National Aeronautics and Space Administration

Aviary: A Transformational Tool For Aircraft Design

Carl Recine, ARC, Aviary Co-Lead

Jason Kirk, LaRC, Aviary Lead

Eliot Aretskin-Hariton, Aviary Feature Developer

Presentation Roadmap

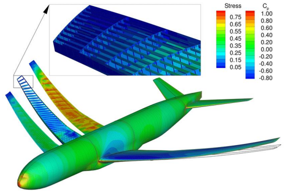
1. What is Aviary?
2. What challenges does Aviary solve?
3. New features and upcoming additions
4. Walkthrough using Aviary

Aviary: An Open-Source Tool for Next-Gen Tool for Aircraft Design

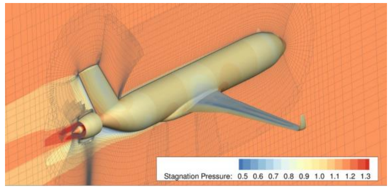
Challenge

- Existing conceptual design tools limited in capability to model and determine trends with advanced configurations and new technologies
- Need for better analysis & optimization techniques to produce more performant and robust designs

advanced concepts



analysis of coupled subsystems



Unified Aviary Model

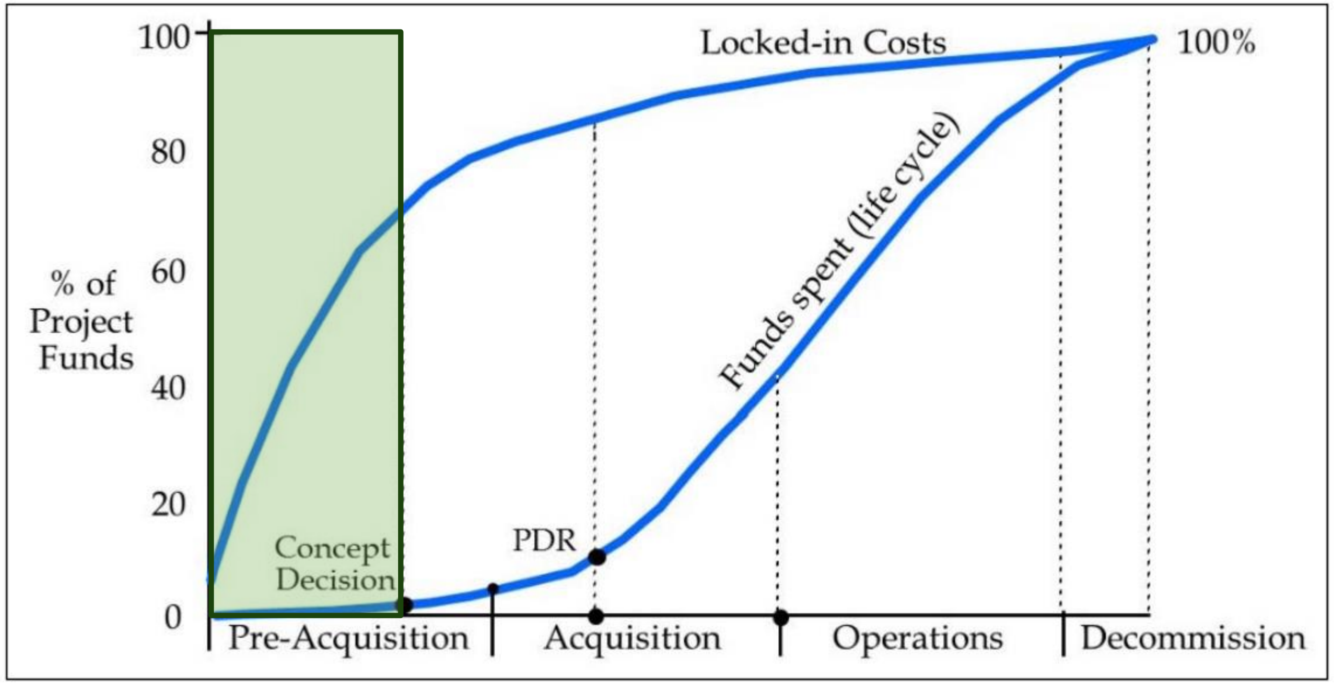
Objectives

- Maintain capabilities of legacy tools
- Support electrified vehicles out-of-the-box
- Easy integration of external models
- Use state-of-the-art analysis and optimization techniques, with a focus on gradient-based optimization
- Open-source release, maximize partner collaboration

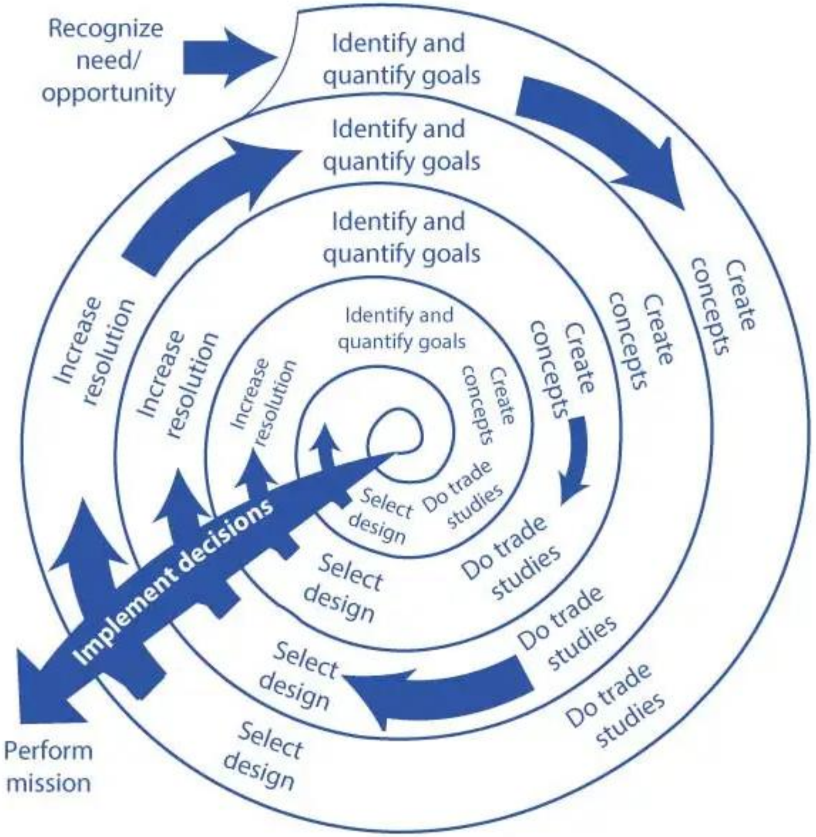
Presentation Roadmap

1. What is Aviary?
2. What challenges does Aviary solve?
3. New features and upcoming additions
4. Walkthrough using Aviary

Conceptual Design: An Iterative Process



- Iteration is key – time and resources are limited
- Quantity of analysis allows exploration of more of the design space
- Quality of analysis reduces risk on key decisions

