

POLITECNICO DI TORINO



Master's Degree in Aerospace Engineering
Academic Year 2024-2025

Integration of Model-Based Design and Surrogate-Based Multi-Disciplinary Optimization into a Dynamic Simulation Framework for Thermal Management Systems

Supervisors

Prof. Nicole VIOLA

Francesco TORRIGIANI (DLR)

Sparsh GARG (DLR)

Candidate

Andrea MAMELI

Table of Contents

List of Figures	V
List of Tables	IX
Acknowledgements	XI
Abstract	XIII
Symbols and Acronyms	XVII
1 Introduction	1
2 Literature Review	7
2.1 State of the Art and Research Gap	7
2.2 Hybrid-Electric Aircraft	11
2.3 Thermal Management System	15
2.4 TMS Components Sizing	18
2.5 Optimization in Aircraft TMS	19
2.5.1 Design Of Experiments	19
2.5.2 Surrogate Modeling	19
2.5.3 Surrogate-Based MDO	21
3 Methodology	23
3.1 Model-Based Design	23
3.1.1 Modelica libraries	26
3.1.2 OpenModelica Models Definition	30
3.1.3 Modelica Models Construction	34
3.2 Mission Profile and Boundary Conditions	54
3.3 Components Empirical Sizing	65
3.3.1 Turbomachinery - Powers estimation	66
3.3.2 Turbomachinery - Masses estimation	68
3.3.3 Skin Heat Exchanger	69

3.3.4	Safety Factor	70
3.4	DOE Implementation	71
3.5	Surrogate Modeling	81
3.6	Surrogate-Based MDO	88
3.7	Pareto Validation	93
3.8	General Workflow Wrap-Up	94
4	Validation	97
4.1	Modelica Models Validation	98
4.1.1	Model Components	98
4.1.2	System Behaviour	99
4.2	Surrogate Models Validation	102
4.3	Results Validation	105
5	Application Case and Results	107
5.1	Level-1 DOE	108
5.2	Level-2 DOE	110
5.3	Optimization	112
5.4	Pareto Front Filter	114
6	Conclusions	119
6.1	Limitations	121
6.2	Further Steps	122
	Appendices	125
A	Python Implementations	125
A.1	DOE Implementation	126
A.2	Surrogate Modeling	128
A.3	Surrogate-Based Optimization	130
A.4	Pareto Validation	133
B	Workflow Detailed Diagrams	135
B.1	DOE-L1 Focused Diagram	136
B.2	DOE-L2 Focused Diagram	137
B.3	MDO Focused Diagram	138
	Bibliography	143

List of Figures

1.1	CO_2 emissions trend in aviation in the Net Zero Scenario, 2000-2030. Light blue: domestic aviation emissions. Dark blue: international aviation emissions. On the right side: bar to represent Net Zero Scenario emissions threshold [1]	2
1.2	Global total commercial air passenger traffic, 2019-2023. Light blue bars show the number of civil aviation passengers in terms of trillions of passengers per travelled kilometer. Dark blue dots mark the average load factor (with respect to total capacity) percentage reached in that year [2]	2
1.3	Clean Aviation project logo [3]	3
1.4	<i>ODE4HERA</i> project banner [4]	4
2.1	Five proposed architectures by Coutinho <i>et al.</i> Schematic diagram of the loops and their main components [5]	9
2.2	Series Hybrid architecture block diagram. Blocks indicate system components and arrows indicate power transfers [6]	11
2.3	Parallel Hybrid architecture block diagram. Blocks indicate system components and arrows indicate power transfers [6]	12
2.4	Series-Parallel Hybrid architecture block diagram. Blocks indicate system components and arrows indicate power transfers [6]	12
2.5	Turboelectric Hybrid architecture block diagram. Blocks indicate system components and arrows indicate power transfers [6]	12
2.6	All-Electric Hybrid architecture block diagram. Blocks indicate system components and arrows indicate power transfers [6]	12
3.1	Different approaches to system simulation. Blocks: mathematical tools. Blue arrows: simulation language/method [7]	24
3.2	OMEdit opening page with the Modelica library on the left hand side	26
3.3	Blocks sub-library. As an example, Continuous sub-category has been opened, showing all its related elements	27
3.4	FirstOrder block on Modelica textual notation	27

3.5	Overview of ThermofluidStream library and its main packages on the left side and key components on the right side [8]	28
3.6	OpenModelica Connection Editor home view. This is the first view shown to an user when the software is launched	30
3.7	Test model in OMEditor. An example of model construction is shown, with blocks from both ThermofluidStream and Modelica libraries	30
3.8	Test model from figure 3.7 in Modelica textual notation	31
3.9	Parameters window for pump block. the rows can be filled with desired values	32
3.10	Plotting window in OpenModelica. On the right hand side, a list of all the plottable variables will appear, allowing the user to choose which variables he wants to plot	34
3.11	<i>Simulation Setup window. This window allows the user to define the simulation parameters like stat and end time or mathematical method</i>	34
3.12	Heat load model in Modelica graphical notation	36
3.13	Non-filtered (red line) and filtered (blue line) heat load plotted in OpenModelica plotting environment	36
3.14	EGW loop model in Modelica graphical notation	37
3.15	Ram Air circuit in Modelica graphical notation	41
3.16	Skin Heat Exchanger model in Modelica graphical notation	42
3.17	Vapour Compression System model in Modelica graphical notation	45
3.18	Fuel loop model in Modelica graphical notation	46
3.19	Fuel flow CombiTimeTable block definition in OpenModelica Editor environment	47
3.20	Architecture 1 in Modelica graphical notation	49
3.21	Architecture 2 in Modelica graphical notation	50
3.22	Architecture 3 in Modelica graphical notation	51
3.23	Architecture 4 in Modelica graphical notation	52
3.24	Architecture 5 in Modelica graphical notation	53
3.25	Altitude curve for the mission profile in table 3.2	55
3.26	Mach curve for the mission profile in table 3.2	56
3.27	Mission profile: Heat Load from table 3.3	57
3.28	Partial screenshot of FlatPlateGc_ISA block in Modelica textual notation	63
3.29	Non-filtered (red line) and filtered (blue line) Mach in OpenModelica plotting window	64
3.30	Typical Safety Factors in engineering projects [9]. Aircraft components safety factor usually lies between 1.5 and 2.5	70
3.31	DOE L2 coverage with PoV scattering. Yellow points are the feasible generated design space points, purple points are the unfeasible ones	79

3.32	Comparison between classical design process (top) and optimized design process (bottom) [10]. Optimization adds more loops and decision blocks to the workflow	88
3.33	2D Pareto plot. On the left: all the evaluated points plotted with respect to the two objectives, f_1 and f_2 . On the right, an example of Pareto front [10]	89
3.34	Complete workflow proposed in this thesis. It is composed by seven consecutive steps	94
3.35	Workflow <i>XDSM</i> diagram. The two DOEs (yellow boxes), the optimizer (purple box) and the validation filter (blue box and red box) can be seen	95
4.1	EGW pump pressure differential throughout the defined mission in OpenModelica plotting window	98
4.2	EGW (red) and ATJ (blue) temperatures during the reference mission. Plotted in OpenModelica plotting window	99
4.3	Heat load Heat Flow (red), RHX Heat Flow (blue), FHX Heat Flow (green), SHX Heat Flow (purple) throughout the reference mission. Plotted in OpenModelica plotting window	100
4.4	EGW Pump inlet pressure (red) and ATJ Pump inlet pressure (blue) throughout the reference mission. Plotted in OpenModelica plotting window	101
4.5	Mass surrogate (top) and power surrogate (bottom) relative error distribution histogram. In both cases, relative errors converge to 0%	103
5.1	Level-1 DOE distribution plot. Yellow points are the feasible generated design space points, purple points are the unfeasible ones	108
5.2	Level-2 DOE distribution plot. Yellow points are the feasible generated design space points, purple points are the unfeasible ones	110
5.3	Resulting Pareto front. The front is indicated with filled points whose color depends on PoV value (right hand side colour bar). The rest of the generated population is plotted with empty points	112
5.4	Total mass (top) and peak electrical power (bottom) validation error distribution histogram. red dashed lines mark the applied acceptance region	114
5.5	Surrogate vs Modelica values comparison: mass (top) and power (bottom). Blue points coordinates are x : surrogate-predicted value and y : value obtained from Modelica re-simulation. Red dashed line marks the ideal optimum	115

5.6	Pareto front from validation results. Pareto points coordinates indicate peak electrical power and total system mass obtained from Modelica re-simulations	116
5.7	Surrogate-predicted (blue line) vs Modelica-based (red line) Pareto front. the shift to right-bottom side is clearly visible: re-simulating in Modelica gives lower peak electrical powers and higher total system masses	118
B.1	Detailed <i>XDSM</i> diagram for DOE L1. The latter is expanded and it is possible to see all blocks that make part of it. It adapts data to a Modelica-friendly format, launches simulations, takes the output to perform sizing and finally builds PoV function and modifies design space bounds	136
B.2	Detailed <i>XDSM</i> diagram for DOE L2. It performs the same operations as DOE L1 (figure B.1), with two main differences: PoV function is needed as input because it filters DOE candidate points and, as second difference, this time both surrogates (mass and power) and PoV are built	137
B.3	Detailed <i>XDSM</i> diagram for MDO. Showing the complete Optimizer block it is possible to see the iterations performed on mass and power surrogates. The output is the value of the two objective values in optimum (Pareto) points	138

List of Tables

2.1	Design variables considered in the MDO process [5]	8
3.1	Common components/loops for the 5 architectures	35
3.2	Mission profile data	55
3.3	System heat load per mission segment [11]	56
3.4	Primary constants for ISA model [12]	59
3.5	Sizing input variables for each component	65
3.6	Empirical sizing mass law coefficients	69
3.7	Design variables for DOE and MDO workflow	72
3.8	Design variables ranges	73
3.9	EGW Pump nominal values [13]	73
3.10	ATJ Pump nominal values [14]	74
3.11	Ram-Air Fan nominal values [15]	74
3.12	VCS Compressor nominal values [16]	74
5.1	Design space boundaries after DOE L1	109
5.2	Pareto front values range	112
5.3	Minimum values comparison for Level 2 DOE and Pareto set	113
5.4	Validated points from final Modelica validation	117

Acknowledgements

I would like to express my heartfelt thanks to all the people who made this work possible.

First of all, I would like to thank Professor Nicole Viola from Politecnico di Torino for giving me the opportunity to live this experience. Her help and guidance have been essential in defining the topic of interest and finding a suitable thesis work.

I would also like to thank my university, Politecnico di Torino, and all the people who are part of it. The academic and personal formation that I built at Politecnico has been the essential basis for this work.

A special thanks to DLR, Deutsches Zentrum für Luft- und Raumfahrt, the institution that hosted me for six months during the development of this thesis, and especially the DLR Institute of System Architectures in Aeronautics in Hamburg. I would also like to thank Dr Björn Nagel, the Institute Director and Dr Luca Boggero, leader of the Digital Methods for System Architecting team.

A special thanks goes to my two supervisors at DLR, Francesco Torrigiani and Sparsh Garg. Their help and dedication have been an essential part of these six months in Hamburg, guiding me in the development of the thesis. Their comments and suggestions pushed me to give my best and put my knowledge into practice, adding every single day a small new lesson.

If I look back to six months ago, I can truly feel how much my preparation has improved.

This experience has been the keystone of my academic path.

Finally, I must not forget all the people who, in many different ways, helped build the person before the engineer.

Thanks to my parents Paolo and Nanda, my girlfriend Angelica and all my friends.

You are my roots and my future, I would be nobody without you.

Abstract

Aviation industry has been undergoing a deep transformation in recent years. The main focus in civil aviation industry is nowadays the environmental impact. One possible solution is to address the emissions concern is to move towards more-electric architectures.

One of the main problems in Hybrid-Electric Architectures is waste heat produced by the electric devices. As higher powers are required, more waste heat is produced: a good waste heat management system is required to move towards more-electric architectures. For this reason, having a good Thermal Management System (TMS) design is vital.

To achieve better TMS architectures, new design challenges must be addressed: integrating Model-Based Design and Multi-Disciplinary Optimization with system dynamic simulations may be an option to enhance the design process.

First step will be building TMS architectures in Modelica language and performing dynamic simulations. The choice of dynamic simulations is justified by the need of analyzing those critical systems in a real-world behaviour, addressing all possible issues related with dynamic simulations. Static simulations, the most performed type of simulation in literature, do not cover all the possible happenings that a robust TMS must address.

To pursue execution speed and reduce computational cost, surrogate functions are used in optimization instead of the actual models. Surrogate functions can, given the same inputs as Modelica simulations, represent simulation outputs with very high reliability. These functions give also the opportunity to have an easier optimization (Multi-Disciplinary Optimization - MDO) process in terms of problem definition and computational cost. Surrogate functions need DOEs (Design OF Experiments) to be built.

The real novelty brought by this work is the usage of DOE and surrogates to integrate dynamic simulation in MDO in order to solve two key problems: computational speed and dynamic simulation execution instability (hidden constraints). Two DOEs are used, the first one to train a PoV (Probability of Viability) function, which then drives the second DOE, the one that trains the surrogates. PoV functions have been the answer to address the problem of *hidden constraints*. Several literature sources address this problem: in this thesis it revealed itself in terms of failed simulations due to out-of bounds values for the working variables, meaning that the design point is not feasible. Using PoV functions to produce more reliable and dense DOEs helps in having better surrogate functions, which means more accuracy for the final optimization results. Accuracy is finally evaluated with a validation filter by simulating all the Pareto front points.

Scientific literature helps in having a good overview of all the aforementioned aspects, but almost no research work tries to join all the elements: dynamic simulations of TMS models, DOEs, surrogate functions, PoV functions and surrogate-based optimization.

The objective of this thesis is to propose a structured methodology that enables integrating all the aforementioned aspects into a single workflow.

Symbols and Acronyms

SYMBOLS:

ε	Heat exchanger effectiveness	σ	Stefan-Boltzmann constant
T_0	Sea level air temperature	T_m	Mean film temperature
p_0	Sea level air pressure	A_{eff}	Effective area (plate)
ρ_0	Sea level air density	η_f	Effectiveness (plate)
ρ	Density	$\dot{m}$	Mass flow
g_0	Gravitational acceleration	Δp	Pressure Differential
R	Specific constant for dry air	pr	Pressure ratio (fans)
L	Thermal gradient (troposphere)	p_{asp}	Aspiration pressure (compressors)
γ	Specific heat coefficient	T_{asp}	Aspiration temperature (compressors)
M	Mach number	$\dot{V}$	Volumetric flow
V	Airspeed	P_{hyd}	Hydraulic power
a	Speed of sound	P_{el}	Electrical power
q	Dynamic pressure	η	Efficiency
r_r	Recovery Factor	γ	Isentropic index
T_r	Recovery temperature	π	Compression ratio (fans)
Pr	Prandtl number	C	Empirical scaling component
T_∞	Static free-stream temperature	α	Scaling exponent
T_f	Film temperature	t	Thickness
T_s	Surface temperature	λ	Scaled design variable
μ	Dynamic viscosity	TDH	Pump pressure head
S	Sutherland's constant	ω	Rotational speed
k	Thermal conductivity	β	Builder margin
ν	Cinematic viscosity	$RMSE$	Root Mean Square Error
Re	Reynolds number	R^2	Coefficient of determination
RE_{cr}	Critical Reynolds number	$\hat{\pi}$	PoV surrogate function
ε	Emissivity		

ACRONYMS:

ATJ:	Alcohol-To-Jet
DAE:	Differential-Algebraic Equations
DEP:	Distributed Electric Propulsion
DOE:	Design Of Experiments
EGW:	Ethylene-Glycol and Water mixture
FHX:	Fuel Heat Exchanger
GUI:	Graphical User Interface
GP:	Gaussian Process
HE:	Hybrid-Electric
HER:	Hybrid-Electric Regional
HEX:	Heat Exchanger
HFC:	Hydrofluorocarbon
IATA:	International Air Transport Association
ICAO:	International Civil Aviation Organization
ICE:	Internal Combustion Engine
IEA:	International Energy Agency
ISA:	International Standard Atmosphere
ISO:	International Organization for Standardization
LHS:	Latin Hypercube Sampling
MBD:	Model-Based Design
MBSE:	Model-Based System Engineering
MDAO:	Multi-Disciplinary Architecture Optimization
MDO:	Multi-Disciplinary Optimization
NTU:	Number of Transfer Units
ODE:	Open Digital Environment
ODE:	Ordinary Differential Equation
OMC:	OpenModelica Compiler
PoV:	Probability of Verification function
RA:	Ram Air
SBAO:	Surrogate-Based Analysis and Optimization
SBO:	Surrogate-Based Optimization
SAF:	Sustainable Aviation Fuel
SHX:	Skin Heat Exchanger
TAS:	True AirSpeed
TFS:	ThermofluidStream library
TMS:	Thermal Management System
TRL:	Technological Readiness Level
VCS:	Vapour Compression System

Chapter 1

Introduction

Air transport is one of the main transportation forms in modern times. Since the first controlled flights in the early 1900s, aeronautical technology has done many steps forward, becoming one of the fastest-evolving technologies in modern world.

Petrescu *et al.* [17] proposed a division in six technological eras for aviation history:

1. precursors: early experiments and lighter-than-air flight;
2. pioneers of heavier-than-air flight: first gliding experiments. It culminates with the Wright Flyer in 1903;
3. World War One: industrialization and specialization;
4. between the Wars: first civil and commercial flights, records and fast innovations;
5. World War Two: technological apex for piston engines and blades propellers;
6. second half of the 20th century: jet era. Democratization of civil flight in terms of costs and availability, supersonic speeds.

However, in today's technological context, this list may be incomplete: a seventh era should be added.

A new sensibility centred on the environmental aspect of our technological production and, in a wider view, in the environmental impact of our society has grown in modern society.

Aviation accounted for the 2.5% of global CO_2 emissions [18] in 2023. To address this problem, several studies and agreements have been proposed in the last years. ICAO (International Civil Aviation Organization) proposed a Net-Zero CO_2 aviation goal for 2050.

IEA (International Energy Agency) tracked (figure 1.1) CO_2 emissions from aviation, reporting on the right side the threshold to reach net-zero aviation in 2030. Light blue represents domestic aviation, while dark blue represents international aviation.

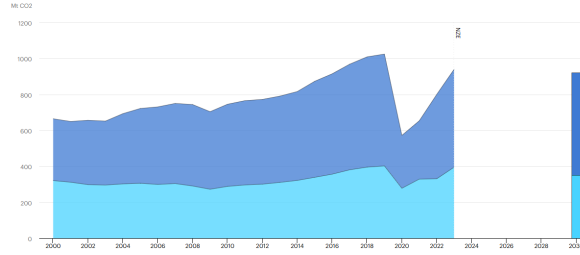


Figure 1.1: CO_2 emissions trend in aviation in the Net Zero Scenario, 2000-2030. Light blue: domestic aviation emissions. Dark blue: international aviation emissions. On the right side: bar to represent Net Zero Scenario emissions threshold [1]

Aviation emissions appear, excluding the COVID-19 pandemic years, in constant growth. In 2023, they already exceeded the 2030 net-zero threshold.

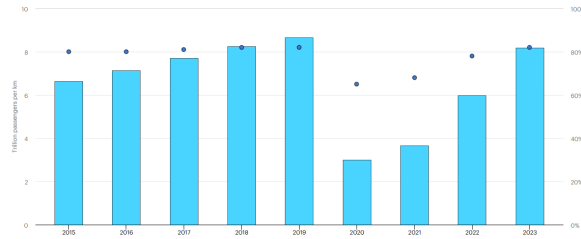


Figure 1.2: Global total commercial air passenger traffic, 2019-2023. Light blue bars show the number of civil aviation passengers in terms of trillions of passengers per travelled kilometer. Dark blue dots mark the average load factor (with respect to total capacity) percentage reached in that year [2]

Figure 1.2 underlines that, without any technological innovation, aviation emissions will continue to grow, consequently with the passengers growth. It shows the

civil aviation passengers trend continuous growth. Considering the current technological level, this passengers increase can only lead to an increase of emissions. This happens because, if current technologies are not strongly improved, per-passenger emissions will not decrease while passengers number will. New disruptive technologies must be found to stop the emissions increase.

Once understood that passenger volume decrease is unlikely and, in any case, impossible to control from a technological point of view, only one option remains: aviation emissions must be reduced.

Several governments, agencies and companies started development programs to meet the emissions reduction objective. Among these projects, European Union's research and innovation program is *Clean Aviation* [3] (logo in figure 1.3).



Figure 1.3: Clean Aviation project logo [3]

Clean Aviation project aims two main objectives:

- *to integrate and demonstrate disruptive aircraft technological innovations to decrease net emissions of greenhouse gases by no less than 30% compared to 2020 state-of-the-art technology to support the European Green Deal and climate neutrality by 2050* [3];
- *to ensure that the technological and the potential industrial readiness of innovations can support the launch of disruptive new products and services for an entry-into-service by 2035, enabling 75% of the world's civil aviation fleet to be replaced by 2050 in support of the European Green Deal* [3].

Clean Aviation is implemented through several projects. One of the projects is *ODE4HERA* (logo in figure 1.4). This project aims to *enable and accelerate the development of Hybrid-Electric Regional (HER) aircraft thanks to improved tools and techniques implemented in a transferable and Open Digital Platform (ODP)* [4]. Due to increased complexity in this next generation of aircraft, the goal is to enhance design with renewed digital tools and workflows, like:

- Model-Based System Engineering (MBSE): branch of engineering that makes a strong use of virtual models to build, size, simulate and validate systems and subsystems;
- Multi-Disciplinary Optimization (MDO): design workflow that aims to improve (optimize) some defined features of the design, called "objectives". They are either maximized or minimized, keeping in mind the inter-relationship among them.



