Conceptual Aircraft Design in OpenMDAO

Eytan Adler

2022 OpenMDAO Workshop
NASA Glenn
October 24–25th
Conceptual design is multidisciplinary

- Propulsion
- Aerodynamics
- Weight
- Thermal
- Cost?
Who is “we”? 

Ben Brelje
Hybrid electric aircraft design optimization

minimize: fuel burn + 0.01MTOW

by varying:

- MTOW
- $S_{ef}$
- $d_{prop}$
- $W_{battery}$
- $P_{motor}$ (rated)
- $P_{turboshift}$ (rated)
- $P_{generator}$ (rated)
- $H_E$ (degree of hybridization w.r.t energy)

subject to scalar constraints:

- $R_{turb} = W_{turb} - W_{motor} - W_{battery} - W_{fuel} \geq 0$
- $R_{fuel} = W_{fuel} - W_{fact} \geq 0$
- $BFL \leq 4452$ ft (no worse than baseline)
- $V_{stab} \leq 81.6$ kt (no worse than baseline)

and vector constraints:

- $\vec{F}_{turb} \leq 1.05 \vec{F}_{motor}$ (rated)
- $\vec{F}_{turboshift} \leq \vec{F}_{turboshift}$ (rated)
- $\vec{F}_{generator} \leq \vec{F}_{generator}$ (rated)
- $\vec{F}_{battery} \leq \vec{F}_{battery} \cdot P_f$
But we can do a lot more…

Start with a parallel hybrid turbofan

Battery powers the electric motor

Battery powers the electric motor

Cool the battery with a refrigerator

Refrigerator dumps heat into freestream

Same for the electric motor

...and the fault protection

And there is still more complexity

• All in a mission analysis
• Battery and motor can accumulate heat (not assuming steady state)
• Chiller bypass, variable exit duct, and engine hybrid fraction can be controlled during the mission
After running a mission, we can...
...analyze component temperatures
...and optimize duct area in time
Code flexibility
Mission analysis
Lessons learned
Code flexibility
Mission analysis
Lessons learned
Flexible

- Use on a wide range of problems
- Steeper learning curve
- Longer case setup time

Easy to use

- Well-defined interfaces
- Easier to learn
- Simpler setup
Flexible

OpenConcept

Easy to use
Aircraft model only has a few requirements

- Computes drag, thrust, and weight from $C_L$ and throttle
- Atmospheric and flight conditions available as inputs
Must compute drag and thrust

```python
class Aircraft(om.Group):
    def setup(self):
        self.add_subsystem("aero", AerodynamicModel())  # compute drag using lift coefficient
        self.add_subsystem("prop", PropulsionModel())  # compute thrust using throttle

        intfuel = self.add_subsystem("intfuel",
            Integrator(num_nodes=21, method="simpson", diff_units="s", time_setup="duration"),
        )
        intfuel.add_integrand("fuel_used", rate_name="fuel_flow", units="kg")
        self.connect("prop.fuel_flow", "intfuel.fuel_flow")

        self.add_subsystem("weight", WeightModel())  # compute weight
```

- Computes drag, thrust, and weight from $C_L$ and throttle
- Atmospheric and flight conditions available as inputs
Must compute weight

- Computes drag, thrust, and weight from $C_L$ and throttle
- Atmospheric and flight conditions available as inputs
Can use OpenConcept’s integrator for fuel

- Computes drag, thrust, and weight from $C_L$ and throttle
- Atmospheric and flight conditions available as inputs
class Mission(om.Group):
    
def setup(self):
        # Define variables from airplane data file
        ac_vars = self.add_subsystem("ac_vars", DictIndepVarComp(acdata), promotes_outputs=["*"],)
        ac_vars.add_output_from_dict("ac|aero|polar|e")
        ac_vars.add_output_from_dict("ac|aero|polar|CD0")
        ac_vars.add_output_from_dict("ac|geom|wing|S_ref")
        ac_vars.add_output_from_dict("ac|geom|wing|AR")