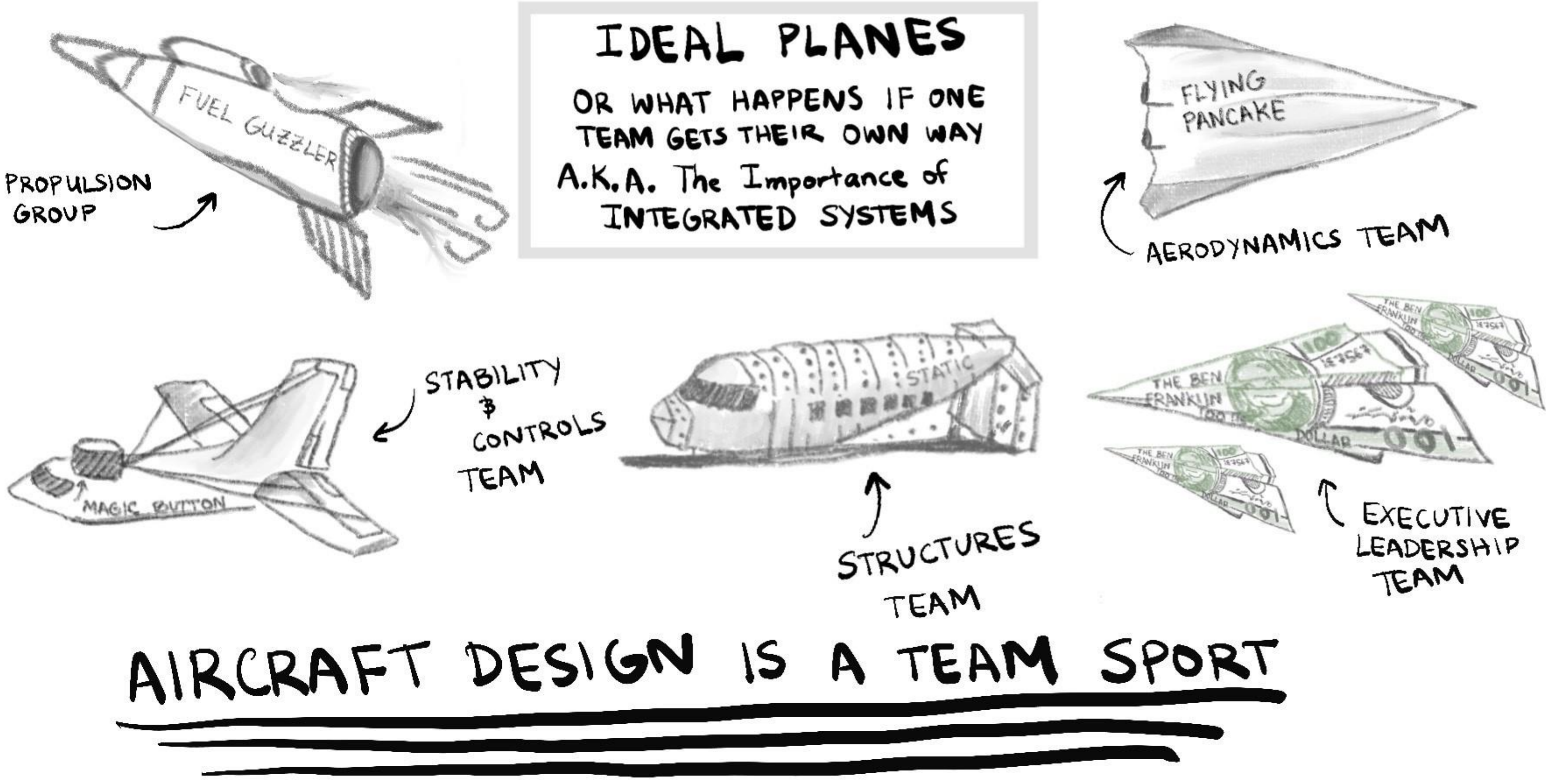


Conceptual design process has the most design freedom
Key decisions must be made before “locked-in”
For NASA Aeronautics:

- Which vehicle configurations are worth pursuing?
- What technologies buy their way onto a vehicle?
- What are the projected performance and cost?

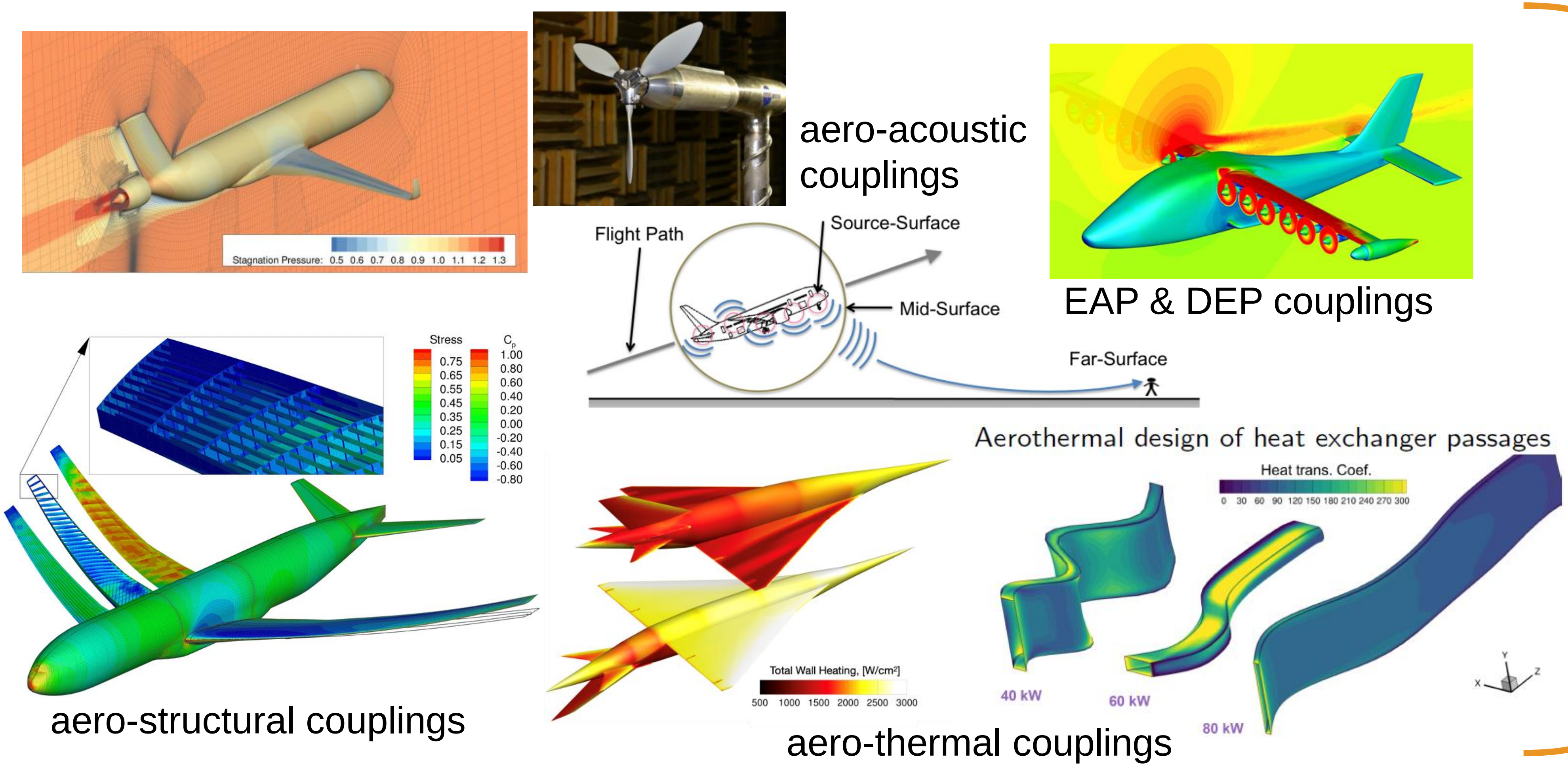
Aircraft Design

- Always a compromise between competing disciplines
  - Structures
  - Aerodynamics
  - Propulsion
  - Controls
  - etc...
- Discipline experts must work together and compromise on a design – this takes time!

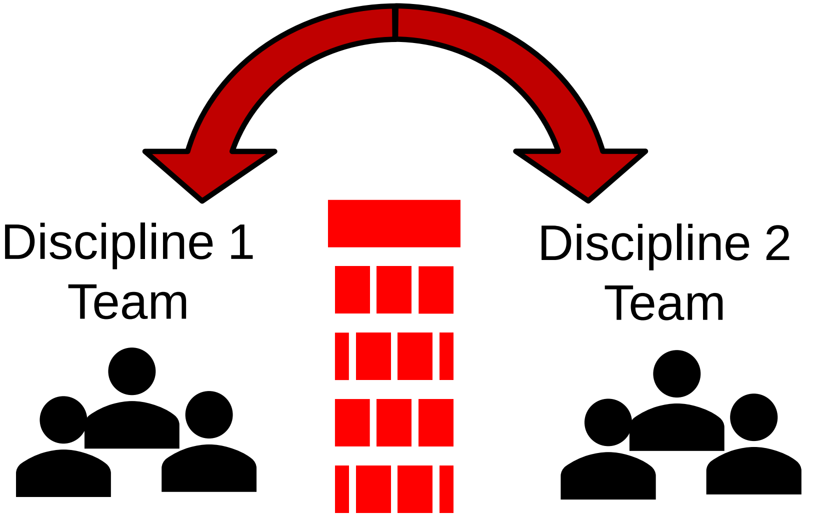


Complex Coupling Between Disciplines

Aircraft system-level metrics are dependent on interactions between technologies

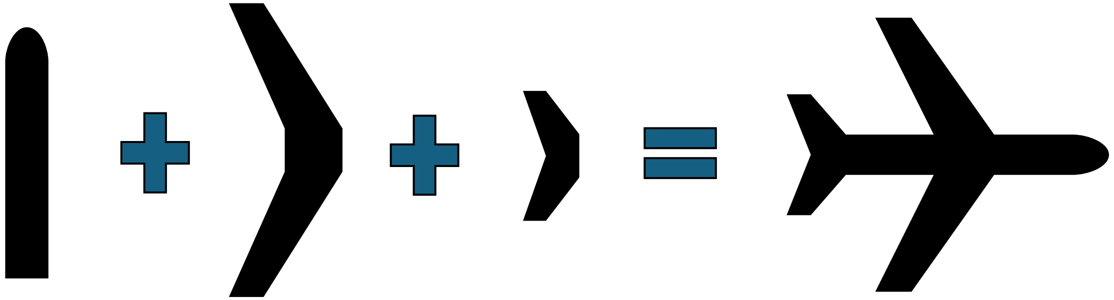


Historical Aircraft Design Method



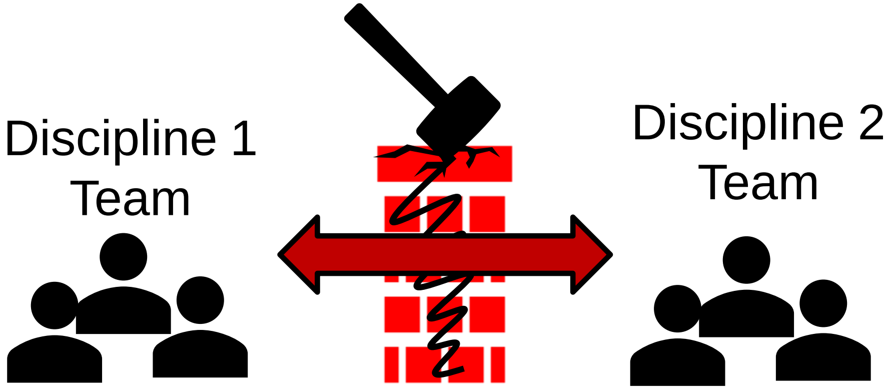
- Teams conduct analysis independently
- Data thrown “over-the-wall”
- Manual iteration required