Figure 1.4: *ODE4HERA* project banner [4]

The objective of ODE4HERA project is developing digital tools that integrate the aforementioned workflows and best practices in the design phase, accelerating it and matching the 2035 entry-into-service goal [4].

This new generation of aircraft is pushing aerospace industry towards more-electric and all-electric solutions. Having more electrical propulsive power means, on the other hand, more waste heat to deal with. Indeed, producing and using more electrical power means more electricity, and therefore more waste heat due to Joule effect. This leads designers to focus on Thermal Management Systems (TMS). If industry wants to move towards this new technological generation TMSs must be the focus of the technological revolution. Ensuring a good waste heat rejection is an essential aspect of future designs. Applying the new design challenges to MBSE and MDO is a way to address the need for improved and optimized design processes. Since TMSs will gain more importance with technological evolution, their design must be improved and optimized. Digital systems models can be a technological leverage that can enhance design process.

For instance, applying dynamic models simulations to an optimization process allows a faster addressing of design problems, considering all the aspects that a dynamic simulation brings: transients, cycles and abrupt variables peaks. An efficient workflow that starts from TMS models and performs output handling, sizing and optimization can be one of the aimed innovative tools to enhance next-generation Hybrid-Electric Regional Aircraft (HERA) design.

A research question can be formalized:

RESEARCH QUESTION

How can Model-Based Design (MBD) and dynamic simulations be integrated into a Multi-Disciplinary Optimization (MDO) framework?

This question summarizes the aim of the thesis. The work will start from building models for TMS architectures, than simulating them in a time-dependant environment (time-variable heat loads and boundary conditions). This will be the MBD/MBSE-related part of the thesis. Then, the main scientific hurdle will be addressed: finding a way to efficiently integrate the simulations in an optimization workflow. The main difficulties will be the management of dynamic simulations and their behaviour, considering failed simulations and abrupt unpredictable simulation values variations.

A novel methodology must be proposed in order to address so many coupled design challenges and finally obtain realistic and optimized results.

Chapter 2

Literature Review

Literature review chapter is meant to give a deeper insight into hybrid-electric aircraft technologies and thermal management systems used to deal with the waste heat produced by electrical components. A detailed overview of methods for sizing and optimization will be given: being MBD-MDO integration the final objective of this thesis, a good theoretical knowledge of different workflows and methods will provide useful tools to further analyses.

2.1 State of the Art and Research Gap

The starting point is *A study on thermal management systems for hybrid-electric aircraft* from Maria Maia Coutinho *et al.* [5] and the related Master's thesis *On the Design of Thermal Management Systems for Hybrid-Electric Aircraft* [11] from Coutinho M. M. as well.

These two works share the same topic: assessing the feasibility of five proposed TMS architectures for a Hybrid-Electric Regional Aircraft, studying and finally optimizing them to understand which one would be suitable for a real-world application. The analysis is conducted via MATLAB/Simulink static simulations. The authors made a survey on several scientific papers to find five suitable TMS architectures. They are all based on an Ethylene-Glycol and Water 60% – 40% mixture (EGW) cooling loop, which extracts residual heat from the electrical propulsive system and expels it in several ways. This cycle is composed by the heat load, lately defined, and a fluid pump. The heat rejection method is the main difference among the five architectures:

- **Architecture 1** (figure 2.1a): the EGW cycle transfers waste heat to a Vapour Compression System (VCS), composed by an Evaporator that absorbs the heat from EGW, a Compressor, a Condenser that releases the heat into

Ram Air and an Expansion Valve. The VCS cycle relies on R134a gas, a typical refrigerant;

- **Architecture 2** (figure 2.1b): similar to Architecture 1, but the heat is rejected by both the VCS cycle and a SHX, a plate in contact with both external air and EGW fluid;
- **Architecture 3** (figure 2.1c): the EGW cycle does not transfer the heat to a VCS cycle anymore: there is instead a direct contact between EGW and Ram Air through a Ram Air Exchanger;
- **Architecture 4** (figure 2.1d): similar to Architecture 3. Differently from the latter, it also presents a SHX to perform a better heat rejection;
- **Architecture 5** (figure 2.1e): identical to Architecture 4, with one exception: between the EGW cycle and the SHX is now present another loop, the fuel loop. This loop, composed by a fuel valve (which divides the fuel flow between the cycle and the engine, with commanded requests from the latter), a fuel tank and the SHX. The heat exchange between EGW and fuel happens through a Fuel Heat Exchanger. The fuel in this case is a SAF, the Alcohol-To-Jet (ATJ) type.

Heat load is set constant and equal to the peak load.

The simulation is then run in a MATLAB/Simulink environment. Once selected the design variables considered in the MDO process (tab 2.1), an objective function $g(x)$ is composed.

Table 2.1: Design variables considered in the MDO process [5]

Variable	Variable Name	Unit
Skin Heat Exchanger Area	A_{SHX}	m^2
Skin Heat Exchanger Position	X_{SHX}	m
Boundary layer thickness	t_{bd}	m
Cooling liquid mass flow rate	$\dot{m}_{EGW}$	kg/s
Re-circulation fuel mass flow rate	$\dot{m}_{EGW}$	kg/s
Heat Exchangers temperature difference	ΔT_{HEX}	K

The optimization process relies on NGSA-II Algorithm [19]. Uncertainty was added to some variables. Following a parametric study meant to evaluate models' sensitivity to variables variation, the authors chose architectures A2 and A4 for the optimization process.

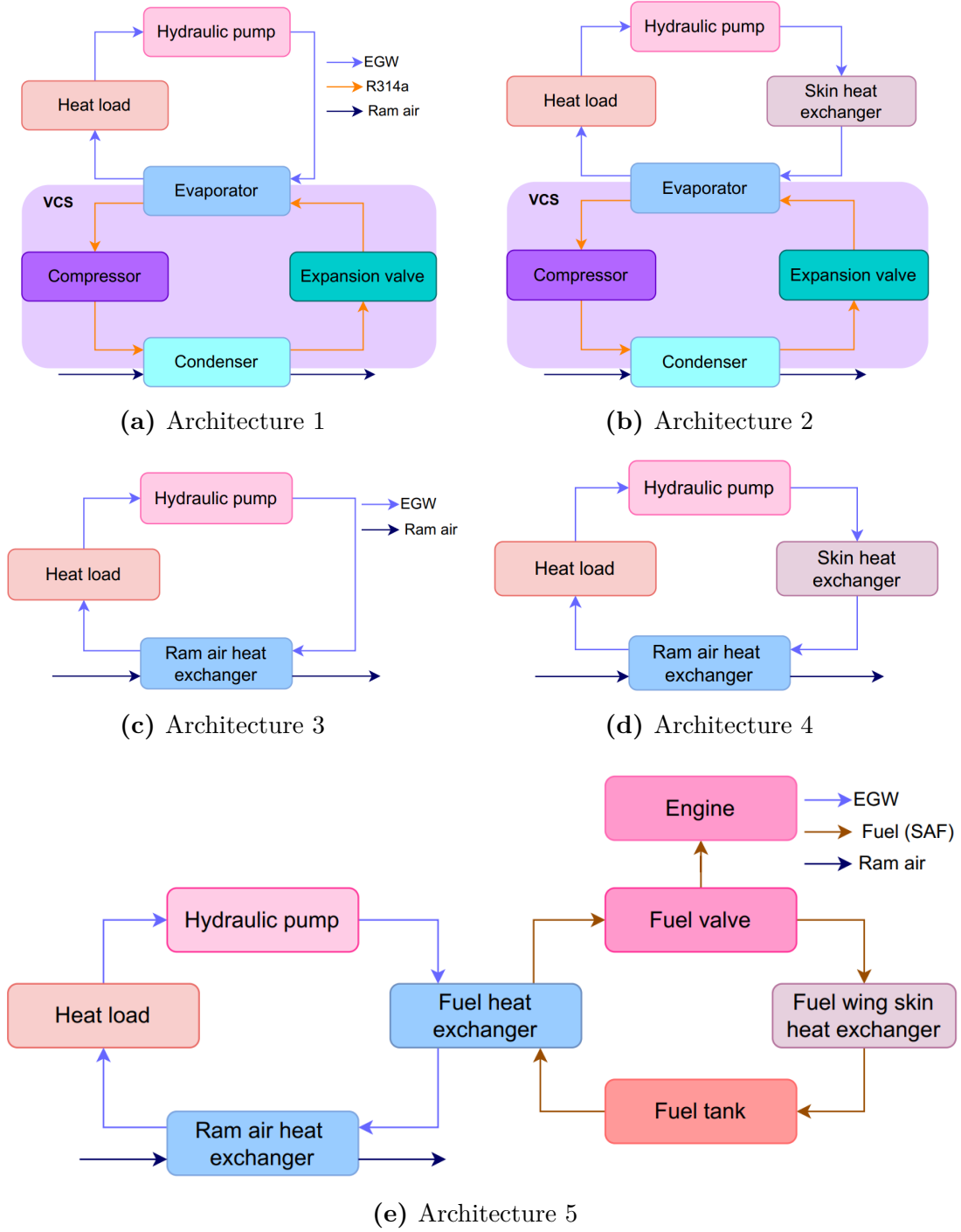


Figure 2.1: Five proposed architectures by Coutinho *et al.* Schematic diagram of the loops and their main components [5]

None of the investigated architectures yielded to acceptable results, highlighting the need for more efficient RHX and the possibility of installing more SHXs, also due to their drag reduction effect.

An important gap from this research must be highlighted: only static simulations are performed. This means imposing constant heat load and boundary conditions, not considering thermal inertia, cycles and abrupt variables value changes. The proposed workflow is valid, but this missing point forces to further studies in order to perform a reliable analysis.

2.2 Hybrid-Electric Aircraft

Due to the increasing necessity of *green* technologies, several electrical propulsion systems have been developed. The reference aircraft for this thesis is *ODE4HERA* project, a hybrid-electric regional aircraft [4]. Hybrid-Electric propulsion relies on an Internal Combustion Engine (ICE) to generate power, either as an electricity generator, moving the propeller shaft or by recharging batteries [6]. IATA (International Air Transport Association) defined a roadmap to achieve CO_2 net-zero aviation in 2050 [20]. The roadmap cites hybrid-electric propulsion and divides it in five categories:

- Series Hybrid (figure 2.2): electrical motors are linked to the fans, while the ICE runs an electrical generator;
- Parallel Hybrid (figure 2.3): ICE and battery-powered electric motors are connected to different shafts. Shafts are then coupled into a single propulsive shaft;
- Series-Parallel Hybrid (figure 2.4): combination of the earlier technologies. Some fans are powered by parallel architectures, others by series ones;
- Turboelectric (figure 2.5): similar to the series architecture, but the electrical motors are powered by ICEs, not batteries;
- All-electric (figure 2.6): complete reliance on battery-powered electric motors.

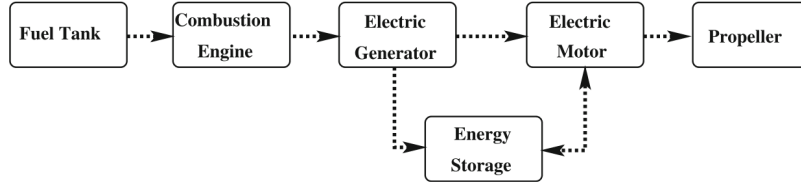


Figure 2.2: Series Hybrid architecture block diagram. Blocks indicate system components and arrows indicate power transfers [6]

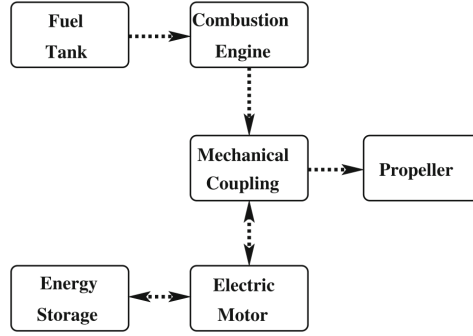


Figure 2.3: Parallel Hybrid architecture block diagram. Blocks indicate system components and arrows indicate power transfers [6]

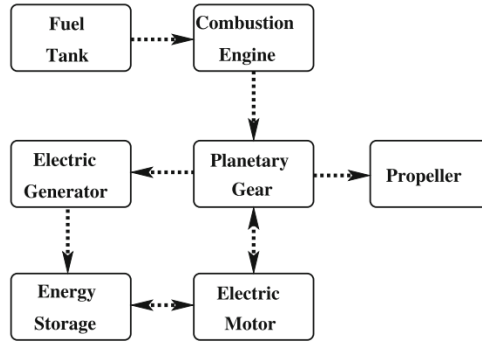


Figure 2.4: Series-Parallel Hybrid architecture block diagram. Blocks indicate system components and arrows indicate power transfers [6]

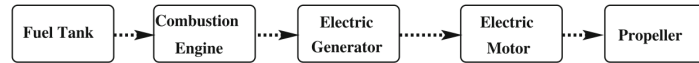


Figure 2.5: Turboelectric Hybrid architecture block diagram. Blocks indicate system components and arrows indicate power transfers [6]

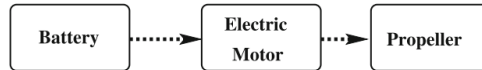


Figure 2.6: All-Electric Hybrid architecture block diagram. Blocks indicate system components and arrows indicate power transfers [6]

Turboelectric architecture results the most promising option for hybrid-electric aircraft designs [6]. Following these indications, a possible entry-into-service target could be 2035-2050 [6], respecting IATA indications [20].

Hybrid-electric technologies and design methods can be divided in several topics [21]:

1. electric-powered architectures classification: conventional, all-electric, turbo-electric, series hybrid, parallel hybrid;
2. key technological parameters: specific energy, specific power, total electric efficiency. In particular, the authors observed that an improved efficiency doesn't really helps in improving total range, but can reduce by 60 – 70% waste heat;
3. metrics: environmental parameters and main constraints like range, takeoff length, noise, NO_x ;
4. technological demonstrators and studies: Pipistrel Alpha Electro, Siemens Extra 330LE, etc.;
5. aircraft-level benefits: direct benefits, propulsion, aerodynamics, safety, weight;
6. electric architecture: components, improvements on materials, engines, AC-DC converters;
7. batteries: trade-off between specific energy and specific power requests;
8. thermal management: technical hurdles;
9. MBD and MDO: lack of high-fidelity public frameworks that can couple these aspects.

The main technological hurdles are specific power and specific energy values of components [21]. For this reason, only turbo-electric single aisle aircraft are realistic for a 2035 entry into service, constraining all-electric architectures to short-range projects only. A better MBD-MDO integration is necessary.

It is possible to explore preliminary concepts for HER aircraft, trying to align emission levels to future targets. The main focus is the application of an increasing Degree of Hybridization for Useful Power, the power fraction electrically produced [22].

By integrating the electric power units in the preliminary design, a reduction in fuel burn around 30% can be achieved, contrasted by a sensible mass increase.

This result leads to the conclusion that this kind of architectures would be better suitable to regional aircraft.

Gohadrani *et al.* [23] explore the topic of electric and hybrid propulsion for fixed-wing aircraft, focusing on thermal and aerodynamic aspects. It represents the main reference for the concept of Distributed Electric Propulsion (DEP). DEP idea is distributing the propulsion in several points of the aircraft, contrasting the current design sight which concentrates thrust in few (generally from 1 to 4) big engines. Electrical propulsion allows to distribute more the thrust, opening the way to DEP concepts. The most interesting aspect of this study is the hybrid-electric implementation benefits analysis. It can be divided in aerodynamic, acoustic and system-level benefits. These improvements also face some technical challenges, as battery energy density, thermal management, weight, reliability, noise and vibrations. Among all these, the main two challenges are similar to the other papers' results, highlighting a trend: energy and thermal management. To solve these problems, a more-integrated design process is required.

2.3 Thermal Management System

Thermal Management Systems are acquiring more importance thanks to the introduction of more-electric aircraft. Electrical components create, due to Joule effect, an amount of thermal energy proportional to the electrical energy received/produced. The more electrical energy is managed, the more waste heat is produced. This fact is particularly relevant in an environment like an airplane, where spaces are tight and everything must be precisely embedded.

Four main aspects contribute to waste heat generation [24]:

- engines/generators: 5 – 10% of generated electrical power results in waste heat;
- inverter and power converters: 2 – 5% of electrical power results in waste heat;
- batteries: they generate waste heat during charge and discharge processes;
- auxiliary components: cables, actuators and other components that are affected by Joule effect.

In traditional combustion engines, the air flow used to produce power is also the main heat rejection system. It is a "free" rejection, because the heated air mass will flow out of the engine anyway, but it helps reducing in a drastic way the waste heat to reject with a TMS [25]. The only possible effect is related to aerodynamic drag due to different temperatures in air flows. In traditional combustion engines, around 54% of chemical power from the fuel is lost in waste heat [25], while only 15% of electrical power would be lost with HER architectures [24]. Increasing propulsive electrical powers requires more heat dissipation due to the direct relationship between propulsive capacity and waste heat.

Another aspect to consider is the variability of thermal loads: due to their strong interconnection with required propulsive power, they also present high variability during the mission: if for a traditional propulsive system this variability would be noticed in terms of fuel flow fed to the engine, in HER architectures, several *kW* of waste heat differentiate phases, requiring dynamic simulations to fully comprehend and design the system.

Several technologies have been explored for heat dissipation [11]:

- air cooling: atmospheric air is the main heat sink. The most used form is based on ram air, which brought in contact with a heat exchanger wall absorbs heat and then gets rejected into atmosphere (open loop), just like in traditional engines. It provides a good heat rejection system, but adds weight and drag;

- liquid cooling: uses liquid closed loops to absorb and reject heat to a heat sink with some forms of heat exchangers. They can also be Vapour Compression Systems, which take advantage of evaporation and condensation of refrigerants inside the loop to dissipate heat;
- fuel as heat sink: fuel can be the intermediate thermal sink, absorbing enough heat to reach its engine-entry optimal temperature and then rejecting the rest of the heat in the same way of refrigerant fluids.

It is possible to classify TMSs in 4 categories depending on the novelty and technological level [26]: classic, centralized, revolutionary and cost-effect. Going through the innovation process, several aspects must be considered [26]:

- altitude effect: less dense air at higher altitudes reduces heat exchange;
- extreme conditions: hot days and cold days;
- mission profiles and transients;
- nowadays TMS architectures are at their maximum development level: a disruptive technological change is necessary for the future;
- future aircraft will need lighter, more integrated and smarter TMS;
- thermal management will become a primary design driver in civil aviation starting from 2035.

TMS will become a future core system of the aircraft, requiring a more integrated and optimized design process that can communicate with both the propulsive and the electrical system [27]. New fluids and technologies will be necessary, starting to plan them from the preliminary design phase [27].

Affonso Jr. *et al.* [28] give an overview on technical challenges that must be overcome for future TMS development. They focus on design impact on weights, powers and safety. They also develop a map based on TRL (Technology Readiness Level) for TMS technologies, proposing an integrated approach between exergy analysis (ability to extract the electrical power produced and transform it into work) and MDO workflows. The main necessities in future designs are:

- batteries thermal runaway;
- failure propagation;
- reliability and maintainability;

- certification and equivalence of safety with existing technologies.

The interesting aspect of the study is the novelty of challenges: the focus is moved towards the entry into service of the technology, not just its design. The response to innovative challenges is a group of disruptive technologies, such as thermal energy reuse, energy harvesting, smart heat sinks, thermal accumulators and energy-based optimization.

Several heat sinks can be considered [29]: fuel, ram air, engine fan air, SHX, expendable heat sink. Focusing on fuel, ram air and SHX the results are:

- fuel: the most effective but dependant on residual fuel quantity;
- ram air: penalized by drag;
- SHX: minimum penalty but limited capacity.

A quantitative criterion is proposed to size and balance the heat sinks in the design phase.

Focusing on Ram-Air cooling, it results from Kellerman *et al.* [30] that RA cooling for a TMS is a realistic solution, with geometrical integration as the main challenge.

Architecture 5 from Coutinho [5] comes from Figueiras *et al.* study [31]. It relies on a fuel loop to extract heat from the EGW loop and reject it through a SHX. Selected fuel is a SAF, specifically an ATJ (Alcohol-To-Jet). Liquid cooling with RHX as the most technologically ready solution. SHX and FHX are the most promising components to apply it [11].

SHX can also be located on fuselage skin [32]. SHX integration in both wings and fuselage can also lead to a reduction of drag due to wall (skin surface) heating. In this perspective, a SHX system would reduce aerodynamic drag, increasing overall efficiency [33].

2.4 TMS Components Sizing

The sizing process started by studying two manuals from Professor Sergio Chiesa (Politecnico di Torino) regarding hydraulic system [34] and fuel system [35] design. These manuals have been integrated with the literature and notes derived from three Politecnico di Torino Aerospace Engineering Bachelor's and Master's degrees courses: *Sistemi di Bordo Aerospaziali* (Prof. Paolo Maggiore), *Sistemi Aerospaziali* (Prof. Nicole Viola) and *Progetto dei Sistemi Aerospaziali Integrati* (Prof. Nicole Viola).

This literature has been essential to deeply understand hydraulic and fuel systems. Another Politecnico di Torino Aerospace Engineering MSc course (and its related literature) has been important in the modeling process: *Modellazione, Simulazione e Sperimentazione dei Sistemi Aerospaziali* from Prof. Matteo D. L. Dalla Vedova.

