob.setup()
111 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

```
114 # Part 7: Run model with initial values
115 print('\nSingle evaluation')
116 prob['l'] = 0.1
117 prob['w'] = 0.1
118 prob.run_model()
119 print('l=',prob['l'])
120 print('w=',prob['w'])
121 print('theta=',prob['theta'])
122 print('F=',prob['F'])
123 print('f=',prob['obj'])
124 print('\n')
125
126 # Part 8: Run optimization and print outputs
127 prob.model.approx_totals()
128 prob.run_driver()
129 # -----
130 print('minimum found at')
131 print('l=',prob['l'])
132 print('w=',prob['w'])
133 print('theta=',prob['theta'])
134 print('F=',prob['F'])
135 print('con1=',prob['con1'])
136 print('con2=',prob['con2'])
137 print('minimum objective')
138 print('f=',prob['obj'])
```

Problem 4: Airflow sensor system design

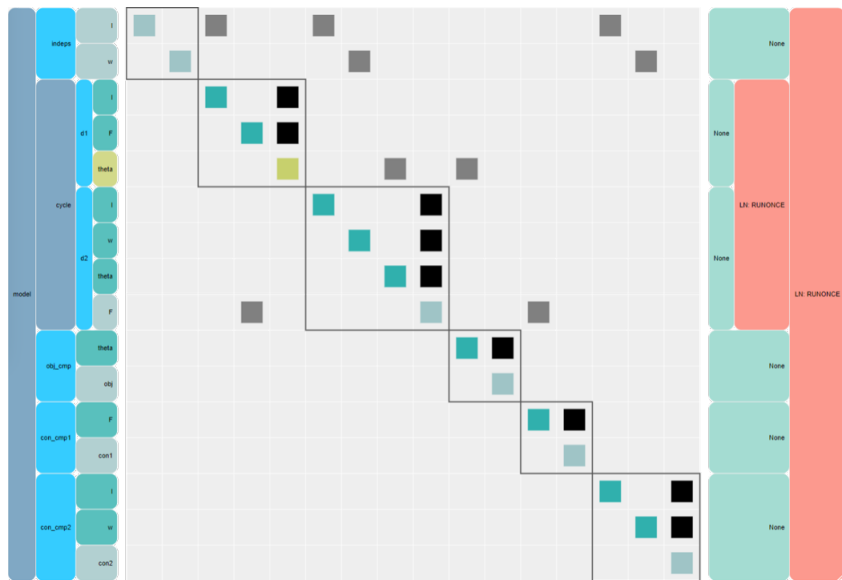
- Output

```
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]:
```

Problem 4: Airflow sensor system N2 diagram



Further reading

- Optimization of Paraboloid: [▶ Link](#)
- Multidisciplinary Optimization: [▶ Link](#)