Aircraft is treated as the sum of its parts



Old method too slow to rapidly iterate
Complex interactions missed, suboptimal design

Advanced Digital Design Method



- Discipline analysis connected through software
- No “wall”, teams work on the same aircraft level model
- Version control for models
- No manual iteration needed

Complex interactions are captured, aircraft is designed holistically

Aviary enables this conceptual aircraft design workflow

Aviary is NASA's next-generation conceptual aircraft modeling and optimization framework

Aviary's goals are to:

- Combine the capabilities of legacy codes into a single tool
- Use a modern code architecture, built in Python, that enables easy modification of source code and modular analysis that can be easily swapped with custom tools or methods
- Perform state-of-the-art coupled optimization with openMDAO, dymos
- Provide capability to analyze advanced future aircraft concepts, including electrification
- Open-source release for maximum impact to the aerospace research community

Combine legacy tools

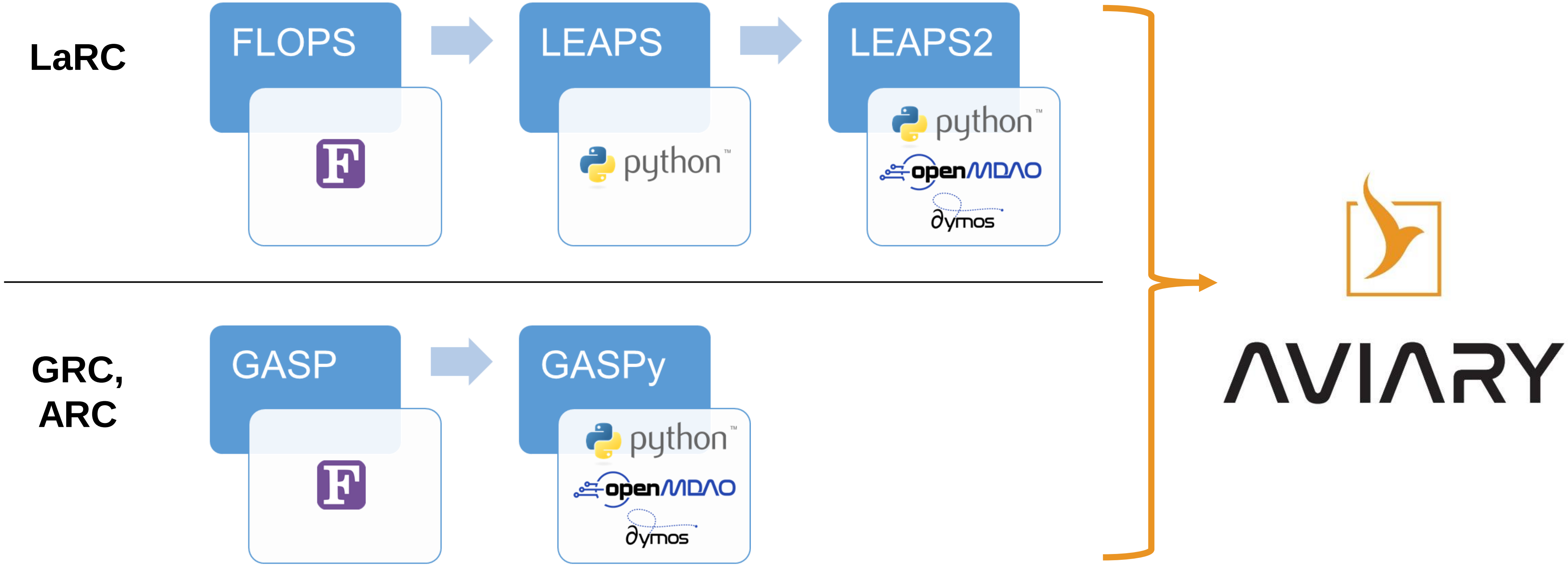
Modular for modification and substitution

Coupled Optimizations

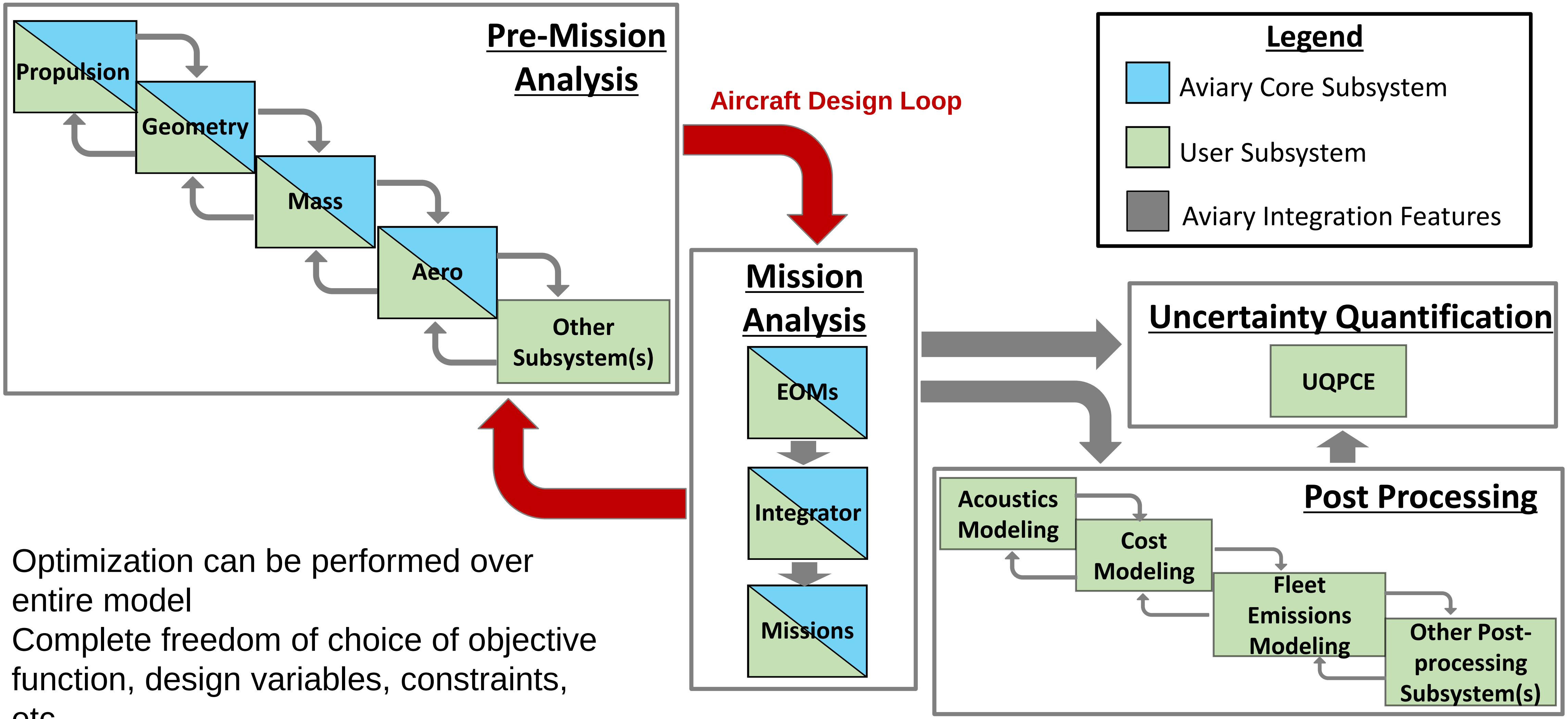
Analyze advanced concepts

Open source software

History Of Aviary



Aviary is a Modular, Extensible Design Framework



- Optimization can be performed over entire model
- Complete freedom of choice of objective function, design variables, constraints, etc.