Concerning pumps sizing, Dixon and Hall handbook [36] has been the core reference in the definition of empirical mathematical relations between pumps functioning parameters and their weight. This manual clarifies how pumps masses can be derived with only their working parameters as pressure differentials and mass flows. Uncertainty must be considered.

Concerning fans and compressors, scaling laws have been used. Negoita work [37] provides a workflow to build scaling laws that allow, starting from an existing component, to scale it with sufficient accuracy in all its available (from datasheet) working values. Datasheets regarding plausible candidate components are needed, possibly more than one in order to achieve a better statistical population.

For powers estimation, the main references are Coutinho's works [5] [11]. They provide ready-to-use, simple relationships that link components' working parameters to their power consumption. They start from fluid-mechanics values, defining necessary power to move a given amount of fluid with a defined speed (mass flow and pressure drop). Then, latter is transformed into a shaft mechanical power and, finally, into the electrical power that the shaft needs to be moved at that regime. In every step of this procedure fluid, mechanical and electrical efficiency are considered.

Uncertainty must be considered. Safety factors are not defined a priori by coded laws, they are considered a best practice that may change depending on the stage of the process. Best references regarding this topic are official agencies: NASA (National Aeronautics and Space Administration) [38], [39] and ESA (European Space Agency) [40]. These references highlight some common best practices for aerospace designs. They don't prescribe any mandatory value, but instead they suggest one depending on the design process stage. NASA handbooks mostly focus on masses estimations, while the ESA report goes through all design aspects. Both documents show similar values (around 20 – 30%) for the safety factor, remarking the reliability of the sources.

2.5 Optimization in Aircraft TMS

Aircraft Thermal Management System design is a system level problem that requires integrating studies about thermal loads, mechanics, masses, power consumption and operating conditions. Traditional sequential and margin-based design processes (D. P. Raymer textbook [41] is a good example) have become less prevalent in favour of innovative ones, such as Multi-Disciplinary Optimization. With this new framework, traditional methods for aircraft design are embedded in a multi-loop process. The aim of the process is to minimize or maximize certain values, called objective variables [42]. System-level optimization techniques allow a more systematic exploration of design trade-offs [42].

2.5.1 Design Of Experiments

Applying MDO to high-fidelity dynamic models is often difficult because of the elevated computational cost required. An optimization algorithm usually requires multiple simulations to find the best configurations. Depending on the fidelity level required and the selected algorithm, necessary simulations number may vary from tenths to thousands.

To address this problem, a Design Of Experiments (DOE) can be useful. It is meant to explore the design space by sampling it with mathematical methods for a defined number of points. T. W. Simpson *et al.* [43] apply DOEs to computer science design to explore the design space and highlight the best areas of the latter. It is possible to describe and map system behaviour, highlighting better and more feasible areas inside the defined design space.

To build a DOE, a mathematical criterion is necessary. The most used one in aerospace field is Latin Hypercube Sampling (LHS) [44]. LHS has been initially developed for computer experiments, high-cost deterministic simulations in which the number of evaluations is limited. The objective is to efficiently sample a multi-dimensional space ensuring uniform coverage, reduction of statistic variance and a reduced number of samples. Every design variable is sampled in an uniform way through its interval [44].

In practice, each variable interval is divided in N iso-probability intervals, each one gives a single value. The values from all the intervals are randomly combined [44]. The result of this process is a set of points with uniform mono-dimensional projections and randomized multi-dimensional combinations [44].

2.5.2 Surrogate Modeling

DOEs are used to build surrogate models. These models are built starting from a system input-output exploration and can define with good fidelity the system

behaviour [43] [45]. The major advantage of using surrogate models is that, once built, they don't need any run of the system model. Thanks to this feature, the optimization process can be based not on heavy, time-consuming simulations but on mathematical models, with negligible computational times. The branch of MDO that focuses on surrogate models is called Surrogate-Based Optimization (SBO). This method represents a link between model-based design and optimization, allowing an optimization run with reasonable runtimes but still backed by scientific evidences that confirm its reliability. Many SBO studies have been performed by ONERA (Office National d'Études et de Recherches Aéronautiques) [46], giving insights of DOEs and surrogate models as solution of computational high-cost optimization problems. Reliability of the workflow is also assessed. One of the most interesting scientific productions available online is **SBArchOpt** Python library [47]. This open-source library wraps other Python libraries, adding a focus on surrogate-based optimization methods and tools.

Another good reference for Surrogate-Based Analysis and Optimization (SBAO) comes from Queipo work [48]. It describes an entire workflow composed by DOE, surrogate definition, model validation, sensitivity analysis and optimization. This path covers both model-based design and MDO aspects, being one of the most used in current scientific literature.

A relevant problem inside the design space are the so called "hidden constraints". J. H. Bussemaker *et al.* [45] describe them as *evaluation failures, which are also known as unknown, unspecified, forgotten, virtual, and crash constraints* [47]. They consist in failed simulations because of unfeasible geometries or non-convergence of the solver. They are called *hidden* because they're not known before running the simulation, but once they show up they mark an unfeasible area of the design space. Dealing with hidden constraints inside DOEs is difficult, because they are invalid points for the design space exploration. For this reason, Bussemaker work [47] made use of the Probability of Viability (PoV) function.

PoV function is a kind of surrogate function that *predicts where the failed region lies. The idea is to assign binary labels to all design points based on their failure status, for example 0 meaning failed and 1 meaning viable, train a model on this data, and then use that model to predict the so-called PoV. PoV lies between 0% and 100% and represents the probability that a newly-selected infill point will be viable (i.e. will not fail)* [47].

Multiple methods exist to evaluate a PoV function: Random Forest, Radial Basis, Gaussian process, etc. [47].

By using PoV function, it is possible to understand where the system fails and try to acknowledge this information into the surrogate model construction, transforming failures into useful points.

2.5.3 Surrogate-Based MDO

In aeronautical complex systems, performing MDO with surrogates is not just about optimizing one function: the process needs to embed all stages, including DOE and surrogate models. Every step is meaningful for the final optimization algorithm, because it takes into account failures, probabilities and input-output variations. For this reason, a high integration level is needed, aiming to design a complete framework. Several steps must be considered in this process [48] [46]. The common structure of the process is [48]:

1. design space definition;
2. DOE and high-fidelity evaluations;
3. surrogate construction;
4. surrogate optimization;
5. validation and refinement.

The process must be highly integrated in all its parts, considering possible architecture variations [49]. This approach is called Dynamic MDAO (Multi-Disciplinary Architecture Optimization). It allows having different components among the architectures under exam.

As stated in earlier paragraphs, surrogate-based optimization is a key factor in allowing faster and more detailed research: reducing computational cost can be the main technological leverage to enable strongly optimized designs. It can save up to 94% of computational cost with highly reliable results, validated with actual re-simulations, resulting in excellent results even with a relative error threshold of just 5% [50].

Unfortunately, many frameworks face the problem of hidden constraints: in real-world applications, they must be the main driver for the optimization problem. By using PoV functions [47] during the entirety of the optimization process, the designer can strongly rise his confidence level in final results, because they have been filtered by a reliable tool.

Chapter 3

Methodology

The objective of the proposed methodology is to enable and facilitate dynamic simulation and sizing optimization for the Thermal Management System of a Hybrid-Electric Regional Aircraft.

3.1 Model-Based Design

The modeling process was carried out with the OpenModelica software.

OpenModelica is based on the Modelica language, an *object-oriented language for modeling of cyber-physical systems, supporting a-causal mathematical equations to facilitate modeling from first principles. It provides object-oriented constructs that facilitate reuse of models, and can be used conveniently for modeling complex systems containing, e.g., mechanical, electrical, electronic, magnetic, hydraulic, thermal, control, electric power or process-oriented sub-components* [51].

Models are described, from a user point of view, by schematics, also called object diagrams [52]. In the schematic, components are connected by connectors or ports to describe mutual interactions. Each component is defined by other schematics or by equations written in a specific Modelica syntax.

The first step in Modelica history has been MIMIC language, developed for Control Data super-computers in 1964. Several languages followed, such as CSMP and ACSL in the 1980s. Then, Dymola language was developed by Hilding Elmqvist from Lund University. It was a programming language that could handle non-causal equations, one step ahead of MIMIC language (figure 3.1). In the sake of a higher-level programming language, an object-oriented language was developed in 1996: Modelica language. Dymola became a Modeling and Simulation environment for Modelica language. With Modelica language, there was no more need to derive the state-space form, but instead it was possible to directly punch in the model

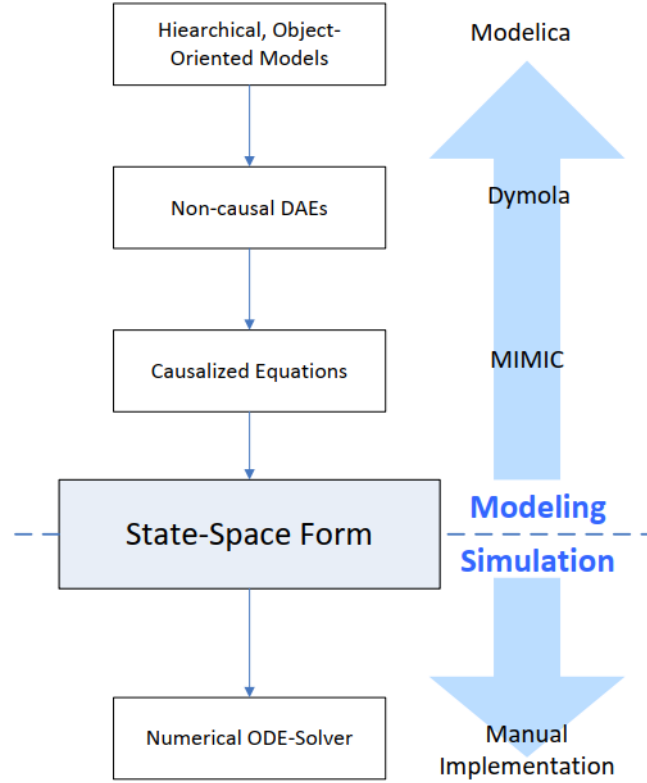


Figure 3.1: Different approaches to system simulation. Blocks: mathematical tools. Blue arrows: simulation language/method [7]

equations [7].

Focusing on figure 3.1, there are two ways of reading the blue arrow:

- on the way up, system modeling becomes a sole matter of physics: there is no need to know computer science. It is just necessary to learn how to formulate the laws of physics in an object-oriented way;
- on the way down, the need to understand how languages are compiled increases. Physics necessary knowledge decreases, while computer science required ability rises. The state-space form is still derived automatically in MIMIC.

The model is represented as a system of discrete, algebraic and differential equations.

During this research work, the Modelica language has been used inside the OpenModelica software. OpenModelica is an open-source modeling and simulation

environment for Modelica language [53]. It is developed by the Open Source Modelica Consortium. It allows both graphical and written model definition in a Dymola-like user interface. The most important components of OpenModelica are:

- OMC: OpenModelica Model Compiler. Translates Modelica models into simulation code, which is compiled and executed to perform simulations [53];
- OMEdit: OpenModelica Graphic Model Editor and Simulator Graphical User Interface (GUI). OMEdit is the OpenModelica graphical model editor for component-based model design. It works by connecting instances of Modelica classes. The editor also provides text editing. Moreover, the OMEdit GUI provides a graphical user interface for simulation and plotting [53].

3.1.1 Modelica libraries

One of the most interesting aspects of Modelica language is the ability to create and use libraries. These libraries contain ready-made blocks that can facilitate the modelling work.

Modelica

The main library for any Modelica environment is **Modelica**.

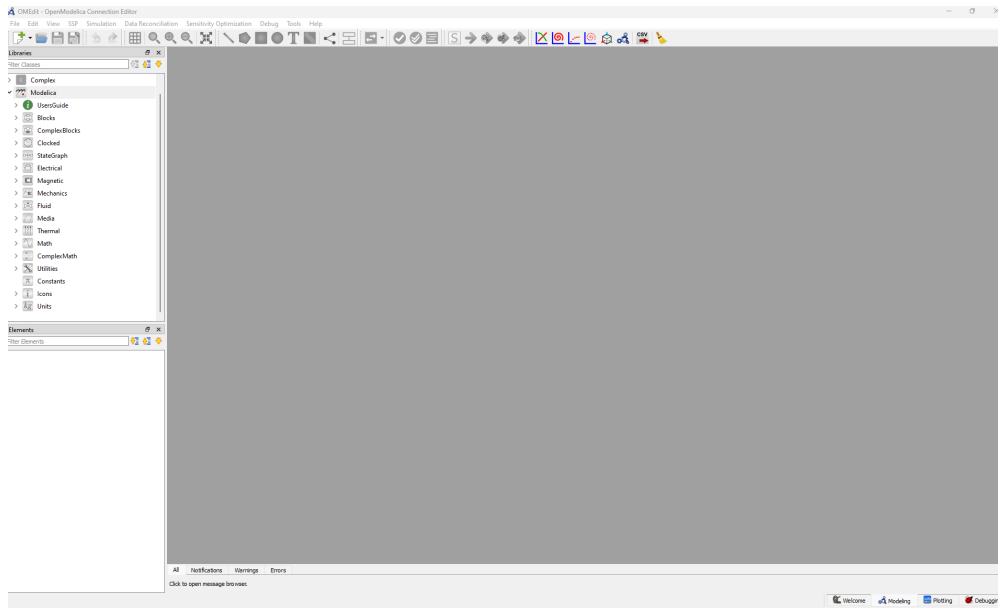


Figure 3.2: OMEdit opening page with the Modelica library on the left hand side

As shown in figure 3.2, **Modelica** library contains several categories, such as **Electrical**, **Magnetic**, **Mechanics**, **Fluid**, but also signal and mathematical categories like **Blocks**. As an example, figure 3.3 shows the sub-category **Blocks** from **Modelica** library.

Opening the **Continuous** sub-sub-category, the first blocks appears. As the name suggests, they all represent continuous functions or blocks that work in the continuous field. These blocks are then used to compose the model. Each of these represent a specific function or set of functions used to model the expected behaviour of the component. For instance, the **FirstOrder** block represents a first order transfer function with one pole. It can be used whether as a filter or to simulate a physical component dynamics. Its textual definition is shown in figure 3.4.

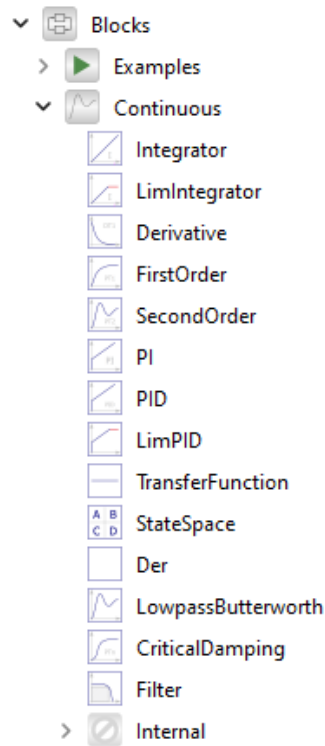


Figure 3.3: Blocks sub-library. As an example, Continuous sub-category has been opened, showing all its related elements

```

351 block FirstOrder "First order transfer function block (= 1 pole)"
352 import Modelica.Blocks.Types.Init;
353 parameter Real k(unit="1")=1 "Gain";
354 parameter SI.Time T(start=1) "Time Constant";
355 parameter Init initType=Init.NoInit
356   "Type of initialization (1: no init, 2: steady state, 3/4: initial output)" annotation(Evaluate=true,
357   Dialog(group="Initialization"));
358 parameter Real y_start=0 "Initial or guess value of output (= state)"
359   annotation(Dialog(group="Initialization"));
360
361 extends Interfaces.SISO(y(start=y_start));
362
363 initial equation
364   if initType == Init.SteadyState then
365     der(y) = 0;
366   elseif initType == Init.InitialState or initType == Init.InitialOutput then
367     y = y_start;
368   end if;
369 equation
370   der(y) = (k*u - y)/T;
371   annotation ( .. );
423 end FirstOrder;
424

```

Figure 3.4: FirstOrder block on Modelica textual notation

ThermofluidStream

The second (and probably the most important) of the two libraries used for this work is **ThermofluidStream** (TFS) from DLR Institute of System Dynamics and Control [54], shown in figure 3.1.1. TFS library is defined as *a free open-source library for the robust modeling of complex thermofluid architectures* [8].

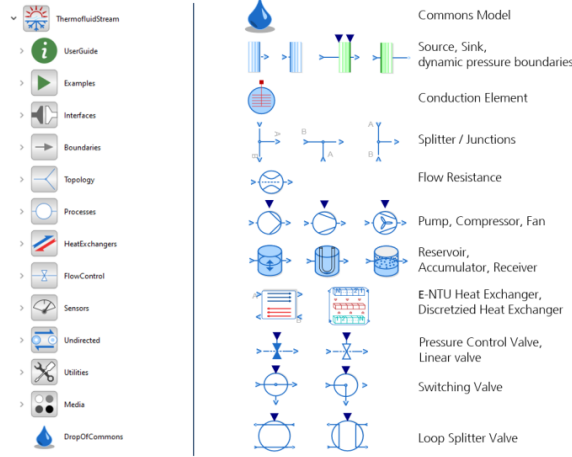


Figure 3.5: Overview of **ThermofluidStream** library and its main packages on the left side and key components on the right side [8]

This library has been designed to be an easy and reliable tool in the design of thermofluid systems, taking into account both thermodynamics and fluid dynamics and their interactions.

One of the key novelties of this library is the explicit use of *fluid inertia* in order to avoid large non-linear implicit systems and therefore improve simulation reliability.

1D flow systems are re-formulated by introducing the *inertial pressure* r and the *inertance* L [8]:

$$L = \int \frac{ds}{a} \quad (3.1)$$

with s being the length of the flow and A its cross-sectional area. Mass-flow rate $\frac{d\dot{m}}{dt}$ is a state variable of the system. Each component in the flow loop expresses a differential equation like:

$$\frac{d\dot{m}}{dt} \cdot L = -\Delta r \quad (3.2)$$

Once acknowledged the concept of *flow inertia*, a new definition of the pressure gradient Δp is possible:

$$\Delta p = \Delta \hat{p} + \Delta r \quad (3.3)$$

with $\Delta \hat{p}$ being the steady mass-flow pressure ($\hat{p} = p$ when $\frac{dm}{dt} = 0$). By doing so, it is possible to approximate $\hat{p} = p$ even in unsteady conditions by accepting a small error, while compensating Δr at the boundaries. The resulting structure for the equation system is more simple in terms of computation, because all the non-linearities can be brought to the explicit form, leaving in the implicit form only linear equations.

Three main aspects must be considered during the implementation in Modelica language:

- directional connectors: r , $\dot{m}$ and the thermodynamic variable are transmitted as signals;
- mass flow initialization: all components start at $\dot{m} = 0$. Equilibrium is guaranteed by r gradients;
- pressure gradients: to ensure plausibility to the algorithm (avoiding unexpected negative mass flows), the outlet pressure has a minimum value that avoids, due to mass flow sudden variations, negative outlet pressures.

There are six main types of components in TFS library:

1. volumes: at least one volume is always required in a loop. Volumes break algebraic loops and define system dynamics with energy and mass balances. They also have a damping function on variables;
2. turbo-machinery (pumps, compressors, fans, turbines): they follow performance maps defined by functions, including mechanical dynamics;
3. heat exchangers: they can be single-phase with the $\varepsilon - NTU$ (Number of Transfer Units) method or multi-phase, discretized into N cells;
4. valves: they are defined by a pressure drop, dependant on the valve characteristics;
5. sensors: they provide physical measurements and additional functions as output filtering and sensor delays;
6. **DropOfCommons** block: defines common values for all one-dimensional flow components. It creates a common framework.

3.1.2 OpenModelica Models Definition

OpenModelica environment was used to define the models. First thing that a user sees by launching OMEditor (the editing software) is the home window shown in figure 3.6.

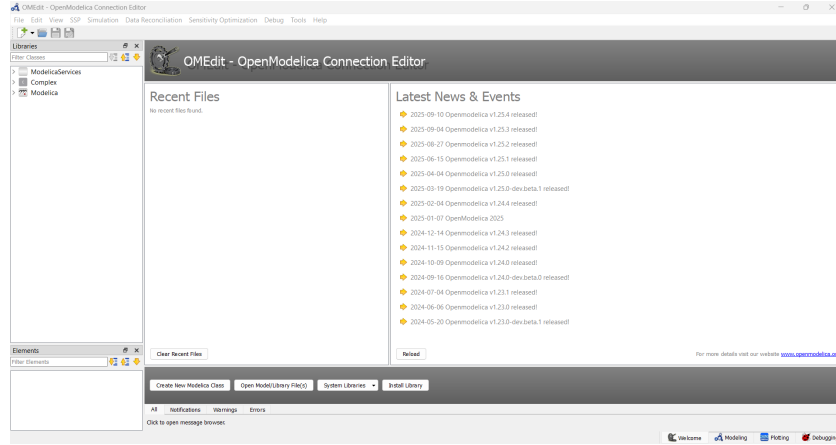


Figure 3.6: OpenModelica Connection Editor home view. This is the first view shown to an user when the software is launched

By clicking on the "New" button on the top left side, the editing window will open. In this window, the user can directly drag the blocks from the libraries, place them and then create the connections between inputs and outputs. An example is shown in figure 3.7.