        # Run a full mission analysis including takeoff, climb, cruise, and descent
        self.add_subsystem("mission_analysis",
            FullMissionAnalysis(num_nodes=21, aircraft_model=Aircraft),
            promotes_inputs=["*"],
            promotes_outputs=["*"],
        )
class Mission(om.Group):
    def setup(self):
        # Define variables from airplane data file
        ac_vars = self.add_subsystem("ac_vars", DictIndepVarComp(acdata), promotes_outputs=['*'])
        ac_vars.add_output_from_dict("ac|aero|polar|e")
        ac_vars.add_output_from_dict("ac|aero|polar|CD0")
        ac_vars.add_output_from_dict("ac|geom|wing|S_ref")
        ac_vars.add_output_from_dict("ac|geom|wing|AR")

        # Run a full mission analysis including takeoff, climb, cruise, and descent
        self.add_subsystem("mission_analysis",
            FullMissionAnalysis(num_nodes=21, aircraft_model=Aircraft),
            promotes_inputs=['*'],
            promotes_outputs=['*'],
        )
Add mission analysis group

```python
class Mission(om.Group):
    def setup(self):
        # Define variables from airplane data file
        ac_vars = self.add_subsystem("ac_vars", DictIndepVarComp(acdata), promotes_outputs=["*"],
        ac_vars.add_output_from_dict("ac|aero|polar|e")
        ac_vars.add_output_from_dict("ac|aero|polar|CD0")
        ac_vars.add_output_from_dict("ac|geom|wing|S_ref")
        ac_vars.add_output_from_dict("ac|geom|wing|AR")

        # Run a full mission analysis including takeoff, climb, cruise, and descent
        self.add_subsystem("mission_analysis",
            FullMissionAnalysis(num_nodes=21, aircraft_model=Aircraft),
            promotes_inputs=["*"],
            promotes_outputs=["*"],
        )
```
What is mission analysis?

- Simulate aircraft flying mission while satisfying physics
- Determine fuel burn
- Analyze component temperatures, hydrogen boil off, etc.
We assume steady flight at integration points

• This avoids working directly with the equations of motion

• Validated against real world data from Pipistrel

Basic mission setup

Climb phase
- airspeed
- vertical speed
- cruise altitude

Cruise phase
- $h_{\text{climb, final}}$
- $\text{range}_{\text{climb, final}}$

Descent phase
- $h_{\text{cruise, final}}$
- $\text{range}_{\text{cruise, final}}$

Mission range
- airspeed
- vertical speed

Range descent, final
- $\text{range}_{\text{descent, final}}$
Basic mission setup

Climb phase
- airspeed
- vertical speed
- cruise altitude

Cruise phase
- \( h_{climb, \text{final}}, \text{range}_{climb, \text{final}} \)

Descent phase
- \( h_{cruise, \text{final}}, \text{range}_{cruise, \text{final}} \)

- \( \text{range}_{\text{descent, final}} \)
Basic mission setup

- **Climb phase**
  - airspeed
  - vertical speed
  - cruise altitude

- **Cruise phase**
  - airspeed
  - vertical speed
  - mission range

- **Descent phase**
  - airspeed
  - vertical speed

- \( h_{\text{climb, final}}, \text{range}_{\text{climb, final}} \)
- \( h_{\text{cruise, final}}, \text{range}_{\text{cruise, final}} \)
- \( \text{range}_{\text{descent, final}} \)
Basic mission setup

- **Climb phase**
  - airspeed
  - vertical speed
  - cruise altitude

- **Cruise phase**
  - airspeed
  - vertical speed
  - mission range

- **Descent phase**
  - airspeed
  - vertical speed

- **Range**:
  - $h_{\text{climb, final}}$, $r\text{ange}_{\text{climb, final}}$
  - $h_{\text{cruise, final}}$, $r\text{ange}_{\text{cruise, final}}$
  - $r\text{ange}_{\text{descent, final}}$
Solving an individual flight phase (climb)
Integrator solves for altitude and range