Aviary’s User Interface

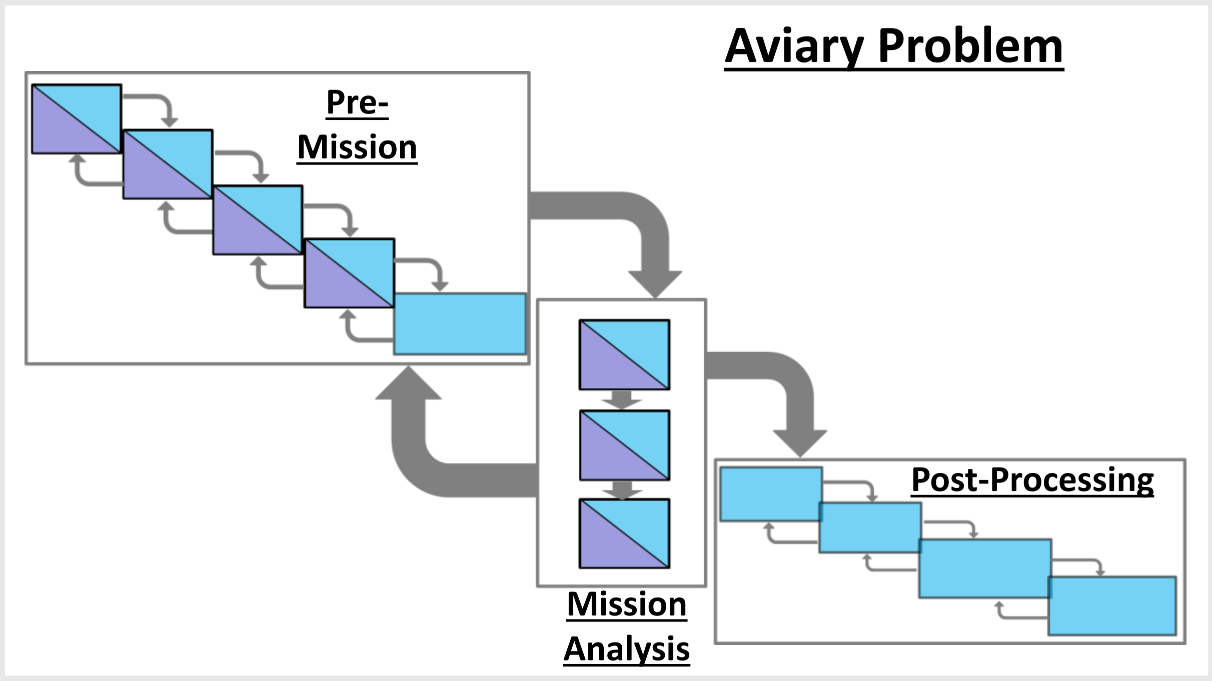
Aviary UI

Level 1: Input File → **Level 2: Python Interface** → **Level 3: Python Script**

- **Level 1: Input File** — Load user inputs into memory
- **Level 2: Python Interface** — Define subsystems and mission analysis
- **Level 3: Python Script** — Build and run model

Users can jump in at any level.

A user might use level 3 for one study, then only need the level 1 interface for another.



Adding External Subsystems

- User-defined modules need to tell Aviary what to expect from your system:
  - The states you want to integrate across the mission
  - Any new variables your system needs
  - The constraints, parameters, and design variables

- Aviary provides the `SubsystemBuilderBase` object, which you use to create your builder

```
def build_pre_mission(self):
    '''
    Build an OpenMDAO system for the pre-mission computations of the subsystem.

    Required for subsystems with pre-mission computations.

    Used in level3.py to build the pre-mission system.

    Returns
    -------
    pre_mission_sys : openmdao.core.System
        An OpenMDAO system containing all computations that need to happen in
        the pre-mission (formerly statics) part of the Aviary problem. This
        includes sizing, design, and other non-mission parameters.
    '''
    return om.Group()

def get_states(self):
    '''
    Return a dictionary of states defined by this subsystem.

    Required for subsystems with mission-based dynamics.

    Note that there must be outputs provided by the user's model that provide
    the time derivative of each state. The convention is to name the output
    f"{state_name}_rate", where `state_name` is the name of the state variable.

    Use in the phase builders (e.g. cruise_phase.py) when other states are added to the phase.

    Returns
    -------
    states : dict
        A dictionary where the keys are the names of the state variables
        and the values are dictionaries with the following keys:

        - 'units': a string indicating the units of the state variable
        - any additional keyword arguments required by Dymos for the state variable.
    '''
    return {}
```

Presentation Roadmap

1. What is Aviary?
2. What challenges does Aviary solve?
3. New features and upcoming additions
4. Walkthrough using Aviary

Converters for aerodynamic and propeller tables

Convert a FLOPS or GASP aero table into Aviary format using:

`aviary convert_aero_table GASP_aero.txt aviary_aero.csv -f GASP`

Utility name

Legacy file to be converted

Output filename

Data **F**ormat original file is from

Convert a GASP propeller map into Aviary format using:

`aviary convert_prop_table GASP_Prop.map output.prop -f GASP`

Utility name

Legacy file to be converted

Output filename

Data **F**ormat original file is from

Repository for community made models

README License

This is a supplemental repository where the Aviary community, students and professors, and enthusiasts, can share aircraft models, engines, mission models, and external subsystem builders/wrappers. This was created as a supplement of the Aviary repo. The official Aviary repo only includes aircraft used for testing and didactic examples and there was a need for a place for people to share other models.

The examples in this repo are typically the work of students and no guarantees are given to the accuracy or quality of these models. You are welcome to share and use any of these example aircraft. When uploading a model we encourage you to cite your sources for where you got your data so that other users can trace your footsteps.

Only publicly available data should be shared in this repository. Please exercise caution and carefully review your models for sensitive data before submitting. You are responsible for checking models you share with the community. Thank you for contributing!

https://github.com/OpenMDAO/Aviary_Community

Turboprop (and electroprop)

```python
class TurbopropModel(EngineModel):
    """
    EngineModel that combines a model for shaft power generation
        (default is EngineDeck)
    and a model for propeller performance
        (default is Hamilton Standard).
```

```
propulsion
  gearbox
    model
    test
    __init__.py
    gearbox_builder.py
  motor
    model
    test
    __init__.py
    motor_builder.py
  propeller
    __init__.py
    hamilton_standard.py
    propeller_map.py
    propeller_performance.py
```

Heterogeneous engines

```
def test_multiengine_fixed(self):

    test_phase_info = deepcopy(local_phase_info)
    method = ThrottleAllocation.FIXED

    test_phase_info['climb']['user_options']['throttle_allocation'] = method
    test_phase_info['cruise']['user_options']['throttle_allocation'] = method
    test_phase_info['descent']['user_options']['throttle_allocation'] = method

    engine1 = build_engine_deck(engine_1_inputs)[0]
    engine1.name = 'engine_1' # 28k TurboFan
    engine2 = build_engine_deck(engine_2_inputs)[0]
    engine2.name = 'engine_2' # 22k TurboFan

    prob = AviaryProblem()

    prob.load_inputs(inputs, test_phase_info, engine_builders=[engine1, engine2])
```

Off-Design Mission Analysis Capability

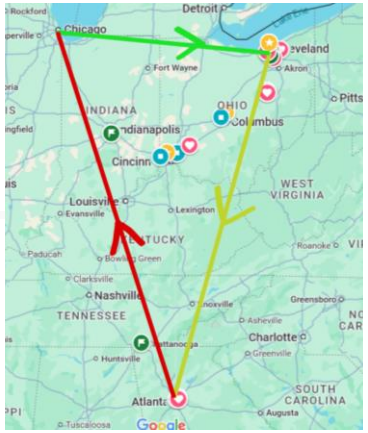
```python
prob.setup()
prob.set_initial_guesses()
prob.run_aviary_problem()
prob.save_sizing_to_json()

# Fallout Mission
prob_fallout = prob.fallout_mission()

# Alternate Mission
prob_alternate = prob.alternate_mission()
```

```python
def fallout_mission(self, run_mission=True,
                    json_filename='sizing_problem.json',
                    mission_mass=None, payload_mass=None,
                    phase_info=None, verbosity=Verbosity.BRIEF):
```