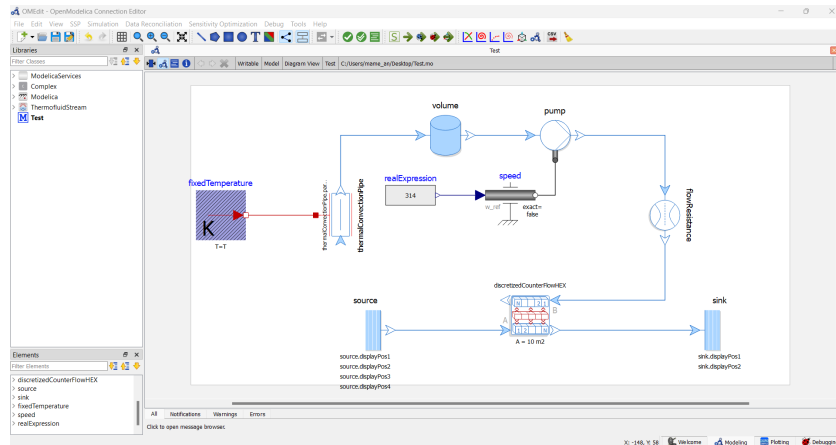


Figure 3.7: Test model in OMEditor. An example of model construction is shown, with blocks from both ThermofluidStream and Modelica libraries

The example shows a loop built with both `Modelica` and `ThermoFluidStream` libraries. It is a cooling loop: the `fixedTemperature` block imposes a temperature to a `thermalConvectionPipe` block, which transfers the heat from the heat load to the fluid in the loop. The fluid then passes through a `pump` block, which changes its stream conditions. The pump is driven by a shaft, represented by the `speed` block, which receives its speed command as a signal. In this simple case, the signal is a constant value, but it usually is a combination of constants, sensor measurements, filters and PID.

The flow then passes through a `flowResistance` block, which simulates the pressure losses in piping. Finally, the fluid can discharge the absorbed heat in a heat exchanger, in this case a `discretizedCounterFlowHEX` block. As explained in section 3.1.1, a heat exchanger may be single-phase or multi-phase. In this case, the heat exchanger is a multi-phase one, with an N number of discretization cells to be set by the user.

The heat exchanger extracts the heat from the cooling fluid and rejects it to another fluid. In this case, the latter is external air. External air cycle is shown in its simplest form, with just a **source** (the atmosphere, defined by temperature and pressure) and a **sink**, only defined by pressure.

It is important to underline the importance of the `volume` block. It represents the amount of fluid in the loop. As mentioned before, the use of at least one volume (in any form from the libraries) is mandatory for every loop.

The model can be edited and shown in text view as well (figure 3.8).

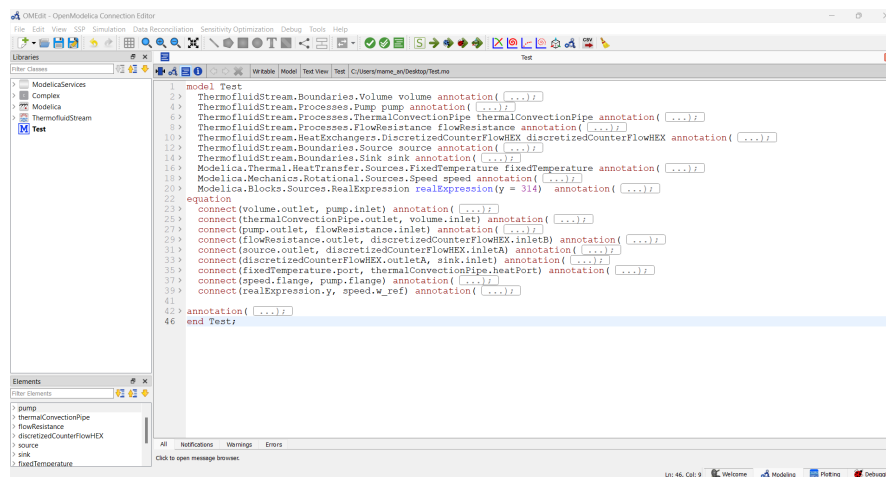


Figure 3.8: Test model from figure 3.7 in Modelica textual notation

Text view is especially useful when the user needs to check and fix connections, setup values and wrong initialization data.

Once created the model, the setup phase begins. Each block (or most of them)

must be set with plausible and appropriated values. Taking as an example the pump block, the user can double-click on it and the window shown in figure 3.9 appears.

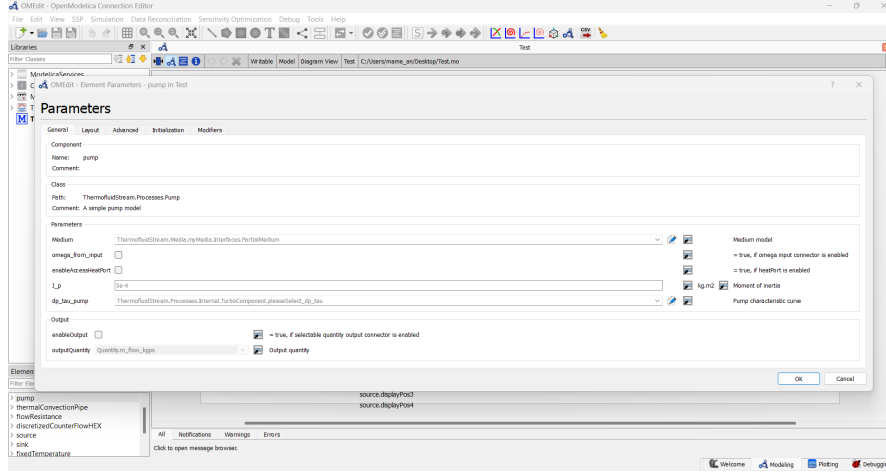


Figure 3.9: Parameters window for pump block. the rows can be filled with desired values

In this window, the user can browse around different tabs, such as *General*, *Advanced* or *Initialization*. In each tab, depending on the component, several parameters need to be filled. In the pump case, one of the most important choices is for the `dp_tau` function. This is the function that defines the performance map to be used by the component. One of the most delicate aspects for any component is the initialization. A wrong, underestimated, overestimated or absent initialization value for some variables can bring instability to the system, resulting sometimes in crashed simulations, even without a physical cause behind. In this research work, the following initialization has been followed: for each loop, the only component that calls an initialization is the turbo-machinery one. This avoids multiple initialization that may stiffen the system.

Compiling phase is performed by the OMC Compiler, the compiling tool available with OpenModelica. OMC translates the Modelica code in an executable code (C/C++ language) with an hybrid symbolic-numeric compilation workflow. The first step is the symbolic analysis to optimize the equation system, then the numerical analysis solves the ODE system [53].

The main phases of the compilation are:

1. parsing and syntactical analysis: `.mo` (the typical Modelica file) file is read by the parser. An Abstract Syntax Tree is built and syntactical errors are found;
2. semantic analysis and translation in SCode: the semantic analyzer checks

meaning and coherence of the code and creates an SCode (Structured Code), an intermediate form of code;

3. flattening: all classes, instances and connections are transformed in a flat mathematical model. This means that all the components' instances become equations, as well as the connections;
4. symbolic optimization and flattening: the flat model is optimized and re-organized in a well-structured form;
5. code generation: equations are converted in C/C++ code. OMC generates: residual equations code, Jacobian code, initialization and homotopy code, user function and time derivatives code;
6. final compilation: the code is sent to a system compiler and an `.exe` executable file is produced, which can interact with the OpenModelica runtime;
7. execution and interfaces: the executable file interacts with numerical integrators, event handling, profiler and debugger, OMEdit and script.

Once the executable is compiled, the solver performs the simulation. It deals with discrete events and DAEs (Differential-Algebraic Equations). In this phase the output file is produced. It can be in `.mat` or `.csv` format. In a sake of simplicity for the following Python implementation, the results will be written in a `.csv` file. Otherwise, if the user is running the models on its own in OMEdit, the easiest option would be the `.mat` format.

Once the simulation reached 100%, the user can plot the results in the *Plotting* window (figure 3.10). Here, on the right hand side of the screen, the user can choose the variable among a list divided by component. The choice will always be on the y axis, because the x axis will always represent the time in the standard plot mode. It is possible to have more plot options. It is also possible to plot multiple variables in the same plot.

The last aspect an user should take care of while running a simulation, is the *Simulation Setup* window (figure 3.11).

In this window, the user can define the simulation parameters, especially start and finish time and the number (or the size) of intervals in which the program saves the data. Being a variable-step simulation, the interval size is not the actual ODE time step size, but just a data saving time.

Integration method and tolerance choice can be essential to reduce the runtime. For this work, the integration method is *dassl* and the tolerance is 1e-06. This tolerance value is strict and stiffens some of the models, but doesn't sensibly change the runtime. For this reason, a higher-precision approach with low tolerances has been preferred.

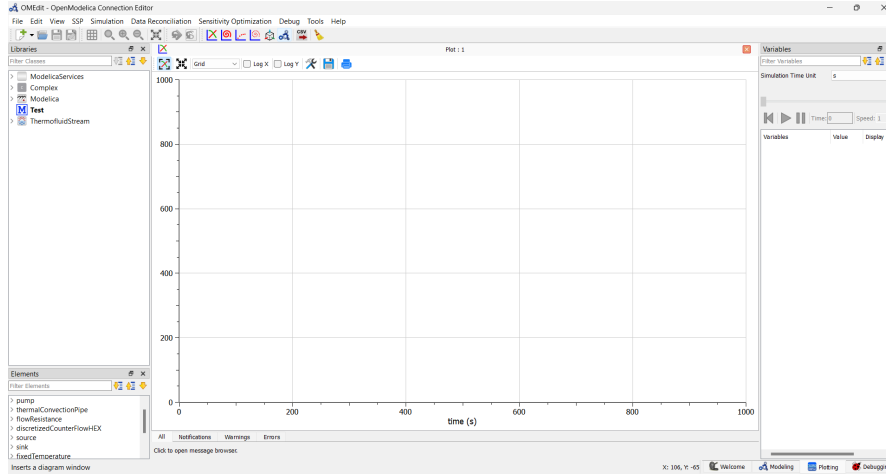


Figure 3.10: Plotting window in OpenModelica. On the right hand side, a list of all the plottable variables will appear, allowing the user to choose which variables he wants to plot

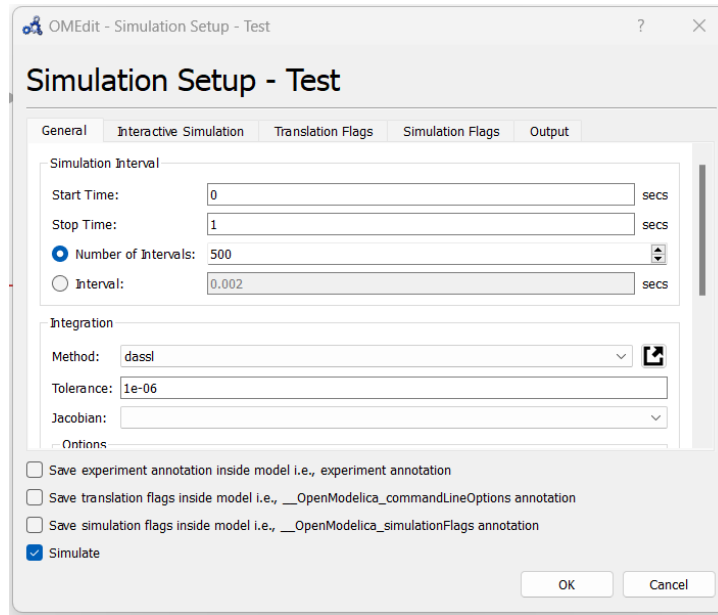


Figure 3.11: Simulation Setup window. This window allows the user to define the simulation parameters like start and end time or mathematical method

3.1.3 Modelica Models Construction

The idea behind models construction for this thesis work has been increasing modularity: all five architectures share high percentages of common components,

enabling for a common building process.

Table 3.1: Common components/loops for the 5 architectures

Architecture	Heat Load	EGW Loop	Ram Air Cooling	SHX	VCS Loop	Fuel Loop
1	X	X	X		X	
2	X	X	X	X	X	
3	X	X	X			
4	X	X	X	X		
5	X	X	X	X		X

As shown in table 3.1, all architectures share at least half of the main components/loops. In particular, they all have a Heat Load, an EGW loop and a Ram-Air cooling.

Heat load

At this point of the work, the Heat Load will be considered as a black box, it will be clarified in the following section. The heat load is defined as a `.txt` table with two columns (time and heat load) and a row for each second of the mission time. The Modelica block used to read the external table is `CombiTimeTable`, which reads the table and provides a time-dependant output. This output is then filtered by a `FirstOrder` block, which serves as a filter to have a "smoother" and more realistic heat load curve, as shown in figure 3.13.

Once filtered the signal, the heat exchange section starts. It has been built with three components. They are, in order of appearance:

1. **PrescribedHeatFlow**: provides boundary conditions for the heat exchange and transforms a signal coming from a table in physical heat;
2. **HeatCapacitor**: represents a physical element with its thermal properties, including thermal capacity;
3. **ThermalConductor**: simulates thermal conduction for a physical element.

The numerical values have been chosen to simulate a DURAL surface of around 8.8 m^2 and 1.5 mm of thickness. There is no sensible distinction between a flat plate and a more complex one due to the absence of thermal convection: it only is a transfer of heat due to contact with the heat source and, in a simplification, steady air which surrounds it.

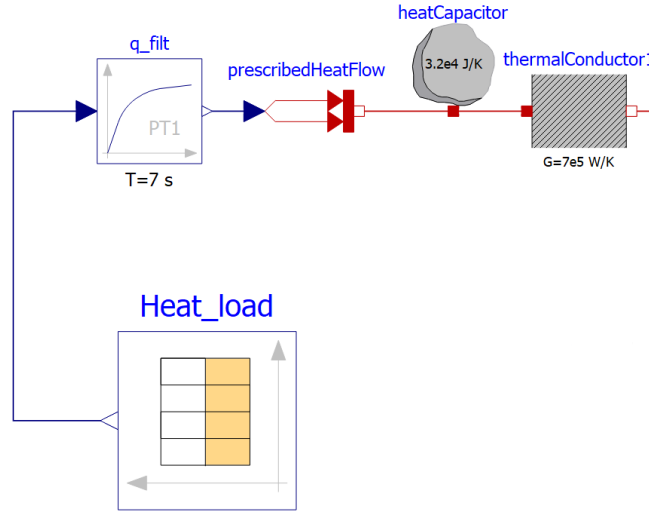


Figure 3.12: Heat load model in Modelica graphical notation

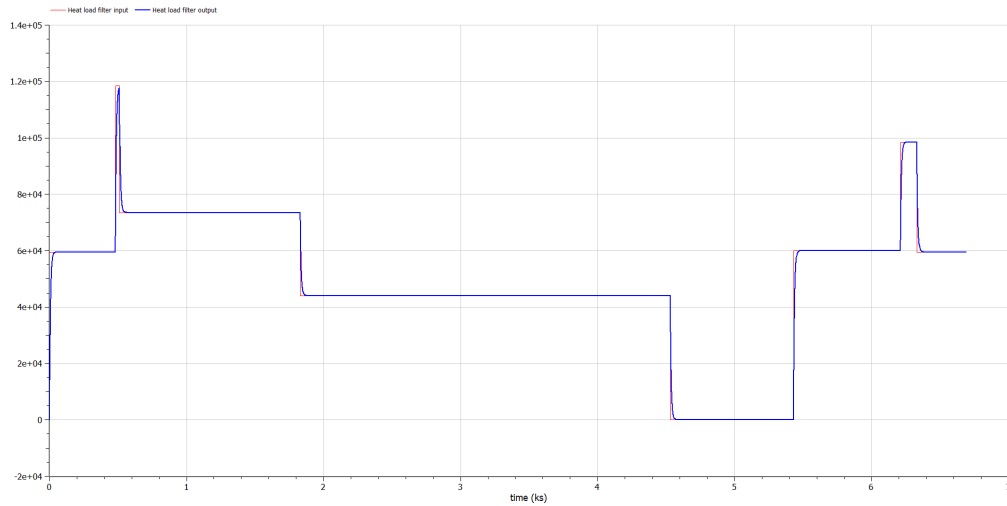


Figure 3.13: Non-filtered (red line) and filtered (blue line) heat load plotted in OpenModelica plotting environment

EGW Loop

The second main area of all architectures is the EGW loop (figure 3.14). Its goal is to extract heat from the heat load branch and reject it in different ways depending on the architecture. It is the main cooling loop of all five possible configurations, being the one in direct contact with both the heat load source and the external heat sinks.

The loop name comes from the aforementioned Ethylene-Glycol/Water 60%-40% mixture, the coolant fluid for this loop. It didn't exist in `ThermoFluidStream` library: it has been necessary to define it from scratch using datasheets from Mokon [55]. Following technical indications, the target temperature for EGW has been set at 45°C .

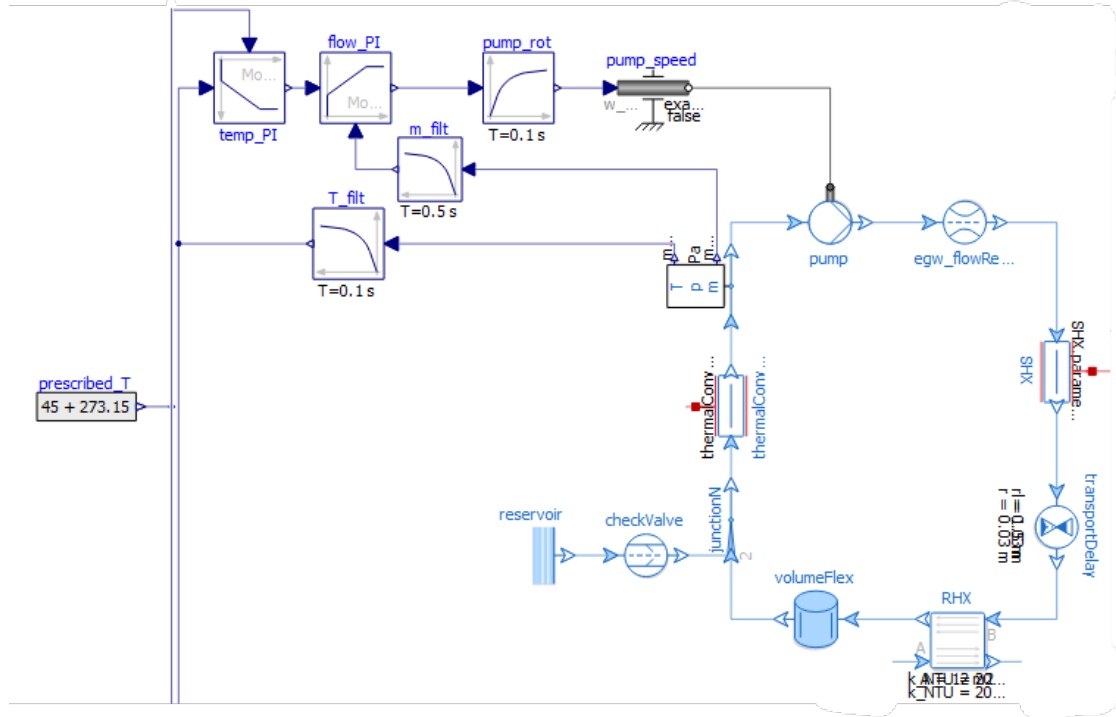


Figure 3.14: EGW loop model in Modelica graphical notation

The EGW loop is composed by the following components:

- **ThermalConvectionPipe:** the thermal-fluid interface of the loop. It presents a heat port which can be connected to heat loads or sinks and a fluid input-output setup. It's been used for both transmitting heat from the heat load and the external environment heat sink (SHX);
- **Pump:** models a simple pump behaviour. The specific behaviour is defined by the `dp_tau_pump` function. In this case, the function is `dp_tau_centrifugal`, representing a centrifugal pump. The pump shaft is driven by a control circuit, lately deepened;
- **FlowResistance:** simulates flow resistance generated in piping. The resistance function is defined by the function `pLoss`. In this case, the function is the linear quadratic pressure loss function;

- **TransportDelay**: simulates real-world delays on fluid velocity. It has been added to give more reality to the model;
- **CounterFlowNTU**: counter flow heat exchanger using the ε - NTU method [56]. It allows heat exchange between two fluids flowing in opposite directions (EGW-air and EGW-Fuel are the explored cases in this work);
- **VolumeFlex**: represents the fluid volume in the loop. It is a flexible volume with extremely high bulk modulus, simulating a fixed-volume component but without stiffening the system. The presence of a volume is mandatory for every loop.

An additional fluid branch has been added between the volume and the heat load exchange: it is the reservoir one. It is composed by three blocks:

- **Source**: fixed pressure source. It's meant to represent the reservoir;
- **CheckValve**: one-way valve. Only allows flows from the reservoir to the main loop;
- **JunctionN**: junction that allows the reservoir branch and the main loop to be connected.

This branch is essential to avoid pressure and mass flow fluctuations: due to high changes in other variables, sometimes the modeling system can produce negative pressure or mass flows values, leading to a crash. In real world, this can happen in similar shapes: high variations can lead to instantaneous unexpected fluctuations. For this reason, in real world as in simulations, installing a reservoir with a fixed pressure and a check valve is always a good practice.

As mentioned before, the pump is driven by a control system. It is composed by the following blocks:

- **MultiSensor_Tpm**: sensor for temperature, pressure and mass flow. All these outputs can be filtered by the sensor itself;
- **RealExpression**: block used to provide a constant input;
- **FirstOrder**: block that implements a first order transfer function. It can be used as a filter for signals or, if on the PID output, to simulate shaft dynamics;
- **LimPID**: PID controller: gives a measured output depending on the two inputs (setpoint and measured value) and its internal parameters (gain, characteristic times);

- **Speed:** models the forced movement of a flange according to a reference angular velocity signal (in input).