Newton solver

Integrator

airspeed
vertival speed
$h_{initial}$, range$_{initial}$

t$_{climb}$

$h_{final}$, range$_{final}$

altitude residual

$h_{final} - h_{cruise}$

cruise altitude

$h_{final}$

$h$

Atmospherics
flight conditions

throttle, $C_L$

$C_L$

Aircraft model

thrust, weight, drag

Force residual

net forces
Integrator uses Simpson’s rule
Integrator uses Simpson’s rule
Integrator uses Simpson’s rule
Integrator uses Simpson’s rule
Integrator uses Simpson’s rule
Integrator uses Simpson’s rule
Set phase duration so it ends at cruise altitude
Atmospherics computes flight conditions

- Airspeed
- Vertical speed
- \( h_{\text{initial}}, \text{range}_{\text{initial}} \)
- Cruise altitude
- \( t_{\text{climb}} \)
- \( h_{\text{final}}, \text{range}_{\text{final}} \)
- Altitude residual
- \( h_{\text{final}} - h_{\text{cruise}} \)
- Integrator
- \( h_{\text{final}} \)
- \( h \)
- Newton solver
- Flight conditions
- Force residual
- Aircraft model
- Thrust, weight, drag
- Net forces
- Throttle, \( C_L \)
- \( C_L \)
Aircraft model computes forces

- Aircraft model
- t_{climb}
- h_{final}, \text{range}_{final}
- altitude residual
- h_{final} - h_{cruise}
- h
- Integrator
- Newton solver
- Airspeed
- Vertical speed
- h_{initial}, \text{range}_{initial}
- Cruise altitude
- Throttle, C_L
- C_L
- Atmospheric
- Flight conditions
- Thrust, weight, drag
- Force residual
- Net forces
Net forces driven to zero by Newton solver
We usually use a single Newton solver
The integrator is also used elsewhere

• Integrate fuel flow to compute fuel burn

• Integrate heat flows to compute component temperature

• Brelje magic™ automatically finds states within aircraft model to integrate and links them across mission phases

```python
intfuel = self.add_subsystem(  
    "intfuel",  
    Integrator(num_nodes=21, method="simpson", diff_units="s", time_setup="duration"),  
)
intfuel.add_integrand("fuel_used", rate_name="fuel_flow", units="kg")
```
Code flexibility
Mission analysis
Lessons learned
Not designed for trajectory optimization

Optimize cruise airspeed and vertical speed
We linearly interpolate within phases
We linearly interpolate within phases

- Takeoff
- Climb
- Cruise
- Descent
class SimpleMotor(om.ExplicitComponent):

def initialize(self):
    self.options.declare("num_nodes", default=11)
    self.options.declare("efficiency", default=0.95)

def setup(self):
    nn = self.options["num_nodes"]

    self.add_input("throttle", shape=(nn,))
    self.add_input("elec_power_rating", units="W")

    self.add_output("shaft_power_out", units="W", shape=(nn,))
    self.add_output("elec_load", units="W", shape=(nn,))

    # declare sparse partials here

def compute(self, inputs, outputs):
    outputs["shaft_power_out"] = inputs["throttle"] * inputs["elec_power_rating"] * self.options["efficiency"]
    outputs["elec_load"] = inputs["throttle"] * inputs["elec_power_rating"]
Propulsion modeling with surrogates

• Detailed turbomachinery and propulsion models use offline surrogates (often pyCycle)
• Avoids challenges with robustness and cost
What is best for mission analysis?

• OpenConcept’s approach
  • Fast
  • Robust-ish
  • Physics valid once Newton solver converges (no optimization required)

• Trajectory optimization-style approach
  • Not as robust
  • More general representation of mission profile
  • Better for mission profile optimization
We developed a tool for efficient conceptual aircraft design with OpenMDAO

Rapidly optimize aircraft architectures

Steady flight mission phases with predefined profile

Hybrid mission analysis approach for the best of both?
github.com/mdolab/openconcept