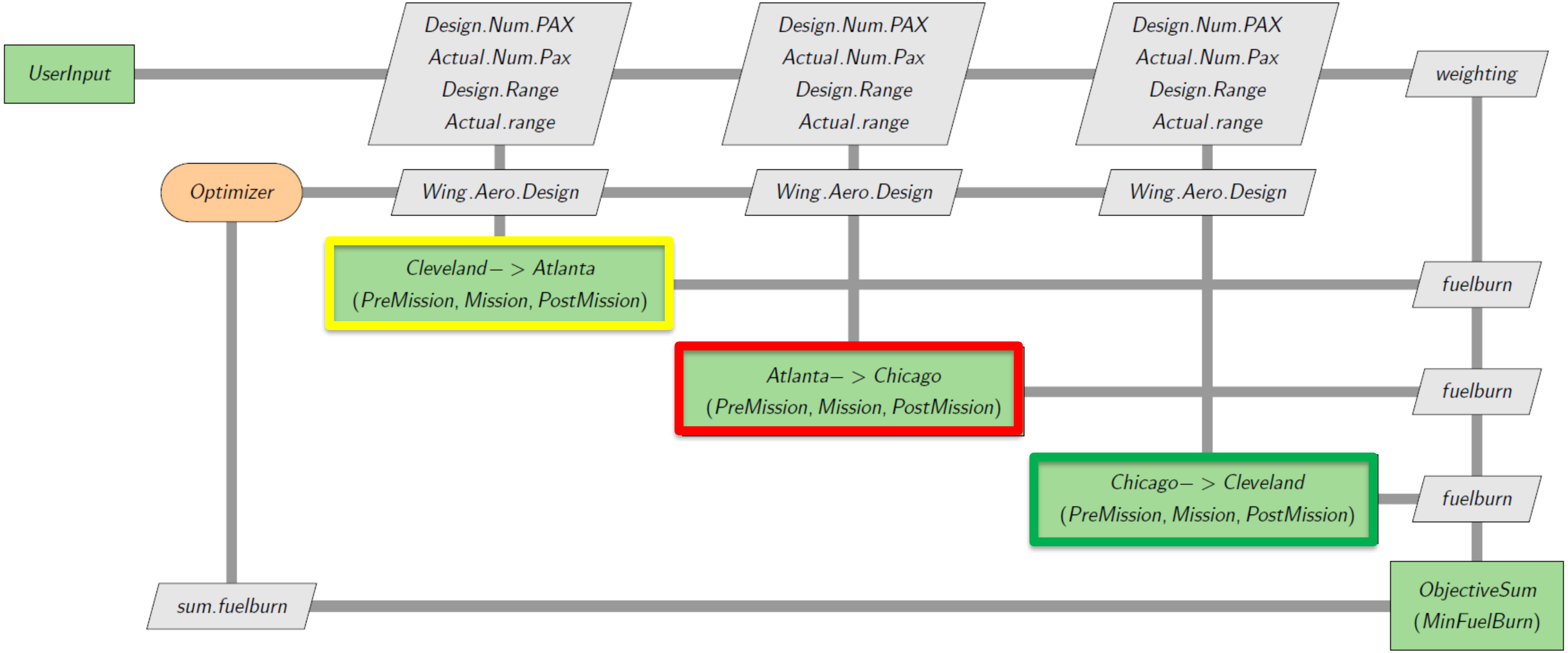
```python
def alternate_mission(self, run_mission=True,
                      json_filename='sizing_problem.json',
                      payload_mass=None, mission_range=None,
                      phase_info=None, verbosity=Verbosity.BRIEF):
```


```json
[
    "aircraft:design:compute_htail_volume_coeff",
    false,
    "unitless",
    "<class 'bool'>"
],
[
    "aircraft:design:compute_vtail_volume_coeff",
    false,
    "unitless",
    "<class 'bool'>"
],
[
    "aircraft:design:part25_structural_category",
    3,
    "unitless",
    "<class 'int'>"
],
[
    "aircraft:design:reserve_fuel_additional",
    3000,
    "lbm",
    "<class 'int'>"
],
```

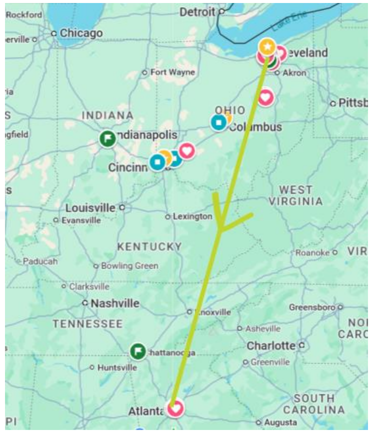
Multi-Mission Design (coming soon w/ PR #529)



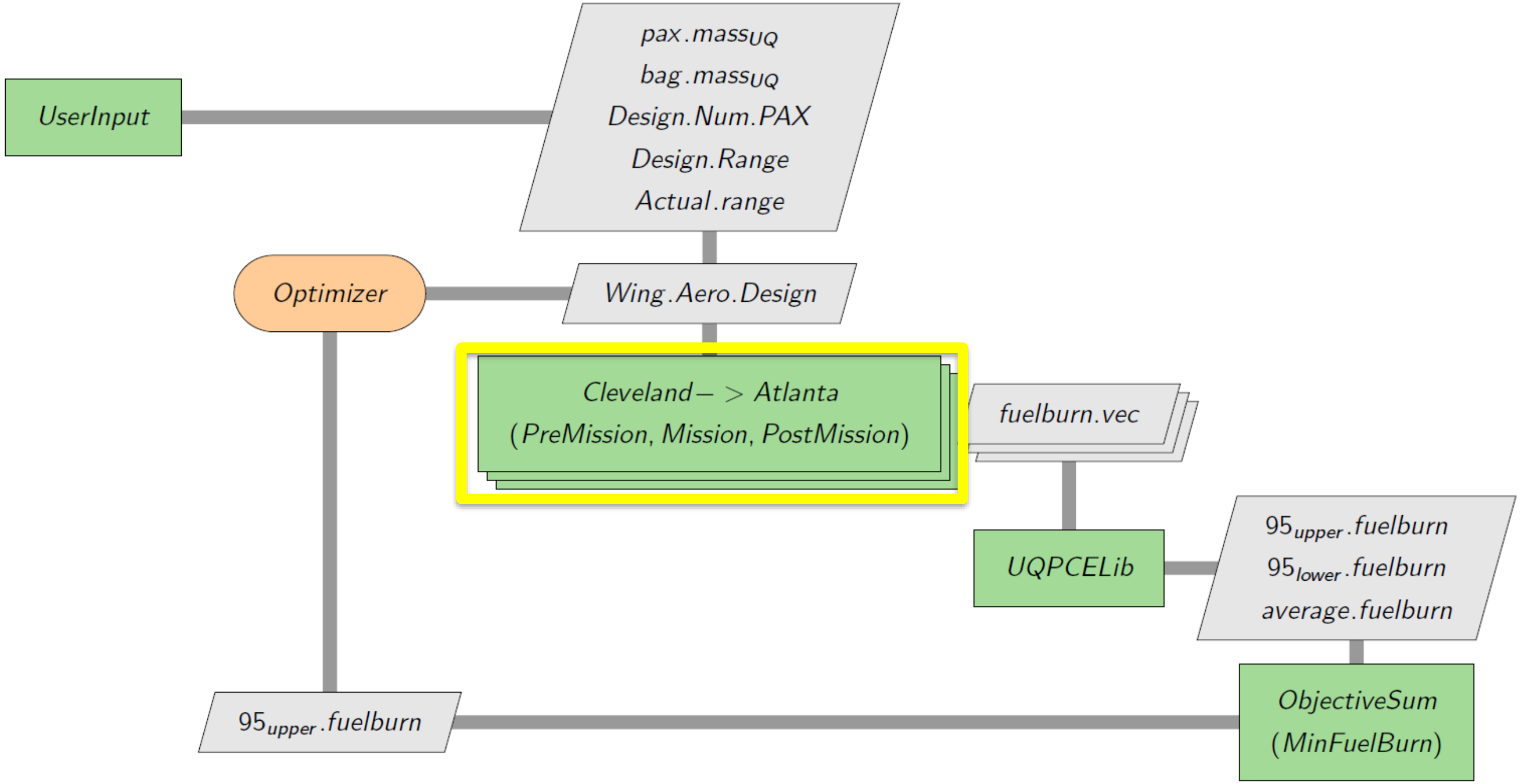
- Adds the capability to run one aircraft design in multiple missions
- Example model route Cleveland -> Atlanta -> Chicago
- Differentiates between “Design” Pax & Range vs. As-flown Pax & Range



Uncertainty Quantification with Multi-Mission (coming soon AIAA Paper)



- Demonstrates capability to combine UQPCE and Aviary for single leg
- Adds uncertainty to passenger & bag mass and propulsion
- UQPCE Library leveraged to post-process fuel burn and build confidence intervals



Presentation Roadmap

1. What is Aviary?

2. What challenges does Aviary solve?

3. New features and upcoming additions

4. Walkthrough using Aviary

Installing Aviary

- Follow the instructions in Aviary’s documentation to get your Python environment set up
- Once you are ready, Aviary can be installed as follows:

The simplest way to install Aviary to use pip in a terminal:

```
pip install om-aviary
```

This will install the latest release of Aviary and all of its dependencies.

That’s it! If you’ve done that successfully, you can now use Aviary in your Python environment.

- For the latest development version of Aviary, obtain the code directly from GitHub and install

https://openmdao.github.io/Aviary/getting_started/installation.html

Aviary's Documentation

AVIARY

Search Ctrl + K

Aviary Documentation

Getting Started

- Installation
- What Aviary Does
- Tools That Aviary is Built Upon
- Expected User Knowledge
- Onboarding Guide
- Now What?

User Guide

- Aviary User Interface
- Drawing and running simple missions
- Pre-Mission and Mission
- Outputs and How to Read Them
- Understanding the Variable Metadata
- Features and Functionalities
- Troubleshooting

Examples

- Discussing the Aviary Examples
- Conventional Aircraft and Simple Mission

Aviary Documentation

This is the landing page for all of Aviary's documentation, including a user's guide, developer's guide, and theory guide, as well as other resources. Welcome!

What Aviary is

Aviary is an aircraft analysis, design, and optimization tool built on top of the Python-based optimization framework OpenMDAO. Aviary provides a flexible and user-friendly optimization platform that allows the beginning aircraft modeler to build a useful model, the intermediate aircraft modeler to build an advanced model, and the advanced aircraft modeler to build any model they can imagine.