Ram Air cooling

One of the biggest resources in terms of heat sinks is ram air. Ram air is external air which flows from outside the aircraft into a duct. It is used, thanks to its speed given by the aircraft movement, to efficiently cool down components. The ram-air circuit (figure 3.15) is very simple. It has been built with five components, listed in order:

1. **Source:** simulates the external environment. It has controlled temperature and pressure. Unlike the reservoir, they are not fixed: they are variable depending on the altitude. The values for temperature and pressure are provided by two blocks that will be explained in section 3.2;
2. **Fan:** powered by a control circuit similar to the pump one, provides a pressure differential to the air flow. Its `dp_tau_fan` function had to be re-written in collaboration with DLR Robotics and Mechatronics Center to avoid negative enthalpy values. This issue was caused by an error in the original `ThermofluidStream` fan `dp` function;
3. **FlowResistance:** like the EGW loop, simulates pressure losses in piping;
4. **CounterFlowNTU/DiscretizedCounterFlowHEX:** heat exchangers between a cooling fluid (EGW or R134a from the VCS loop) and ram air.
5. **Sink:** atmosphere in which ram air is expelled after absorbing waste heat. It has controlled pressure but not temperature.

Depending on the architecture, Ram Air will have an interface with the main EGW loop or with the VCS cycle. The main difference in terms of components is the heat exchanger: for EGW loops, a continuous exchanger has been used, while for VCS loops, numerically more delicate because of the two possible phases of R134a, a discretized one has been preferred.

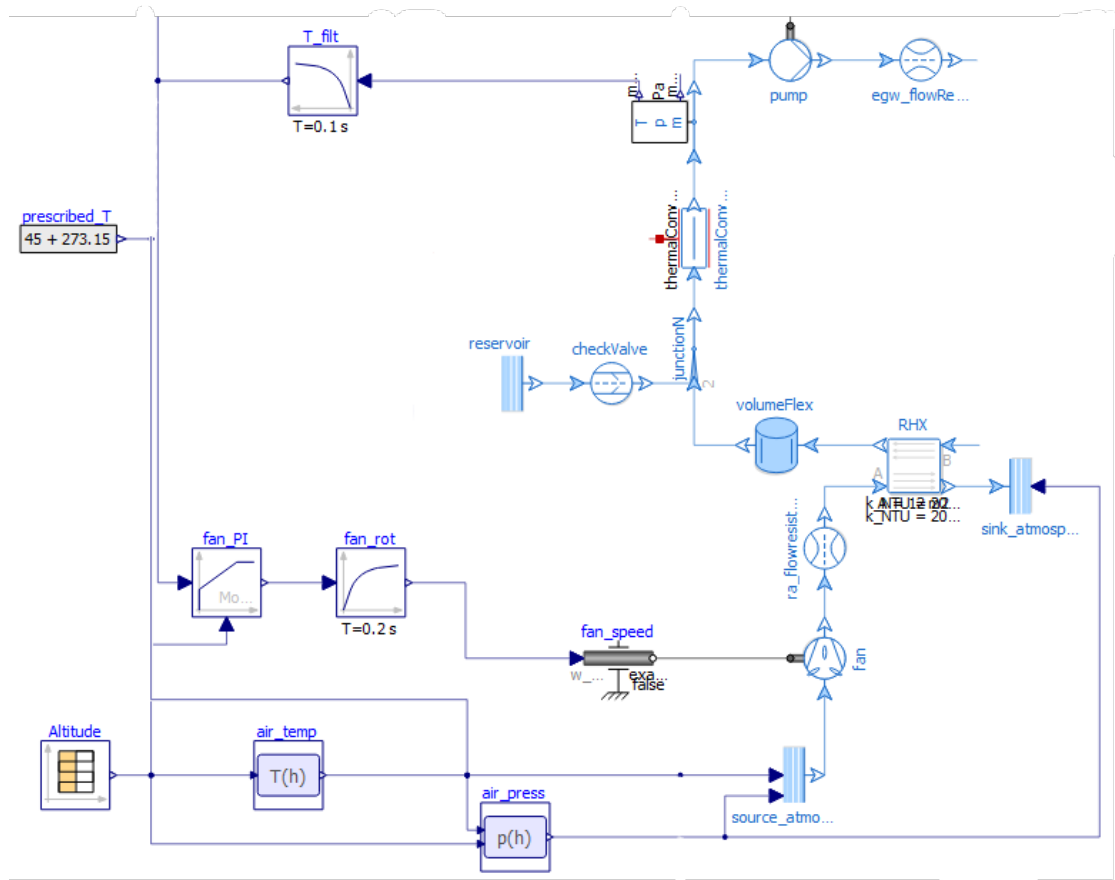


Figure 3.15: Ram Air circuit in Modelica graphical notation

Skin Heat Exchanger

The skin heat exchanger allows to directly reject heat to external atmosphere by putting into direct contact (only separated by a DURAL layer of 1.5 mm) hot fluid (EGW or Fuel) with external air. It significantly helps in heat dissipation, but requires a specific design for the component itself, needing more piping and specific plates for heat exchange. It is usually placed in the airframe skin and it can be in the fuselage or in the wing. It is modeled in a very similar shape to the Heat Load source, with the only addition of convection on the air side (figure 3.16).

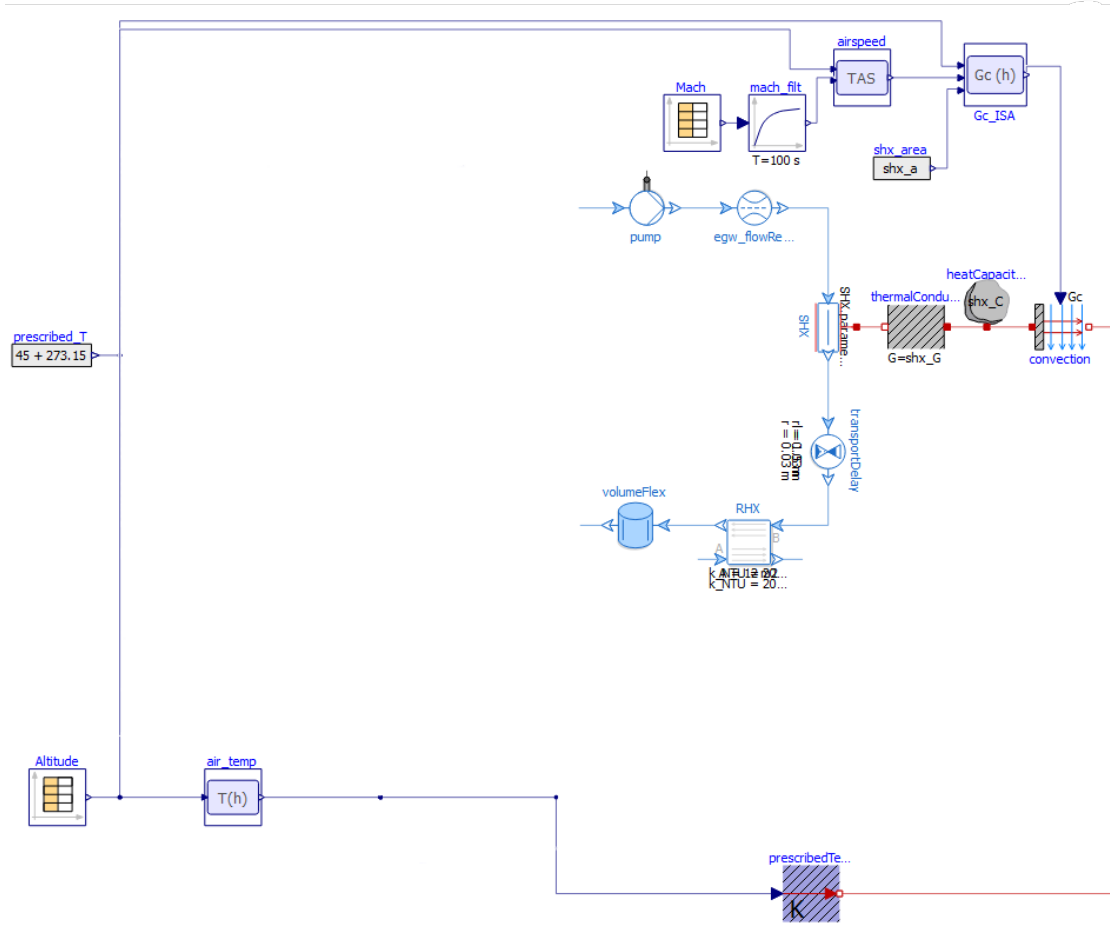


Figure 3.16: Skin Heat Exchanger model in Modelica graphical notation

The components of the SHX model are:

- **ThermalConvectionPipe**: models thermal interaction between fluid and thermal load;
- **ThermalConductor**: simulates thermal conduction;
- **HeatCapacitor**: models a physical component with its thermal properties;
- **Convection**: simulates the convection side of thermal transfer for the plate. It has the thermal output like all other components, but also an input for equivalent global conductance G_c , clarified in section 3.2.

Is important to underline that values for **ThermalConductor** and **HeatCapacitor** are dependant on SHX plate area.

As mentioned in the literature review section, SHX can be placed in both fuselage and wings. For larger SHXs, the fuselage location must be preferred due to lack of wing surface. Fuselage SHXs are innovative technologies, but they have not been applied to flying demonstrators yet.

Vapour Compression System

The Vapour Compression System is an additional loop that can be added between EGW cycle and Ram Air Circuit with the aim of extracting heat (from EGW) and rejecting it to the Ram Air stream. The fluid for VCS cycle is R134a, a refrigerant gas belonging to the hydrofluorocarbon (HFC) family. It has $C_2H_2F_4$ chemical formula, it is non-flammable and non-toxic. It is widely used for refrigeration cycles, presenting a boiling point of $-26^\circ C$, a critical temperature of $101^\circ C$ and a critical pressure of 40.6 bar [57].

R134a has a high Global Warming Potential (around 1430 [58]), which is prompting the European Union to remove it from installations [59]. Nevertheless, R134a has been chosen for this study for two reasons: to stick with Coutinho work [5] and because refrigerant gases are hard to define from scratch in Modelica language. They have two distinct phases (liquid-vapour) and go through multiple transformations. R134a is one of the only available gases in **ThermofluidStream** library. Further studies may implement new gas definitions in Modelica language.

To build the VCS cycle in Modelica language (figure 3.17), several components have been used:

- **FlowResistance**: simulates pressure losses in piping due to flow resistance;
- **DiscretizedCounterFlowHEX**: heat exchanger. It is used both as an evaporator (heat exchange EGW-R134a) and as a condenser (heat exchange R134a-Ram Air). It differs from previous heat exchangers because it is discretized in cells;
- **Compressor**: working with ideal gas assumptions, it provides a pressure differential to the fluid;
- **Receiver**: similar to **VolumeFlex**, but suited for double-phased fluids;
- **BasicControlValve**: simulates a basic valve driven by an input signal.

Similarly to other architectures, mechanical components are driven by control circuits made by sensors and PIDs. In this cycle, there are two control circuits: one to drive the valve opening and one for the compressor.

The valve control circuit has a specific goal: maintain an appropriate gas overheating value at evaporator output. A sensor, the **TwoPhaseSensorSelect** block, measures the *overheat temperature*, defined as:

$$T_{oversat} = T_{out_{evap}} - T_{sat}(p_{evap}) \quad (3.4)$$

in which $T_{out_{evap}}$ is the output temperature from the evaporator and $T_{sat}(p_{evap})$ is the saturation temperature, function of evaporator pressure. $T_{oversat}$ describes how much R134a is in vapour phase after passing through the evaporator [60]. The PID block has two inputs: measured $T_{oversat}$ and a setpoint $T_{oversat}$ temperature, 5 K [60].

One aspect that has to be highlighted regarding the VCS cycle is that it results very stiff to simulate. This problem was already known by the **ThermoFluidStream** developers at DLR. Having a two-phase cycle with two heat exchanges results in long runtimes, passing from a minutes order of magnitude to an hours one. Runtime has been brought below two hours by increasing receiver volumes (providing more thermal inertia), modifying nominal flow parameters and reducing the number of cells in heat exchangers.

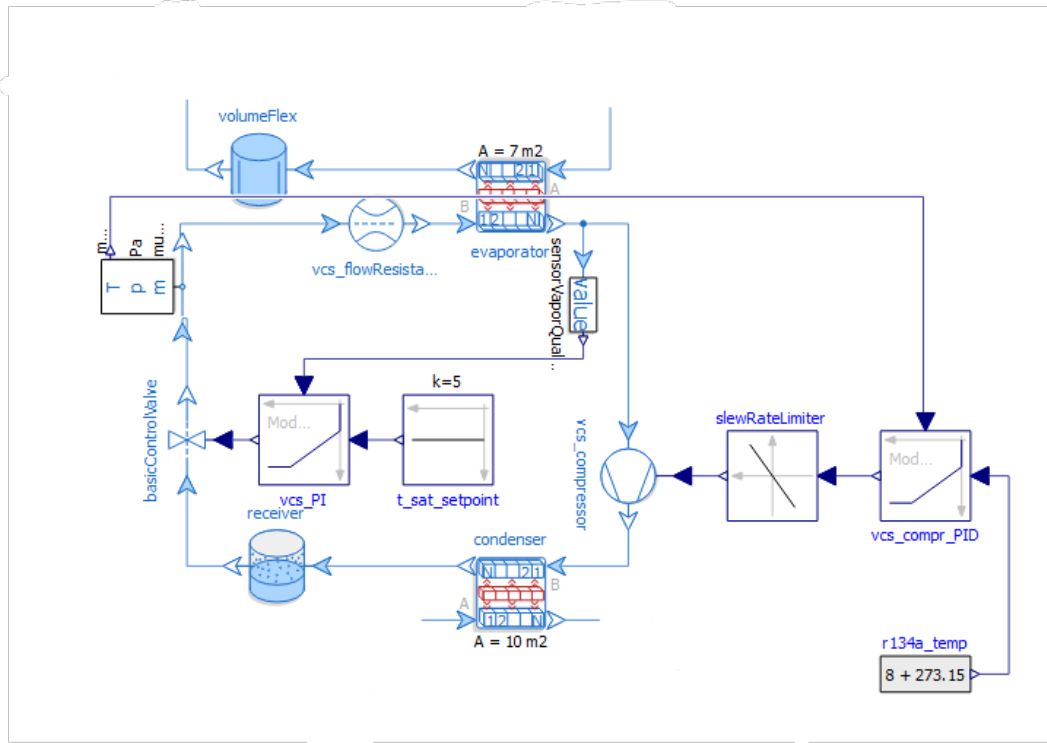


Figure 3.17: Vapour Compression System model in Modelica graphical notation

One difference with the EGW loop is that the fuel loop presents an additional branch directed to the engine. It is composed, in order of appearance, by:

1. **JuncionNM**: junction with N inputs and M outputs;
2. **BasicControlValve**: valve that controls (via control circuit) the fuel flow to the engine;
3. **Sink**: models the engine as a sink with no mass return.

Fuel flow had to be defined as a **CombiTimeTable** block (figure 3.19) which doesn't have a **.txt** file as input but a vector directly written in OpenModelica environment. Fuel flow data have been extracted from an ATR-72 datasheet [61].

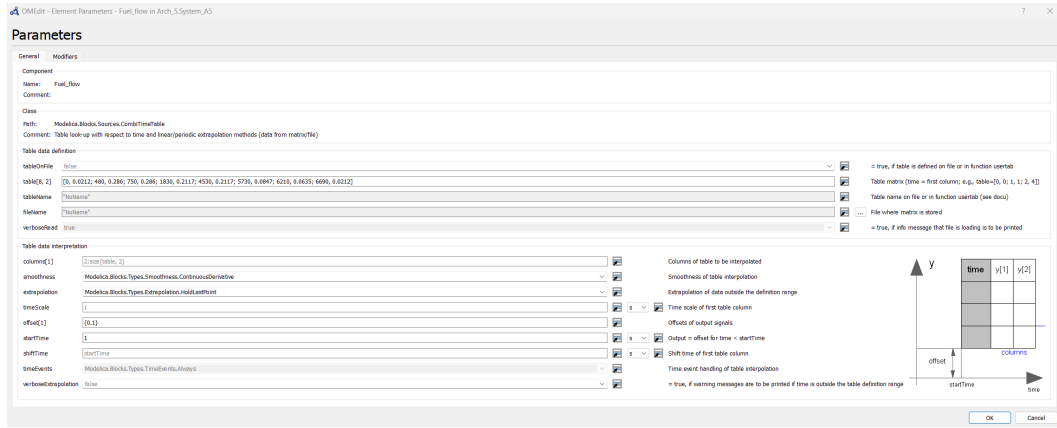


Figure 3.19: Fuel flow **CombiTimeTable** block definition in OpenModelica Editor environment

The ATR-72 reference has been chosen thanks to its similarity with the expected final design of the airplane under analysis. The output of the table is then received by a PID and elaborated with the measured fuel flow in the engine branch. The PID output is then used to control the valve.

The selected fuel for this study belongs to the SAFs family. It is an Alcohol-To-Jet blend, and it had to be defined from scratch in Modelica language in the same way of EGW, starting from datasheets [62] and implementing it as a table-defined fluid, asking Modelica to interpolate between temperatures to find the corresponding value of all thermo-physical variables.

Architectures assembly

Following the indications on table 3.1, the five architectures can be assembled.

Modularity concept from earlier pages is now applied. Loops are built once and then re-used. The combination of different loops, following table 3.1 indications, enabled the definition of all five architectures. This approach led to a faster model definition. In addition, having the same components repeated in different models allowed a faster troubleshooting when some problems were noticed.

All five architectures from Coutinho work [11] have been built using Modelica language (figures 3.20, 3.21, 3.22, 3.23 and 3.24). Models are shown in the following pages.

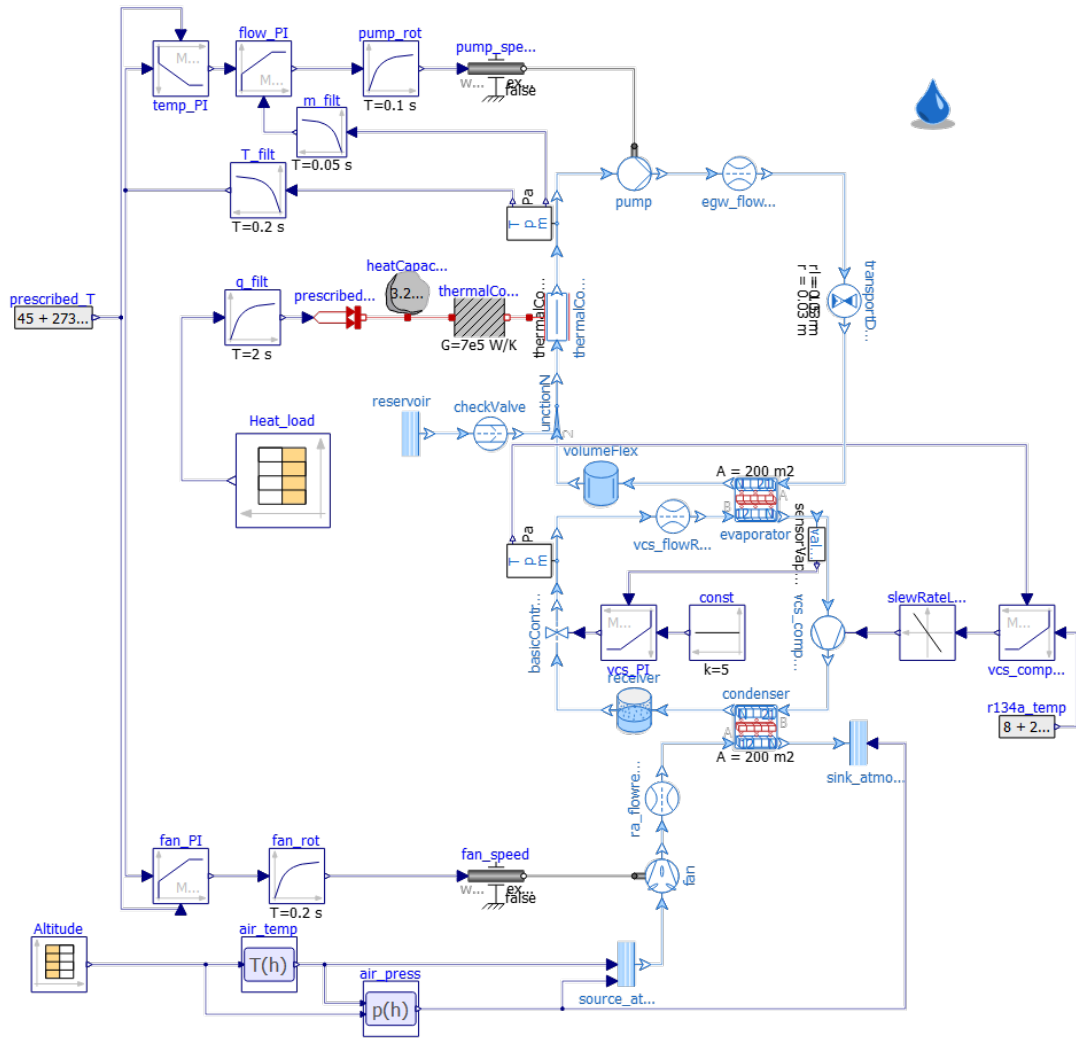


Figure 3.20: Architecture 1 in Modelica graphical notation

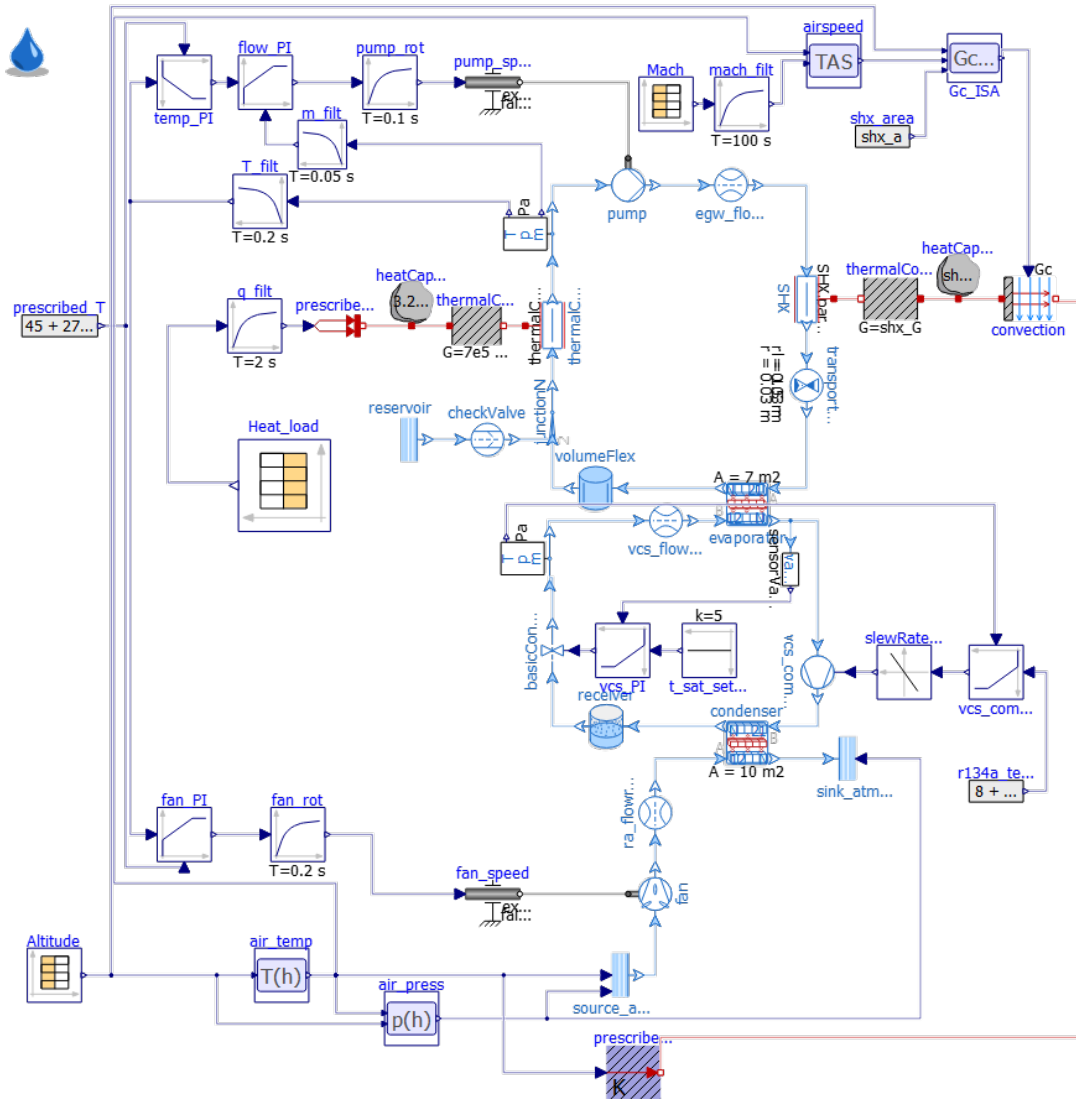


Figure 3.21: Architecture 2 in Modelica graphical notation

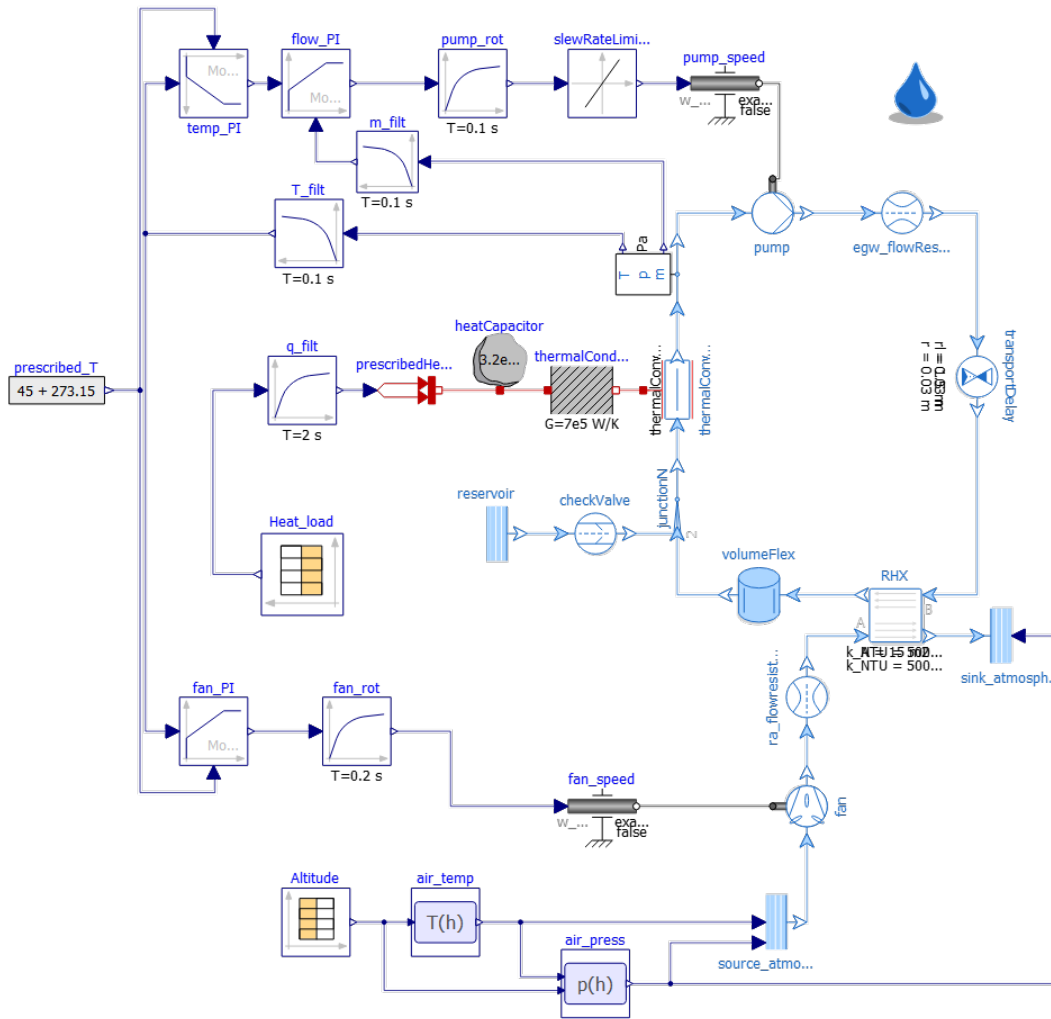


Figure 3.22: Architecture 3 in Modelica graphical notation

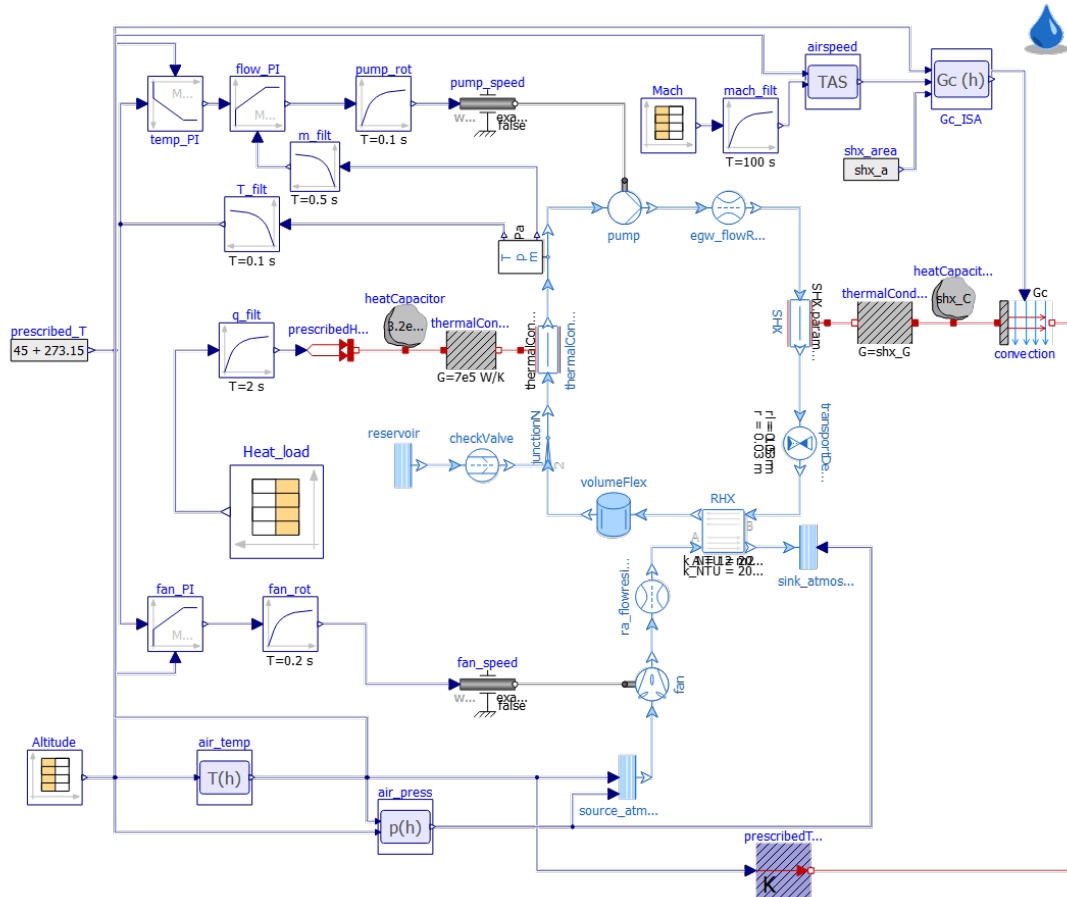


Figure 3.23: Architecture 4 in Modelica graphical notation

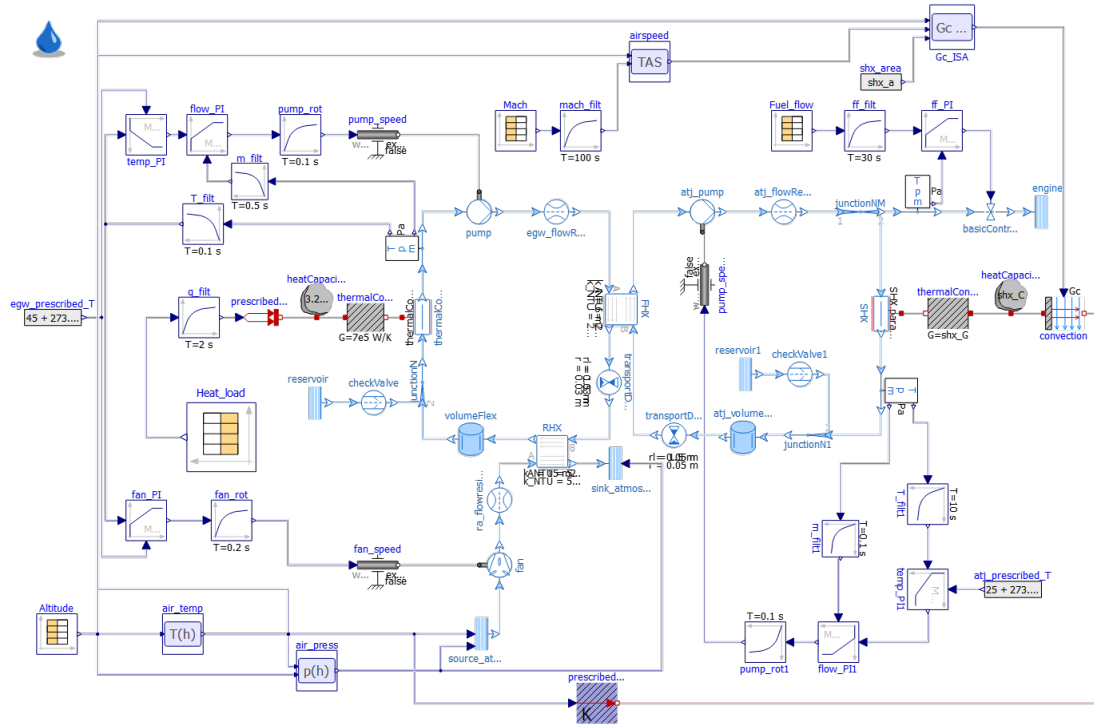


Figure 3.24: Architecture 5 in Modelica graphical notation

3.2 Mission Profile and Boundary Conditions

Performing dynamic simulations means a time dependency of external variables and, as a consequence, of internal model variables. The external context for the simulation of flight components is usually defined as a mission profile. It is usually defined by flight altitude and speed.

For a civil aircraft, the most common mission goal is to transport people or goods from one place to another. This mission can be usually divided in many different phases. Being this thesis based on an hypothetical airplane design, the mission profile has been kept as simple as possible. It is composed by 10 phases:

1. taxi-out: the airplane performs taxi, from apron to the runway;
2. take-off roll: while on the runway, the airplane accelerates until rotation. It then goes airborne to fly over a 30 *m* high obstacle;
3. climb: the airplane keeps gaining altitude to reach the cruise flight level;
4. cruise: the airplane has reached cruise altitude (27'000 *ft*);
5. descent: the airplane starts decreasing its altitude;
6. approach: the airplane performs its approach to destination airport, reducing speed and altitude until touchdown on runway;
7. landing: once landed, the airplane slows down on the runway, reaching a safe speed to exit it;
8. taxi-in: the airplane performs taxi, from runway exit to apron.

Each mission phase is defined by its starting and ending time, altitude (figure 3.25) and speed (Mach in figure 3.26) as shown in table 3.2.

The total flight time is 6690 seconds, which corresponds to 115 *min* and 30 *s* or 1 *h*, 55 *min* and 30 *s*.

Table 3.2: Mission profile data

Phase	Time [s]		Altitude [m]		Mach [–]	
	Start	End	Start	End	Start	End
Taxi-out	0	480	0	0	0.015	0.015
Take-off roll	480	510	0	15.24	0.020	0.200
Climb	510	1830	15.24	8229.6	0.300	0.300
Cruise	1830	4530	8229.6	8229.6	0.450	0.450
Descent	4530	5430	8229.6	3048	0.300	0.300
Approach	5430	6210	3048	457.2	0.200	0.200
Landing	6210	6330	457.2	0	0.050	0.050
Taxi-in	6330	6690	0	0	0.020	0.020

**Figure 3.25:** Altitude curve for the mission profile in table 3.2

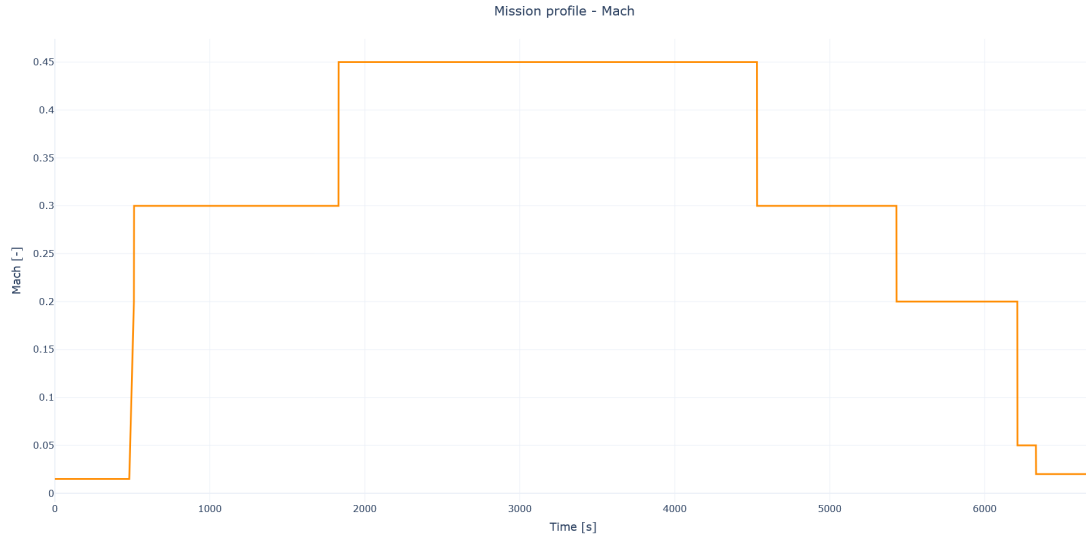


Figure 3.26: Mach curve for the mission profile in table 3.2

Each mission phase is characterised by different actions. In a propulsion-wise sight, this means different thermal loads from the electric powertrain. The waste heat for each phase has been defined from Coutinho [11] work. It is shown in table 3.3.

Table 3.3: System heat load per mission segment [11]

Mission phase	Heat load [kW]
Taxi-out	119
Take-off	237
Start/End Climb	147
Cruise	88
Descendent	0
Landing	197
Taxi-out	119

Considering that the heat load shown in table 3.3 refers to the entire airplane, it has been divided by two to account for a single wing and its related subsystems (figure 3.27).

Heat load presents two major peaks: take-off and approach/landing. In this phases, the maximum amount of power is required to the propellers. For this reason, these two phases will probably be the most-demanding ones in terms of

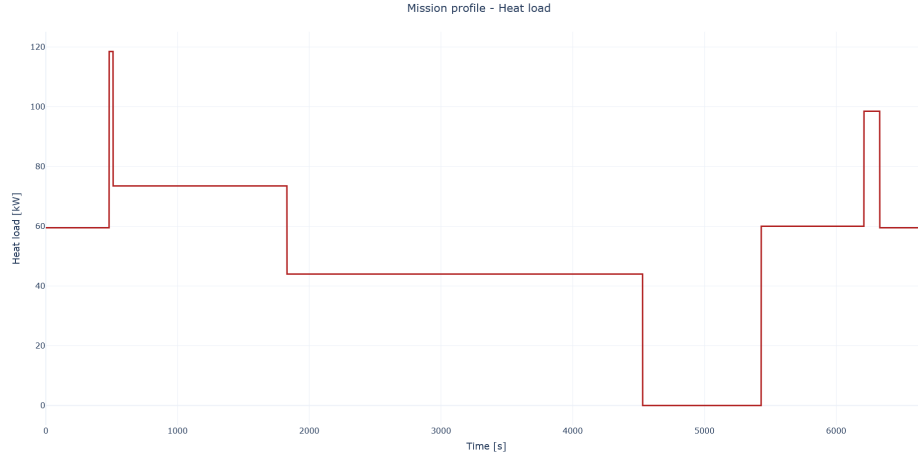


Figure 3.27: Mission profile: Heat Load from table 3.3

heat rejection, becoming the most likely sizing points for the design.

The mission profile curves have been produced via Python code. The 8 phases have been defined using the `linspace` function, creating 8 vectors of equally-spaced values, one vector for each phase. To create each vector the starting and final value were required. The function then created all intermediate values, equally spaced depending on the length (time duration of the phase) of the vector. Finally, the vectors have been combined to create complete altitude and speed vectors. To create the heat load vector a similar procedure has been followed, but the values have been set as constant for each phase, starting from Coutinho [11] data. The pseudo-code for the mission profile creation is shown in the following lines.

```

1  time = np.linspace(0, 6690, 6690 + 1)
2
3  PHASE_1 = np.linspace(start_ph1, end_ph1, length_ph1)
4  PHASE_2 = np.linspace(start_ph2, end_ph2, length_ph2)
5  PHASE_3 = np.linspace(start_ph3, end_ph3, length_ph3)
6  PHASE_4 = np.linspace(start_ph4, end_ph4, length_ph4)
7  PHASE_5 = np.linspace(start_ph5, end_ph5, length_ph5)
8  PHASE_6 = np.linspace(start_ph6, end_ph6, length_ph6)
9  PHASE_7 = np.linspace(start_ph7, end_ph7, length_ph7)
10 PHASE_8 = np.linspace(start_ph8, end_ph8, length_ph8)
11
12 tot = np.concatenate([PHASE_1, PHASE_2, PHASE_3, PHASE_4,
13 PHASE_5, PHASE_6, PHASE_7, PHASE_8])
14 tot_tab = np.column_stack((time, alt))

```

The code then creates two independent `.txt` files for altitude and speed and their corresponding plots, both in `.png` and `.html` format. The `.html` format has been widely used (thanks to `plotly` library [63]) to have more user-friendly and interactive plots, allowing in some cases the user to choose which variable to plot.

The two `.txt` files (`alt_table.txt` and `mach_table.txt`) are then read by OpenModelica as input tables. They are two-columns tables, reporting time in the first one and altitude/Mach values in the second one.

Boundary Conditions - External Atmosphere

Another aspect to be considered is external environment..

The most common way to model the atmosphere parameters is the ISA (International Standard Atmosphere) model, adopted by ICAO (International Civil Aviation Organization) from the ISO (International Organization for Standardization) formulation. Following the ISA formulation from the ICAO manuals [12], the handbook defines some important constants, shown in table 3.4.