Features of Aviary include:

- included simple subsystem models for aerodynamics, propulsion, mass, geometry, and mission analysis
- ability to add user-defined subsystems
- gradient-based optimization capability
- analytical gradients for all included subsystems

How to Read These Docs

The Aviary documentation is broken up into several sections, each of which is designed to teach a different aspect of Aviary. Reading the entirety of the docs is highly recommended for new users, but please read through the Getting Started section at a minimum.

You can read through the documentation in order or you can jump to the sections that interest you the most.

> **Note**
>
> Use the interactive table of contents on the left side of the page to navigate through the documentation.

User Guide

Contents

Example Problem

- As an example, we will run NASA's N3CC aircraft using Aviary
  - The N3CC is an advanced single-aisle transport, EIS 2035
  - It is a FLOPS model, which we will convert to an Aviary model then run

The N3CC Model

FLOPS input file

N3CC_FLOPSin.txt

```
REF MDL N3CC (26616) AR11 1220t 1340p turbofan_22k M785 20210721

 &OPTION

 ! Program Control, Execution, Analysis and Plot Option Data
 mprint=1,   iopt=1,   ianal=3,   ineng=0,   itakof=1,   iland=1
 nopro=1,   noise=0,   icost=0,   ifite=0

 ! Plot files for XFLOPS Graphical Interface Postprocessor (MSMPLOT)
 ixfl=1

 ! Takeoff and Climb Profile File for Noise Calculations (NPROF)
 npfile=1

 ! Approach and Landing Profile File for Noise Calculations (LPROF)
 lpfile=0

 ! Drag Polar Plot File (POLPLOT)
 ipolp=1,   polalt=39000.0,
 nmach = 3,   pmach=0.3, 0.45222, 0.785

 ! Engine Performance Data Plot File (THRPLOT)
 ipltth=2

 ! Design History Plot File (HISPLOT)
 iplths=0
/
```

FLOPS-formatted engine deck

Turbofan_22k.txt

0.00	0.0	50.	22200.5	0.0	5157.3	0.2323	17.737
0.00	0.0	47.	19980.5	0.0	4500.3	0.2252	17.955
0.00	0.0	44.	17760.5	0.0	3879.8	0.2185	18.493
0.00	0.0	41.	15540.4	0.0	3300.0	0.2123	19.097
0.00	0.0	38.	13320.3	0.0	2756.1	0.2069	19.670
0.00	0.0	35.	11100.2	0.0	2255.1	0.2032	20.252
0.00	0.0	32.	8880.2	0.0	1790.8	0.2017	20.979
0.00	0.0	29.	6660.2	0.0	1368.2	0.2054	21.499
0.00	0.0	26.	4440.1	0.0	964.9	0.2173	23.442
0.00	0.0	21.	1110.0	0.0	500.3	0.4507	55.372
0.10	0.0	50.	22936.3	3299.9	5279.7	0.2689	17.625
0.10	0.0	47.	20834.5	3161.7	4653.7	0.2633	17.750
0.10	0.0	44.	18751.0	3041.7	4059.2	0.2584	18.179
0.10	0.0	41.	16665.2	2919.7	3496.0	0.2543	18.761
0.10	0.0	38.	14543.0	2761.2	2972.4	0.2523	19.161
0.10	0.0	35.	12412.3	2594.1	2464.7	0.2510	19.731
0.10	0.0	32.	10237.7	2383.1	2000.9	0.2547	20.126
0.10	0.0	29.	8024.0	2133.0	1561.0	0.2650	20.506
0.10	0.0	26.	5771.8	1844.5	1138.1	0.2898	21.215
0.10	0.0	21.	2380.8	1399.0	576.5	0.5872	28.966
0.20	0.0	50.	24389.7	6753.8	5444.7	0.3087	17.370
0.20	0.0	47.	22353.3	6481.1	4836.8	0.3047	17.401
0.20	0.0	44.	20360.0	6251.2	4256.6	0.3017	17.711
0.20	0.0	41.	18364.7	6019.6	3706.3	0.3002	18.138
0.20	0.0	38.	16309.0	5727.5	3184.6	0.3010	18.434
0.20	0.0	35.	14231.2	5413.3	2672.2	0.3030	18.805
0.20	0.0	32.	12066.9	5012.5	2198.3	0.3116	18.974

- Both of these files need to be converted to Aviary format
  - Aviary includes command-line utilities that will do this for us!

Aviary Utilities

Aviary's utilities can be found in the documentation, or with `` `aviary --help` `` or `` `aviary -h` ``

```
aviary -h
usage: aviary [-h] [--version]  ...

aviary Command Line Tools

options:
  -h, --help           show this help message and exit
  --version            show version and exit

Tools:

    convert_aero_table
                       Converts FLOPS- or GASP-formatted aero data files into Aviary csv format.
    convert_engine     Converts FLOPS- or GASP-formatted engine decks into Aviary csv format. FLOPS decks are changed
                       from column-delimited to csv format with added headers. GASP decks are reorganized into column
                       based csv. T4 is recovered through calculation. Data points whose T4 exceeds T4max are removed.
    convert_prop_table
                       Converts GASP-formatted propeller map file into Aviary csv format.
    dashboard          Run the Dashboard tool
    draw_mission       Allows users to draw a mission profile for use in Aviary.
    fortran_to_aviary
                       Converts legacy Fortran input decks to Aviary csv based decks
    hangar             Allows users that pip installed Aviary to download models from the Aviary hangar
    plot_drag_polar    Plot a Drag Polar Graph using a provided polar data csv input
    run_mission        Runs Aviary using a provided input deck
```

We will be using these utilities

1. Convert input file
2. Convert engine deck
3. Draw mission profile
4. Run mission
5. View results in dashboard

Convert Input File

Convert the FLOPS input file into Aviary format using:

`aviary fortran_to_aviary N3CC_FLOPSin.txt -o N3CC.csv -l FLOPS`

Utility name

Legacy file to be converted

Output filename

Legacy code original file is from

Check the Generated Input File

N3CC.csv

```
# created 10/17/24 at 15:43 by [redacted]
# FLOPS-derived aircraft input deck converted from N3CC_FLOPSin.txt

# Input Values
aircraft:air_conditioning:mass_scaler,0.98094,unitless
aircraft:anti_icing:mass_scaler,0.53202,unitless
aircraft:apu:mass_scaler,1.02321,unitless
aircraft:avionics:mass_scaler,1.123226,unitless
aircraft:canard:laminar_flow_lower,0,unitless
aircraft:canard:laminar_flow_upper,0,unitless
aircraft:canard:mass_scaler,1,unitless
aircraft:crew_and_payload:baggage_mass_per_passenger,35,lbm
aircraft:crew_and_payload:flight_crew_mass_scaler,1,unitless
aircraft:crew_and_payload:mass_per_passenger,165,lbm
aircraft:crew_and_payload:misc_cargo,0,lbm
```

...

```
# Unconverted Values
AERIN.clapp,2
AERIN.dratio,1
AERIN.elodma,0
AERIN.elodss,0
AERIN.flldg,8190
```

- FLOPS variables were matched to equivalent Aviary variables
- At the bottom of the file are unconverted variables
  - These are unused by Aviary or not yet implemented (such as mission definition)
  - Manually review these!

Convert Engine Deck

Convert the FLOPS-formatted engine deck to Aviary format using:

`aviary convert_engine turbofan_22k.txt turbofan_22k.csv -f FLOPS`

Utility name

Legacy file to be converted

Output filename

Format original file is in

Check the Generated Engine Deck

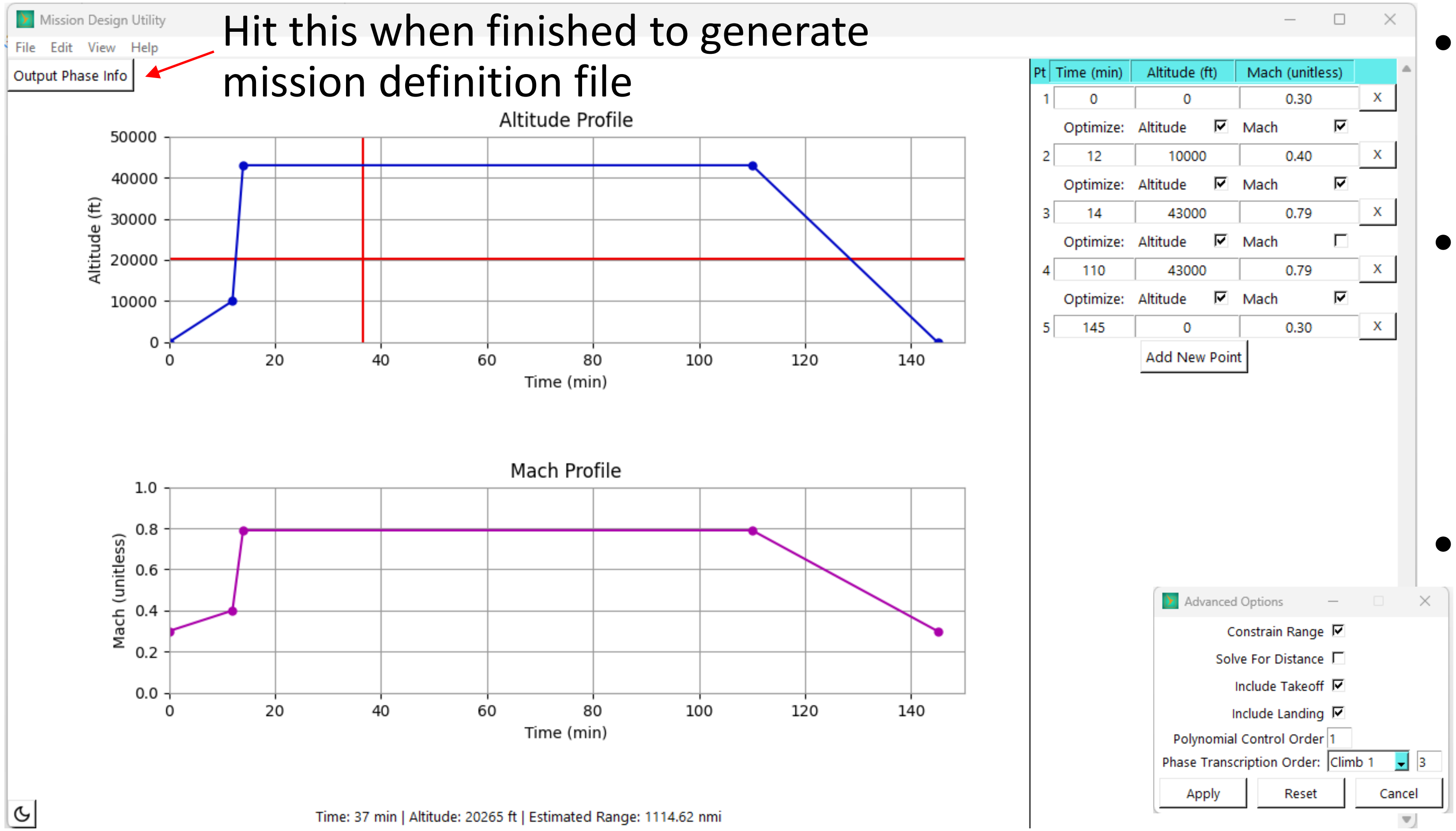
turbofan_22k.csv

```
# created 10/17/24 at 15:13 by [redacted]
# FLOPS-derived engine deck converted from turbofan_22k.txt

Mach_Number, Altitude (ft),   Throttle, Gross_Thrust (lbf), Ram_Drag (lbf), Fuel_Flow (lb/h), NOx_Rate (lb/h)
        0.0,           0.0,       21.0,              1110.0,             0.0,            500.3,          55.372
        0.0,           0.0,       26.0,              4440.1,             0.0,            964.9,          23.442
        0.0,           0.0,       29.0,              6660.2,             0.0,           1368.2,          21.499
        0.0,           0.0,       32.0,              8880.2,             0.0,           1790.8,          20.979
        0.0,           0.0,       35.0,             11100.2,             0.0,           2255.1,          20.252
        0.0,           0.0,       38.0,             13320.3,             0.0,           2756.1,           19.67
        0.0,           0.0,       41.0,             15540.4,             0.0,           3300.0,          19.097
        0.0,           0.0,       44.0,             17760.5,             0.0,           3879.8,          18.493
        0.0,           0.0,       47.0,             19980.5,             0.0,           4500.3,          17.955
        0.0,           0.0,       50.0,             22200.5,             0.0,           5157.3,          17.737
        0.0,        2000.0,       21.0,              1061.5,             0.0,            472.8,          54.005
        0.0,        2000.0,       26.0,              4246.0,             0.0,            917.9,          23.255
        0.0,        2000.0,       29.0,              6369.0,             0.0,           1302.4,          21.423
        0.0,        2000.0,       32.0,              8492.0,             0.0,           1706.2,          20.883
        0.0,        2000.0,       35.0,             10615.1,             0.0,           2149.0,          20.188
        0.0,        2000.0,       38.0,             12738.0,             0.0,           2632.7,          19.552
        0.0,        2000.0,       41.0,             14861.1,             0.0,           3149.6,          19.014
        0.0,        2000.0,       44.0,             16984.1,             0.0,           3709.7,          18.353
        0.0,        2000.0,       47.0,             19107.0,             0.0,           4307.7,          17.846
        0.0,        2000.0,       50.0,             21230.1,             0.0,           4936.7,          17.715
        0.0,        5000.0,       21.0,               989.3,             0.0,            434.8,          52.114
        0.0,        5000.0,       26.0,              3957.4,             0.0,            850.3,            23.0
        0.0,        5000.0,       29.0,              5936.0,             0.0,           1206.2,          21.325
        0.0,        5000.0,       32.0,              7914.8,             0.0,           1582.5,          20.728
```

- Aviary uses a 2D, column separated table with headers including units

Draw a Mission Profile

`aviary draw_mission`



- Points can be added by clicking on the plots or manually typing them out
- This mission was set to match the FLOPS model
  - Allow optimizer to pick alt, Mach except for fixed Mach cruise
- Add takeoff and landing using ‘Edit’ -> ‘Advanced Options’

Examine Mission Definition File

outputted_phase_info.py

```
phase_info = {
    'pre_mission': {'include_takeoff': True, 'optimize_mass': True},
    'climb_1': {
        'subsystem_options': {'core_aerodynamics': {'method': 'computed'}},
        'user_options': {
            'optimize_mach': True,
            'optimize_altitude': True,
            'polynomial_control_order': 1,
            'use_polynomial_control': True,
            'num_segments': 4,
            'order': 3,
            'solve_for_distance': False,
            'initial_mach': (0.3, 'unitless'),
            'final_mach': (0.4, 'unitless'),
            'mach_bounds': ((0.27999999999999997, 0.42000000000000004), 'unitless'),
            'initial_altitude': (0.0, 'ft'),
            'final_altitude': (10000.0, 'ft'),
            'altitude_bounds': ((0.0, 10500.0), 'ft'),
            'throttle_enforcement': 'path_constraint',
            'fix_initial': True,
            'constrain_final': False,
            'fix_duration': False,
            'initial_bounds': ((0.0, 0.0), 'min'),
            'duration_bounds': ((6.0, 18.0), 'min'),
        },
        'initial_guesses': {'time': ([0.0, 12.0], 'min')},
    },
    'climb_2': {
        'subsystem_options': {'core_aerodynamics': {'method': 'computed'}},
        'user_options': {
            'optimize_mach': True,
            'optimize_altitude': True,
```

- This file, referred to as “phase_info”, directly interfaces with dymos
- It can be directly modified by users to fine-tune mission definition

Run the Mission

The mission we set up can be run using `aviary run_mission N3CC.csv`

- Aviary assumes we are using the auto-generated *outputted_phase_info.py*, if we want to use a different mission you can specify that file after the input file

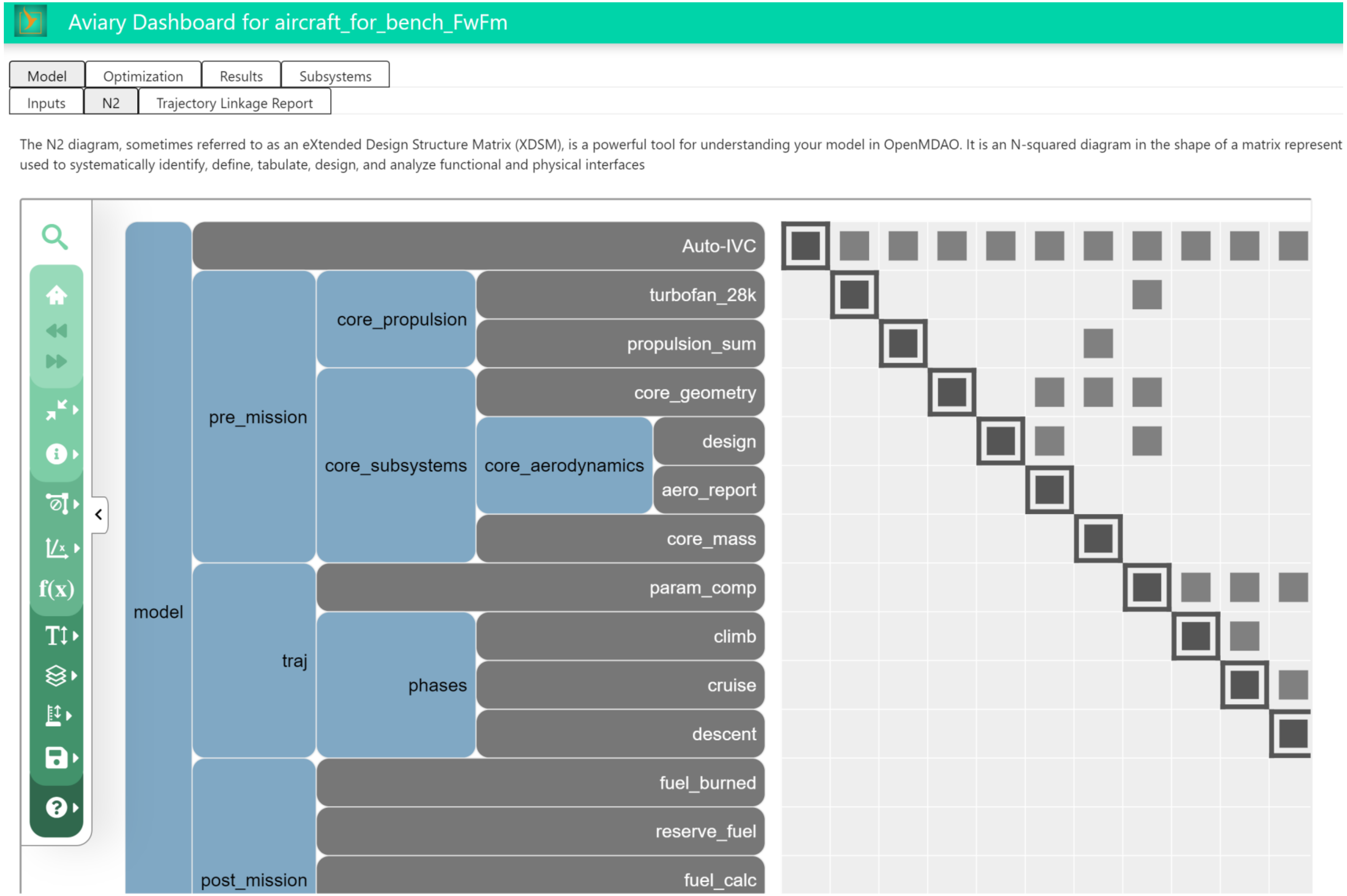
You can also run this aircraft model using the Python interface, which gives you a few more options. For a simple problem like ours, it looks like this:

```
prob = av.run_aviary(
    "N3CC.csv", phase_info, optimizer="SLSQP"
)
```