Table 3.4: Primary constants for ISA model [12]

Symbol	Value	Unit	Description
T_0	288.15	K	Sea level air temperature
p_0	101325	Pa	Sea level air pressure
ρ_0	1.225	kg/m^3	Sea level air density
g_0	9.80665	m/s^2	Gravitational acceleration
R	287.05287	$J/(kg \cdot K)$	Specific constant for dry air
L	0.0065	K/m	Thermal gradient (Troposphere)
γ	1.4	—	Specific heat coefficient

The main starting hypotheses [12] are:

- air as dry gas, without particles and humidity;
- hydrostatic equilibrium and constant gravitational field;
- reference values from table 3.4.

The assumptions led to the following formulations for temperature T , pressure p and density ρ as function of the altitude h [12]:

$$T = T_0 - Lh \quad (3.5)$$

$$p = p_0 \left(\frac{T}{T_0} \right)^{\frac{g_0}{RL}} \quad (3.6)$$

$$\rho = \rho_0 \left(\frac{T}{T_0} \right)^{\frac{g_0}{RL} - 1} \quad (3.7)$$

The equations shown are only valid for the Troposphere, the first layer of the atmosphere. It extends from sea level to 11 *km*.

It is then possible to derive [12]:

$$a = \sqrt{\gamma RT} \quad (3.8)$$

$$V = M \cdot a \quad (3.9)$$

$$q = \frac{1}{2} \rho V^2 \quad (3.10)$$

$$p(h) = p_0 \left(1 + \frac{\gamma - 1}{2} M^2 \right)^{\frac{\gamma}{\gamma - 1}} \quad (3.11)$$

$$T(h) = T_0 \left(1 + \frac{\gamma - 1}{2} M^2 \right) \quad (3.12)$$

in which M is the Mach number and V is the airspeed (TAS - True AirSpeed). The equations describe a direct dependency of speed of sound a , dynamic pressure q , pressure $p(h)$ and temperature $T(h)$ from altitude and speed.

Atmospheric data haven't been given to the simulation as input. Instead, its values have been produced inside the simulation, starting from the altitude and speed values. As mentioned in earlier sections, Modelica language allows to create blocks that represent code lines. To give more flexibility to the models, pre-made environmental databases have not been used. Instead, in-simulation calculations have been used, only relying on speed and altitude databases. Using this approach, if the atmosphere changes the only necessary modification would be changing altitude and Mach tables.

The Modelica model needs some blocks to compute h and M values for each time instant. For this reason, a total of four blocks have been created:

- T_h:
 - input: altitude h [m]
 - output: temperature at flight level $T(h)$ [K]
 - description: implements equation 3.5 to compute temperature at given flight level;

- **p_h:**
 - input: altitude h [m], temperature $T(h)$ [K]
 - output: pressure at flight level $p(h)$ [Pa]
 - description: implements equation 3.6 to compute pressure at given flight level;
- **TAS:**
 - input: temperature $T(h)$ [K], Mach number M [–]
 - output: airspeed V [m/s]
 - description: implements equation 3.8 and 3.9 to compute airspeed for a defined Mach and given external environment and altitude. Only used for architectures that require an SHX;
- **FlatPlateGc_ISA:**
 - input: altitude h [m], airspeed V [m/s], SHX area A_{SHX} [m²]
 - output: equivalent global conductance G_c [W/K]
 - description: implements equation 3.24 to calculate equivalent global conductance. Only used for architectures that require an SHX.

Global Conductance

FlatPlateGc_ISA (textual notation in figure 3.28) is the most complex block. It is essential for a correct SHX simulation because it provides the equivalent global conductance G_c of the flat plate. This parameter indicates how many Watts of waste heat are rejected for a temperature difference of a Kelvin between the plate and external air. It takes into account three factors, the inputs of the block: altitude, from which is possible to have external temperature and pressure, airspeed, which influences the heat exchange capability, and SHX area, which defines in how many square meters of plate the heat exchange takes place. All the workflow and equations come from Incropera book *Fundamentals of Heat and Mass Transfer* [56]. The code contained in **FlatPlateGc_ISA** block derives from it and can be divided in six sections:

1. ISA atmosphere: calculates ISA atmosphere values. The decision to calculate once again these values inside the blocks comes from the goal to simplify visually and conceptually the connection lines between blocks, avoiding a web of inputs for the **FlatPlateGc_ISA** block;

2. Mach number and recovery temperature: evaluate Mach speed with earlier equations and take into account viscous-adiabatic compression of fluid, which causes a temperature increase:

$$T_r = T_\infty \left[1 + \frac{\gamma - 1}{2} r_r M^2 \right] \quad (3.13)$$

in which T_r is the recovery temperature, T_∞ is the static free-stream temperature and $r_r = Pr^{\frac{1}{3}}$ is the *recovery factor* for a laminar stream on a flat plate;

3. thermo-physical properties: evaluates fluid properties depending on the film temperature $T_f = \frac{1}{2}(T_s + T_r)$, in which T_s is the flat plate surface temperature. It calculates dynamic viscosity with Sutherland's Law:

$$\mu = \mu_0 \left(\frac{T_f}{T_0} \right)^{\frac{3}{2}} \cdot \frac{T_0 + S}{T_f + S} \quad (3.14)$$

in which $\mu_0 = 1.716 \times 10^{-5}$ and $S = 110.4 \text{ K}$ and then the thermal conductivity with an empirical fit:

$$k \approx 0.0241 + 7.72 \times 10^{-5} \cdot (T_f - 273.15) \quad (3.15)$$

and finally density and cinematic viscosity:

$$\rho = \frac{p_\infty}{RT_\infty} \quad (3.16)$$

$$\nu = \frac{\mu}{\rho} \quad (3.17)$$

4. Nusselt number and corrective coefficient: evaluates local and average convection coefficient. The first step is to obtain the Nusselt number:

$$Nu = \begin{cases} 0.664 Re^{\frac{1}{2}} Pr^{\frac{1}{3}}, & \text{for } Re < Re_{cr} \\ 0.664 Re_{cr}^{\frac{1}{2}} Pr^{\frac{1}{3}} + 0.037 (Re^{0.8} - Re_{cr}^{0.8}) Pr^{\frac{1}{3}}, & \text{for } Re \geq Re_{cr} \end{cases} \quad (3.18)$$

in which $Re_{cr} = 3.7 \times 10^5$. This is the value at which the fluid changes its flowing regime, passing from laminar to turbulent.

It is now possible to obtain the convection coefficient:

$$h_{conv} = \frac{Nu \cdot k}{L} \quad (3.19)$$

5. linear equivalent radiative exchange: evaluate the radiative component linearising Stefan-Boltzmann Law ($q = \varepsilon\sigma(T_s^4 - T_\infty^4)$):

$$h_{rad} = 4\varepsilon\sigma T_m^3 \quad (3.20)$$

$$T_m = \frac{T_s + T_\infty}{2} \quad (3.21)$$

$$h_{eq} = h_{conv} + h_{rad} \quad (3.22)$$

in which T_m is the mean film temperature, ε is the emissivity and $\sigma = 5.67 \times 10^{-8} \frac{W}{m^2 \cdot K^4}$ is the Stefan-Boltzmann constant;

6. equivalent global conductance: evaluates effective area:

$$A_{eff} = \eta_f N_{sides} A \quad (3.23)$$

in which η_f is a parameter called *fin effectiveness* of the plate, N_{sides} is the number of wet sides of the plate and A is the area of the plate. Finally, the conductance can be calculated:

$$G_c = h_{eq} A_{eff} \quad (3.24)$$

```

65 Area Aeff;
66 Temperature Tm_rad;
67 Real r_rec;
68 equation
69 // ISA T(h), p(h)
70 Tinf = if h <= 11000 then T0 - Lb*max(0,h) else T11;
71 pinf = if h <= 11000 then
72   p0*(Tinf/T0)^(g0/(R*Lb))
73   else
74     p0*((T11/T0)^(g0/(R*Lb)))*exp(-g0*(h-11000)/(R*T11));
75
76 // Mach and recovery temperature
77 a = sqrt(gamma*R*Tinf);
78 M = U / max(1e-9, a);
79 r_rec = Pr^(1/3);
80 Tr = if useRecovery then Tinf*(1 + 0.5*(gamma-1)*r_rec*M^2) else Tinf;
81
82 // Surface and film temperatures
83 Ts = Tinf + dT0 + kU*U;
84 Tf = 0.5*(Ts + Tr);
85
86 // Air properties at Tf (rho at static Tinf)
87 mu = 1.716e-5*(Tf/273.15)^(1.5)*(273.15+110.4)/(Tf+110.4);
88 rho = pinf/(R*max(200.0,Tinf));
89 nu = mu/rho;
90 k = 0.0241 + 7.72e-5*(Tf - 273.15);
91
92 // Re, Nu
93 Re = max(1e-9, U*L/nu);
94 Pr13 = Pr^(1/3);
95 Re05 = sqrt(Re);
96 Re08 = Re^0.8;
97 Rec05 = sqrt(Rec);
98 Rec08 = Rec^0.8;
99 Nu_lam = 0.664*Re05*Pr13;
100 Nu_mix = 0.664*Re05*Pr13 + 0.037*(Re08 - Rec08)*Pr13;
101 Nu = if noEvent(Re < Rec) then Nu_lam else max(Nu_lam, Nu_mix);
102
103 // h and Gc
104 hconv = Nu*k/L;
105 Tm_rad = 0.5*(Ts + Tinf);
106 hrad = if useRadiation then 4*eps*sigma*Tm_rad^3 else 0;
107 heq = hconv + hrad;
108 Aeff = eta_f*sides*A;
109 Gc = heq*Aeff;

```

Figure 3.28: Partial screenshot of FlatPlateGc_ISA block in Modelica textual notation

Mach input has been filtered, as shown in figure 3.29. The profile shown in figure 3.26 is not realistic, but just a theoretic division dependant from the flight phases definition. In the Modelica models, Mach input has been filtered with a `FirstOrder` block, representative of a filter with time constant of $T = 100$ s.

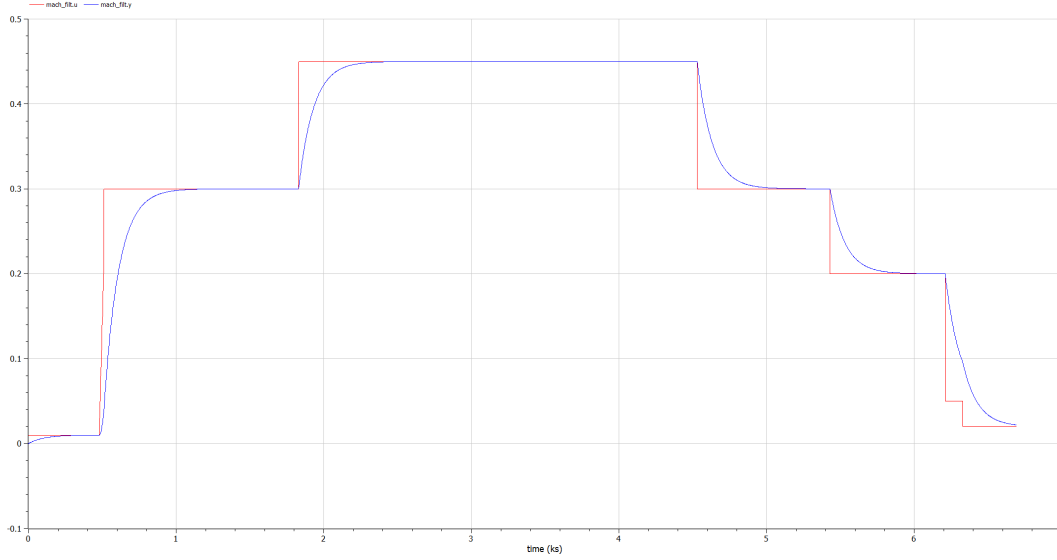


Figure 3.29: Non-filtered (red line) and filtered (blue line) Mach in OpenModelica plotting window

With the filter, the Mach curve is more realistic and less critical in terms of sudden input values changes. Sometimes, strong discontinuities may lead the solver to stiffer systems, that in some cases can result in longer runtimes while in other cases they can lead to crashes.

3.3 Components Empirical Sizing

The inputs for the sizing process are reported in table 3.5.

Table 3.5: Sizing input variables for each component

COMPONENT	SIZING INPUTS
EGW Pump	$\dot{m}, \Delta p$
ATJ Pump	$\dot{m}, \Delta p$
Ram-Air Fan	$\dot{m}, \Delta p$
VCS Compressor	$\dot{m}, \Delta p, p_{asp}, T_{asp}$
SHX	A_{SHX}

in which:

- $\dot{m}$ is the mass flow through the component;
- Δp is the pressure differential between entrance and exit of the component;
- p_{asp} is the compressor aspiration pressure;
- T_{asp} is the compressor aspiration temperature;
- A_{SHX} is the Skin Heat Exchanger area.

The sizing process for turbomachinery will be divided for each component, in order to give a clear vision of the steps undergone. Once electrical powers are given for each component, the final sizing phase will be explained. Both phases come from two textbooks [36] [37].

3.3.1 Turbomachinery - Powers estimation

Pumps (EGW & ATJ)

Pumps sizing relies on mass flow and pressure differential. Two assumptions were made in this process:

1. fluid keeps its density constant during the whole simulation: $\rho_{EGW} = 1029 \text{ kg/m}^3$ for EGW [55] and $\rho_{ATJ} = 776 \text{ kg/m}^3$ for ATJ [62];
2. mechanical efficiency of the component is constant during the whole simulation, with a value of $\eta_{pump} = 0.5$.

The first step is to evaluate volumetric flow:

$$\dot{V} = \frac{\dot{m}}{\rho} \quad (3.25)$$

Then, it is possible to find hydraulic power with:

$$P_{hyd_{pump}} = \dot{V} \cdot \Delta p_{pump} \quad (3.26)$$

and finally the electrical power:

$$P_{el_{pump}} = \frac{P_{hyd_{pump}}}{\eta_{pump}} \quad (3.27)$$

Ram-Air Fan

Ram-Air fan follows a similar procedure to pumps. Similar assumptions are made:

1. air density is constant during the whole simulation: $\rho_{air} = 1.010 \text{ kg/m}^3$ [64];
2. mechanical efficiency of the component is constant during the whole simulation, with a value of $\eta_{pump} = 0.4$.

First, volumetric flow is obtained:

$$\dot{V} = \frac{\dot{m}}{\rho_{air}} \quad (3.28)$$

Then, aerodynamic power:

$$P_{air} = \dot{V} \cdot \Delta p_{fan} \quad (3.29)$$

and finally electrical power:

$$P_{el_{fan}} = \frac{P_{air}}{\eta_{fan}} \quad (3.30)$$

VCS Compressor

VCS compressor sizing follows a slightly different sizing procedure. Starting assumptions are:

1. isentropic efficiency of the fan is constant during the entire simulation, with a value of $\eta_{isVCS} = 0.7$;
2. mechanical efficiency of the fan is constant during the entire simulation, with a value of $\eta_{mechVCS} = 0.8$;
3. electric motor efficiency of the fan is constant during the entire simulation, with a value of $\eta_{motVCS} = 0.8$;
4. isentropic index γ of R134a is fixed [57]:

$$\gamma_{R134a} = \frac{c_p}{c_v} = 1.12$$

5. specific heat c_p of R134a is fixed: $c_p = 880 \text{ J/KgK}$ [57].

The first parameter to evaluate is compression ratio:

$$\pi = 1 + \frac{\Delta p_{VCS}}{p_{asp}} \quad (3.31)$$

Then, isentropic work:

$$w_{is} = c_p T_{asp} \left(\pi^{\frac{\gamma-1}{\gamma}} - 1 \right) \quad (3.32)$$

Then shaft power:

$$P_{shaftVCS} = \dot{m} \cdot \frac{w_{is}}{\eta_{isVCS}} \quad (3.33)$$

and finally the electrical power:

$$P_{elVCS} = \frac{P_{shaftVCS}}{\eta_{mechVCS} \cdot \eta_{motVCS}} \quad (3.34)$$

3.3.2 Turbomachinery - Masses estimation

Two mathematical tools have been used for masses estimation: empirical laws and scaling laws. The combination of these two tools allowed to move from a canonical form of an empirical law based on power:

$$m = C \cdot \left(\frac{P}{P_{ref}} \right)^\alpha \quad (3.35)$$

in which:

- m is the component estimated mass;
- C is an empirical scaling component;
- P is the sizing power value;
- P_{ref} is the reference power from a known datasheet;
- α is a scaling exponent.

to sizing equations for the components.

Three parameters must be defined: C , P_{ref} and α . Their value will be bounded to each specific component.

Starting with P_{ref} , it can easily be defined by looking at datasheets: as shown in former paragraphs, data are already available for each component as nominal state electrical power consumption.

The following parameter to be defined is α : this exponent is usually reported in literature in three variants [37]:

- $\alpha < 1$: mass has a slower growth than power, common in more simple machines;
- $\alpha = 1$: linear relationship between mass and power;
- $\alpha > 1$: mass has a faster growth than power, common in more complex machines.

Pumps and fans can be identified with $\alpha < 1$ category. A value of $\alpha = 0.6$ has been chosen to match with literature common uses. On the other hand, compressors are more complex machines due to their multiple internal stages. An exponent of $\alpha = 1.5$ is usually preferred in literature.

Finally, coefficient C value must be identified. For this coefficient a more complex procedure is needed: it is usually performed via a linear regression with multiple component variants datasheets. It must be underlined that, for each component, a family must be chosen: using regression laws with completely different items is mathematically incorrect. For this reason, components must be precisely chosen and then variants (more or less big/powerful) are used, but always from the same model. Then, the regression law is built imposing that, when reference datasheet power is imposed ($P = P_{ref}$), estimated mass matches the declared mass on datasheet.

Due to this variability, coefficient C varies for each component of the model, as shown in table 3.6.

Table 3.6: Empirical sizing mass law coefficients

Component	C	P_{ref} [W]	α	Equation
EGW Pump	2.55	212.00	0.60	$m = 2.55 \cdot \left(\frac{P}{212.0}\right)^{0.60}$
ATJ Pump	3.90	450.00	0.60	$m = 3.90 \cdot \left(\frac{P}{450.0}\right)^{0.60}$
Ram-Air Fan	7.50	2000.00	0.60	$m = 7.50 \cdot \left(\frac{P}{2000.0}\right)^{0.60}$
VCS Compressor	143.00	28900.00	1.50	$m = 143.00 \cdot \left(\frac{P}{28900.00}\right)^{1.50}$

3.3.3 Skin Heat Exchanger

Skin Heat Exchanger sizing followed a different logic. First, two assumptions were made:

1. SHX is entirely composed by DURAL (Al-Cu alloy), with an indicative density (dependant on alloy percentages) of $\rho_{DURAL} = 2780.0 \text{ kg/m}^3$ [65];
2. SHX thickness is constant, with a value of $t_{DURAL} = 1.5 \text{ mm}$.

The only design variable for SHX is the surface area, A_{SHX} . Therefore, mass estimation is:

$$m = V_{SHX} \cdot \rho_{DURAL} = A_{SHX} \cdot t_{SHX} \cdot \rho_{DURAL} \quad (3.36)$$

in which $V_{SHX} = A_{SHX} \cdot t_{SHX}$ is SHX plate volume expressed in $[m^3]$.

3.3.4 Safety Factor

Every engineering project doesn't rely on raw estimations, it accounts instead for *safety factors* [41]. Safety factors increase the estimation to match a confidence span defined for each case. They can increase raw values by 20%, 100% or more, depending on the type of project, the design workflow stage and the type of component.

There is not a fixed safety factor for a defined case, its choice is up to the designer. For this case, a 50% safety factor has been chosen. This choice was made because Thermal Management Systems are safety-critical components: their failure may bring to engine failures due to overheating. Another reason is that, due to the approximation performed with sizing methods (empirical laws instead of full and detailed component project), a 50% safety factor ensured enough confidence in the results. Indeed, 50% is a very common value in aerospace projects [9], as shown in figure 3.30.

Equipment	Factor of Safety - FOS -
Aircraft components	1.5 - 2.5
Boilers	3.5 - 6
Bolts	8.5
Cast-iron wheels	20
Engine components	6 - 8
Heavy duty shafting	10 - 12
Lifting equipment - hooks ...	8 - 9
Pressure vessels	3.5 - 6
Turbine components - static	6 - 8
Turbine components - rotating	2 - 3
Spring, large heavy-duty	4.5
Structural steelwork in buildings	4 - 6
Structural steelwork in bridges	5 - 7
Wire ropes	8 - 9

Figure 3.30: Typical Safety Factors in engineering projects [9]. Aircraft components safety factor usually lies between 1.5 and 2.5

3.4 DOE Implementation

As described in chapter 2.5, Design Of Experiments is the starting point of several state-of-the-art MDO workflows.

Considering TMS models previously developed, their design process put the focus on the probability of failed simulations because of out-of-bounds values reached by fluids and mechanical components. Failures appeared when the model was working out of datasheet prescribed values. For this reason, a DOE resulted necessary to explore the design space.

The first step for a DOE construction is defining the design variables, the variables that will move into a range, changing the model state and giving, at the end of the MDO process, the final optimized design.

They must be variables defined before running the simulation. Variables variation must give a sensible variation in system outputs. Two groups of variables were considered: physical sizes of components and working parameters.

Physical sizes group contained all dimensions considered in a component design. For example, in a pump they would be diameter and length of the internal chamber.

Working variables group instead is composed by initial datasheet functioning values. For example, in a pump they would be pressure head, mass flow and rotational speed of the shaft.

The choice for design variables started from the two aforementioned groups. To promote an effective optimization process, the driving feature for the evaluation has been the system sensitivity to variables variation. To perform a correct optimization, the system must be sensitive to design variables change.