View Your Results

Open the Aviary dashboard using: `` `aviary dashboard N3CC` ``

View your Results



Understand, debug, and parse results with interactive reports

- High-level summary of final aircraft design
- Detailed optimization reports
- Detailed reports from each subsystem
- Plots and figures of aircraft trajectory and time-dependent variables

View Your Results

Aviary Dashboard for aircraft_for_bench_FwFm

Model | Optimization | Results | Subsystems

Mission Summary | Trajectory Results Report | Aviary Variables | Timeseries Mission Output Report | Aircraft 3d model

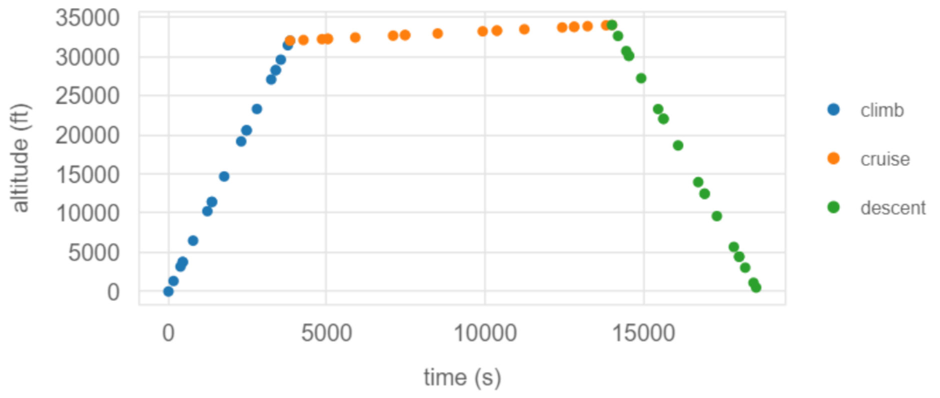
This is one of the most important reports produced by Aviary. It will help you visualize and understand the optimal trajectory produced by Aviary. Users should play with it and try to grasp all possible features. This ı
timeseries tab, users can select which phases to view. Other features include hovering the mouse over the solution points to see solution value and zooming into a particular region for details, etc.

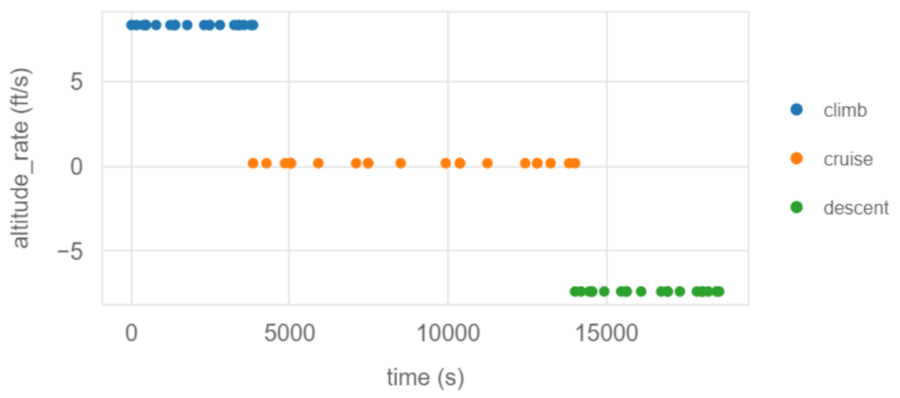
Results of aircraft_for_bench_FwFm
Creation Date: 2024-07-02 11:22:35

Timeseries | climb parameters | cruise parameters | descent parameters

Display data: Solution | Simulation

Plot phases: climb × | cruise × | descent ×





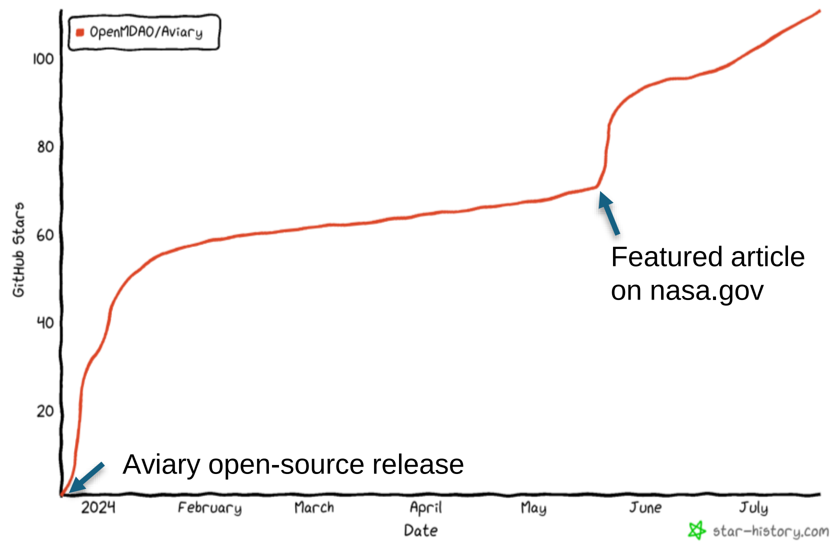
Aviary is already being used across industry, government, and academia

Results

- Aviary was open-source released at end of CY23
- Aviary has been used by AATT to model TTBW with external tool integration
- Built-in capabilities expanding under EPFD partnership to include hybrid and all-electric engines and distributed propulsion

Significance

- Aviary has been used in published design studies, advancing the state-of-the-art for conceptual aircraft design at NASA
- Aviary has gathered widespread interest with partners in industry, government, and academia using the tool and providing feedback
- Over 120 GitHub stars and counting!



"Stars" on Aviary GitHub Repository

External Users/Partners

Boeing*

Georgia Tech*

Northrop Grumman*

University of Michigan

Naval Research Lab

*MBSA&E NRA Partners

The Aviary Team

Current Members
- Jason Kirk (LaRC)
- Eliot Aretskin-Hariton (GRC)
- Xun Jiang (LaRC)
- Ken Moore (GRC)
- Carl Recine (ARC)
- Herb Schilling (GRC)
- Chris Bennett (LaRC)
- Kaushik Ponnapalli (GRC)

Past Members
- Darrell (DJ) Caldwell (LaRC)
- Jennifer Gratz (GRC)
- John Jasa (GRC)
- Kenny Lyons (ARC)
- Ben Margolis (ARC)
- Samara Murri (formerly LaRC)
- Erik Olson (LaRC)
- Janet Ross (LaRC)
- Dahlia Pham (ARC)
- Jeff Chapman (GRC)

Current and Past Advisors
- Rob Falck (GRC)
- Joseph Garcia (ARC)
- Justin Gray (formerly GRC)
- Eric Hendricks (GRC)
- Ben Phillips (LaRC)

Start Using Aviary Today!

AVIARY

SCAN HERE

Email agency-aviary@mail.nasa.gov
to connect with the Aviary team

Or find and install Aviary through GitHub:

https://github.com/OpenMDAO/Aviary

Or through the Python package manager:

`pip install om-aviary`

Aviary activities are co-funded by the T³, AATT, and EPFD projects