A simple sensitivity analysis has been performed by evaluating values variation in different runs inside OMEdit. The working variables group resulted the best one: the model responded in a more sensitive way to working variables variation. Furthermore, working variables are easier to find in datasheets because they're usually interesting data for purchasers, while internal physical sizes are often protected by industrial secret.

Once the variables type has been decided, they must now be identified for each component of interest (to optimize the system design): pumps, fans, compressors and skin heat exchanger.

Table 3.7: Design variables for DOE and MDO workflow

COMPONENT	DESIGN VARIABLES
EGW Pump	$TDH, \dot{m}_{ref}, \omega_{ref}$
ATJ Pump	$TDH, \dot{m}_{ref}, \omega_{ref}$
Ram-Air Fan	$\dot{m}_{ref}, \omega_{ref}, pr_{ref}$
VCS Compressor	$\dot{m}_{ref}, \omega_{ref}$
SHX	A_{SHX}

Design variables are reported in table 3.7. TDH is the pump pressure head, $\dot{m}_{ref}$ is the nominal (maximum working efficiency) mass flow, ω_{ref} is the nominal shaft rotational speed and pr_{ref} is the nominal pressure ratio of the compressor. A_{SHX} is the Skin Heat Exchanger area.

In order to create a DOE, each variable must vary inside a defined range. Range bounds have been arbitrarily chosen during early design phases. In order to give more re-usability to the code, ranges haven't been defined in terms of physical variable values, but normalized ones. In this configuration, the range is built around the "1" value, which represents the starting datasheet suggestion. If the value goes below "1", it means that it's decreasing, while if it goes over "1" it increases. For example, using a "1.1" value for a variable in the DOE means that the datasheet value is increased by 10%.

As mentioned, bounds have been arbitrarily defined. They have not been defined for each variable, but just for each component. This way, all variables of a component (for example mass flow and shaft rotational speed for the compressor) re-scale at the same time. This choice avoids having configurations that present heavy mismatches between variables of the same component, like for instance 90% of the mass flow but 115% of the rotational speed. This wouldn't make sense, because it is not a re-scaling anymore, but just randomly assigned values.

Ranges bounds must be decided. To give a sufficient margin to the DOE a $\pm 15\%$ variation from nominal values has been chosen. Only SHX area presents a different value: being the area not linked to datasheets but just to available surface, it is possible to allow a wider range. For this reason, a $\pm 40\%$ range has been chosen. The only constraint imposed on SHX area is the available surface. Taking into account a similar airplane to ATR-72 [61], wing surface has not been exceeded. Literature review showed some examples of in-fuselage SHX, but their TRL suggests that several years of study are left, compromising the entry into service goal of 2035.

The ranges have been defined (as shown in table 3.8), but the center ("1")

Table 3.8: Design variables ranges

Component	Range	
	Min	Max
EGW Pump	0.85	1.15
ATJ Pump	0.85	1.15
Ram-Air Fan	0.85	1.15
VCS Compressor	0.85	1.15
SHX	0.60	1.40

value still has to be set. For this reason, several datasheets have been studied. The starting point of the research has been practical experience: while building Modelica models, a trial and error process has been followed to determine the best design space. While imposing the final heat load, components values have been empirically defined: by multiple simulation runs, indicative values for each component have been defined. The goal was achieving a full run without any out-of-bounds value and no sudden variations in fluids temperatures or pressures. This empirical workflow became useful when, while building the DOE, the research for real-world components had to start. Instead of searching across all types of components (aeronautical and non-aeronautical), it was already clear that the first try should have been a component close to the values that make the models run without any problem.

Starting from that assumption and looking to Modelica results, datasheets of real turbomachinery components have been found:

- **EGW Pump:** Pierburg GmbH CWA400 Electric Water Pump (PWM Version) [13] - table 3.9

Table 3.9: EGW Pump nominal values [13]

EGW Pump Datasheet		
T_{DH} [m]	$\dot{m}_{ref}$ [m^3/s]	ω_{ref} [rad/s]
9.0	0.0025	314.2

- **ATJ Pump:** Parker Hannifin Corporation 28 VDC Brushless Fuel Pump Product Family [14] - table 3.10

Table 3.10: ATJ Pump nominal values [14]

ATJ Pump Datasheet		
TDH [m]	$\dot{m}_{ref}$ [m^3/s]	ω_{ref} [rad/s]
22.0	0.0012	314.2

- **Ram-Air Fan:** ebmpapst Mulfingen GmbH & Co. KG EC Centrifugal Module RadiCal, Backward-Curved, with Housing (Type K3G175-RC05-03) [15] - table 3.11

Table 3.11: Ram-Air Fan nominal values [15]

Ram-Air Fan Datasheet		
$\dot{m}_{ref}$ [kg/s]	ω_{ref} [rad/s]	pr_{ref} [—]
3.0	850.0	1.01

- **VCS Compressor:** BITZER Kühlmaschinenbau GmbH Hermetic Scroll Compressors ORBIT, ORBIT+ and ORBIT FIT [16] - table 3.12

Table 3.12: VCS Compressor nominal values [16]

VCS Compressor Datasheet	
$\dot{m}_{ref}$ [kg/s]	ω_{ref} [rad/s]
0.13	314.2

The last value to be decided is the DOE size. After several trials, the number of DOE samples has been set to 500 evaluations for two different reasons:

1. many DOE points bring to failed simulations (due to out-of-bounds values), this leads to many unusable points;
2. as a consequence of reason 1, surrogate models strongly improve their reliability with more than 50/60 points.

Out of 500 DOE points, a number between 80 and 90 of them is feasible, leading to a 16% – 26% success rate. This value raises a concern on further steps reliability. The design space is not defined, it presents a lot of discontinuities and "holes" that

do not allow a safe exploration. In any point of the design space, a failure can happen just because of the specific combination of design variables. For instance, exceeding a structural limit can abort a simulation that was running without any problem, just because a single component is under-dimensioned and therefore not able to withstand the working regime.

This unpredictable nature of failures led to the recognition of the "hidden constraints" mentioned in the Literature section (chapter 2.5.2). PoV functions represent the most interesting solution to address hidden constraints. PoV functions can predict whether a given design space point can bring to a successful simulation (within a previously defined threshold of confidence). They are trained on both feasible and unfeasible points: they learn in which kind of combinations feasibility probability is over the threshold.

The number of DOE feasible points must be increased in order to produce better surrogate models. PoV functions can be used to drive the DOE points choice.

The first step is the training of the PoV function. The DOE run produced a dataset which correlates design variables (inputs), normalized in terms of λ , and outputs: objective variables and feasibility mask. PoV functions need the second kind of output: feasibility mask. Feasibility mask is a binary flag that indicates whether the simulation which it refers to can be considered "feasible". Value 1 means feasible, while value 0 means unfeasible.

Criteria to assign the mask are multiple:

1. simulation has ended: checks whether the simulation has reached its final instant of time;
2. finite objectives: mass and power sizing doesn't have `inf` or `NaN` values as outputs;
3. builder bounds filter: control variables (such as pressure differential and mass flow) don't exceed operating bounds.

Almost all of the filtered points derive from the first criterion. This is due to the nature of the latter: if a simulation reaches the end without failing, it means that all the checks that OpenModelica performs for each time instant are satisfied. Checks include boundaries respect (e.g. temperature and pressure inside declared allowed limits for fluids) and numerical checks, meaning that the associated equations system leads to finite results.

Builder Bounds Filter

Builder Bounds filter is based on a simplified equation. Starting from nominal working values, the script checks whether in any instant of time the working variables exceed a defined percentage of the nominal value. This process is dynamic: it adapts to every new nominal value (e.g. the one created by the DOE).

An example of controlled variables are fluid temperature and pressure inside turbomachinery components. Despite OpenModelica already performs checks on minimum and maximum allowed values for fluids, there's the possibility that the value may be acceptable in absolute terms for the fluid but unacceptable in terms of maximum load capacity for the component. It can happen when the component has been designed for, in this example, lower temperatures and pressures than the operational ones for the fluid used.

The mathematical formulation of the builder bounds filter is:

$$\begin{cases} x_{\text{builder,min}} = x_{\text{ref}} \cdot (1 - \beta_{\text{min}}) \\ x_{\text{builder,max}} = x_{\text{ref}} \cdot (1 + \beta_{\text{max}}) \end{cases} \quad (3.37)$$

in which:

- $x_{\text{builder,min}}$: minimum allowed builder bound;
- $x_{\text{builder,max}}$: maximum allowed builder bound;
- x_{ref} : nominal value for the DOE point (defined by combination of the working values);
- β : builder margin. It defines the relationship between nominal and maximum/minimum value. For example:

$$\beta = 0.1 \rightarrow x_{\text{builder,min}} = 0.9 \cdot x_{\text{ref}}, \quad x_{\text{builder,max}} = 1.1 \cdot x_{\text{ref}}$$

If $\beta = \beta_{\text{min}} = \beta_{\text{max}}$, bounds are symmetrical.

The most important consideration for this filter must be done about the value of β . This filter is a simplified process that tries to predict the trend of allowed design regions for components. The only reliable sources about builder bounds are aircraft design manuals like Raymer *Aircraft Design: A Conceptual Approach* [41], in whose the uncertainty about nominal and boundary limits is stated around $\pm 20 - 30\%$. The values are not prescriptions, but they define a path in which every user can adapt the numbers to needs. Due to uncertainty on components behaviour, β value has been set to $\beta = 0.5$.

This filter is a preliminary tool that allows the user to be confident about the results, but must not be confused with technical prescriptions. Once the optimal solutions are defined, they must be validated with this filter as well.

Feasibility Evaluation with PoV function

It is possible to state that a DOE point (and therefore that simulated design) is feasible (value "1") if three conditions have been respected:

1. mission completed within prescriptions;
2. numerically valid sizing;
3. peak values within builder bounds.

Starting from the feasible and unfeasible points distinction, PoV function can be built. PoV function is a surrogate function that predicts whether a certain combination of design variables can lead to a successful simulation. It defines the former as a normalized value between 0 and 1, with 0 meaning 0% of probability to succeed and 1 value meaning 100%.

The mathematical method used to build PoV is called Random Forest. Random Forest Classifier is identified as one of the best solutions to build PoV functions [45]. It builds decision trees trained on random data and variables sub-groups, each of those divides input space in feature threshold groups. For each observation, trees produce forecasts combined to give the final probability evaluation [66].

Surrogate functions construction (training and testing) will be better explained in the following sections. The PoV function is now defined and ready. It can be used to build a second DOE, now called DOE Level-2 (DOE L2) populated with more feasible points to allow a better surrogates definition for mass and power. To create the DOE, a pool of 3000 candidates is built with Latin Hypercube Sampling. It is interesting to remark that this new DOE is not build inside original λ bounds, but the code evaluates whether there are some DOE L1 areas completely unfeasible and located at the original borders. If some unfeasible areas are found, the λ range is narrowed, excluding completely unfeasible areas.

LHS generated 3000 design points. This number must at least double DOE L2 size in order to give enough possible feasible candidates. DOE L2 has been fixed to 1000 candidates. From the 3000 candidates, the program can select a sufficient number of feasible points. Every candidate from the pool is investigated with the PoV function in order to identify:

- **high-PoV region:** points with a higher PoV value than the acceptance threshold, set at 80%:

High-PoV: $PoV > 0.8$

- **mid-PoV region:** points with a lower PoV value than the acceptance threshold but higher than minimum threshold, set at 30%:

$$\text{Mid-PoV: } 0.3 < PoV < 0.8$$

- **low-PoV region:** points with a lower PoV value than the minimum threshold:

$$\text{Low-PoV: } PoV < 0.3$$

DOE L2 can be built with the three regions:

- 80% of points from high-PoV region;
- 20% of points from mid-PoV region;
- fallback points from low-PoV region if the planned amount of points from previous regions is not reached.

Thanks to this DOE building technique, aleatory nature of DOE is not lost (pool candidates are still randomly generated), but PoV has been used to increase the number of feasible candidates without performing a higher number of simulations. Indeed, only the 1000 selected points are simulated, while the remaining 2000 are just created and evaluated with PoV. This is the real advantage of using PoV: the possibility to save the majority of computational cost while previewing in advance whether the design point might be acceptable. Without PoV-driven DOE construction, the program must have undergone all 3000 design points simulations to achieve the same number of feasible points. The workflow passed from around 20% of success rate to 60%-70%, with around 675 feasible points out of 1000.

Figure 3.31 represents the DOE coverage map related to λ variation for an example case study. Yellow points are the feasible ones, while purple points are the unfeasible ones. First thing that the reader can notice is that the design space is very difficult to map: it doesn't have a pre-established frontier and even in areas that present several feasible points some unfeasible spots can appear. This variability is probably due the high number of design variables: since the design space is built around 10-12 dimensions, combining a high number of variables inside a design space can produce several unpredictable outcomes. The figure gives the graphical definition of "hidden constraints": parts of the design space that present unpredictable behaviours.

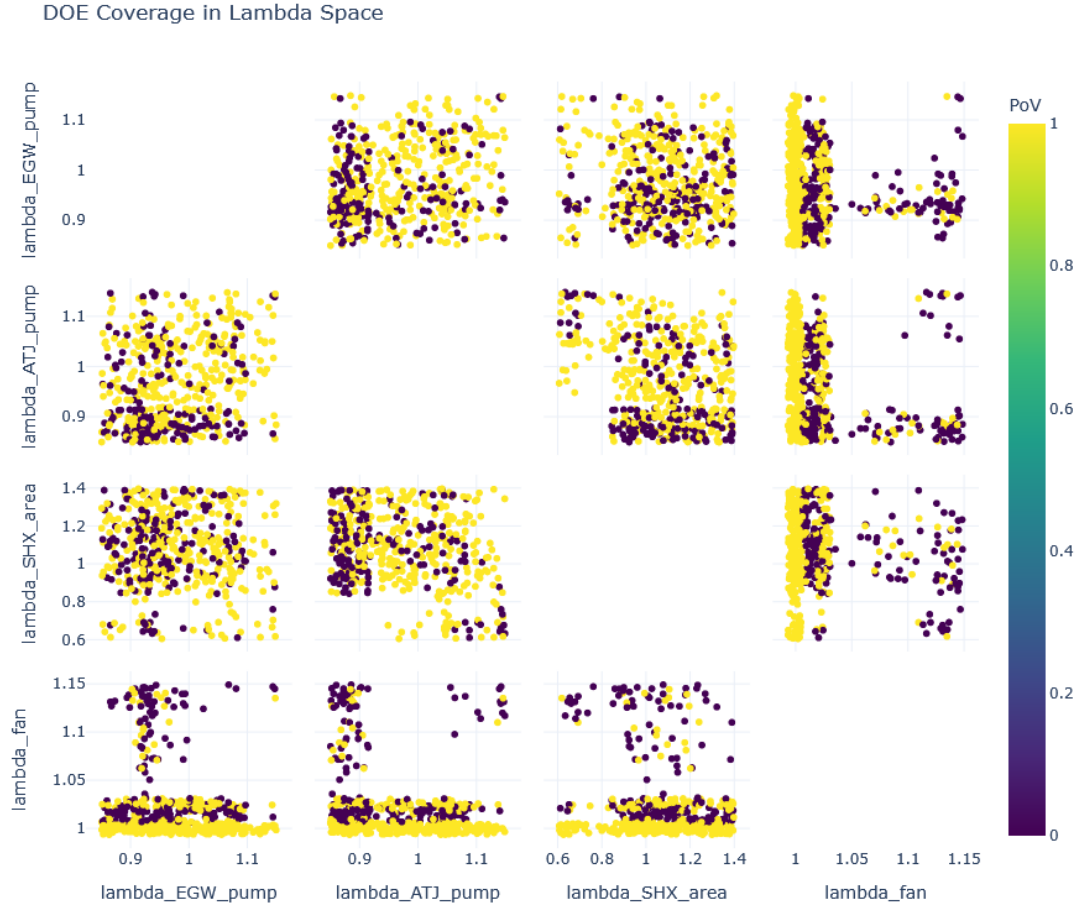


Figure 3.31: DOE L2 coverage with PoV scattering. Yellow points are the feasible generated design space points, purple points are the unfeasible ones

This feature of the design space remarks the difficulty of defining a typical optimization process and pushes to find alternative solutions, just like surrogate-based optimization.

Python implementation

DOEs construction process has been implemented in the Python script. It required two Python libraries:

- **scikit-learn**: *an open source machine learning library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection, model evaluation, and many other utilities* [67]. Used to implement RandomForest Classifier for PoV function;
- **SMT**: this package, developed by ONERA and other partners like NASA and ISAE Supaero, consists of *libraries of surrogate modeling methods (e.g., radial basis functions, Kriging), sampling methods, and benchmarking problems. SMT is designed to make it easy for developers to implement new surrogate models in a well-tested and well-document platform, and for users to have a library of surrogate modeling methods with which to use and compare methods* [68]. Used to implement Latin Hypercube Sampling method in DOE points sampling.

A pseudo-code for DOEs construction implementation is given in appendix A.1. After a first setup of variables, the script relies on an utility function which, using LHS method, builds DOE L1 in the initially defined design space. This DOE is the one used for Modelica simulations (500 for L1): the output is saved into a `.csv` file. PoV model is then trained on L1 results (feasible/unfeasible depending on simulation end, sizing results and builder bounds) file in order to choose 1000 points out of a 3000 candidates pool generated with LHS. The filtering criterion is PoV value. DOE L2 is then built and Modelica simulations are performed. Output `.csv` for both L1 and L2 DOEs is returned with PoV model, old and new `xlimits` (narrowed depending on feasibility).

3.5 Surrogate Modeling

Surrogate models are mathematical tools that can describe the behaviour of a system without simulating it. They just need a database with inputs (design variable of MDO) and outputs (objective variables of MDO). DOE L2 suits with the request: it contains around 650 input-output relationships for the model under exam. They work as "interpreters" of the design space. Once they're built, they can describe how system varies with its inputs without performing a single simulation. Their main advantage is that they can describe multi-dimensional and non-continuous design spaces.

Surrogate models reduce computational cost for heavy simulations and their successive evaluations (like the optimization process in this thesis) while keeping a sufficient reliability.

The script developed for this work builds, starting from DOE L2, three different versions of surrogate models (mass and power). The three models differ in the method that generated them:

- Kriging Method;
- Radial Basis Functions;
- Least Squares Polynomial Method.

The three methods perform the same task: building two surrogate functions, one for total system mass and one for peak electrical power required.

Kriging Method

Kriging method is based on probabilistic regression to approximate an unknown deterministic function, modeling it as realization of a Gaussian Process (GP) [69].

The key idea of Kriging method is interpolating known points and providing for each of them both an estimation of average and uncertainty values.

Ordinary Kriging method models the function as [69]:

$$y(x) = \mu + Z(x) \quad (3.38)$$

in which:

- $y(x)$: surrogate function;
- μ : global average value, unknown;
- $Z(x)$: gaussian process with zero average and not zero co-variance [69].

The final Kriging prediction, given a DOE point $\mathbf{x}^*$, is defined as [69]:

$$\hat{y}(x^*) = \mu + \mathbf{r}^T \mathbf{R}^{-1} (\mathbf{y} - \mu \cdot \mathbf{1}) \quad (3.39)$$

in which:

- $\mathbf{R}$: correlation matrix among DOE points;
- $\mathbf{r}$: correlation between $\mathbf{x}^*$ and DOE points;
- $\mathbf{y}$: "known answers" vector.

Main mathematical properties of this function are:

- it perfectly interpolates known points:

$$if \ \mathbf{x}^* = \mathbf{x}_i \Rightarrow \hat{\mathbf{y}} = \hat{\mathbf{y}}_i$$

- data-wise linearity;
- variable-wise non-linearity.

The three main advantages of Kriging method are:

1. it can deal with strongly non-linear environments;
2. it needs relatively few data;
3. it provides useful information like mean and variance.

Radial Basis Functions

Radial Basis Functions approximate unknown deterministic functions as linear combination of radial functions, each of those centered in DOE points. The key idea of this method is that the influence of each sample only depends on distance from validation point [70].

It represents a less computationally expensive method than Kriging, but it doesn't provide metrics and it doesn't rely on probabilistic evaluations.

The most common mathematical definition for Radial Basis method is [70]:

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^N w_i \phi(\|\mathbf{x} - \mathbf{x}_i\|) + p(\mathbf{x}) \quad (3.40)$$

in which:

- $\mathbf{x}_i$: DOE points;
- w_i : weights. Determined imposing:

$$\hat{y}(\mathbf{x}_j) = y_j \forall j, j \text{ DOE points}$$

and then passing through a linear system to obtain w_i ;

- ϕ : radial function. It can be of several kinds: Gaussian, Multi-quadratic, Inverse Multi-quadratic, Thin Plate Spline or Linear;
- $\mathbf{x}$: polynomial term, optional.

An important parameter is the shape factor ε , included inside radial functions ϕ . If too high, it produces wider confidence areas but less reliable surrogates, while if too low it can bring to over-fitted functions. Its value must be decided for each different case, in order to avoid both under-fitting and over-fitting conditions [70].

One important feature of Radial Basis Functions is their infinite derivability, which makes them ideal for gradient-free optimizations but less robust than Kriging derived functions [70].

The main advantages for this kind of functions are:

- very good model for screening and comparison;
- suitable for large DOEs;
- ideal for repeated evaluations;
- is a valid alternative if Kriging model is badly conditioned.

Least Squares Polynomial

Least Squares Polynomial method approximates a deterministic function with a global polynomial function whose coefficients are evaluated minimizing mean quadratic error on DOE points [71].

It is a purely deterministic function that doesn't interpolate DOE points, achieving high stability and robustness.

System response is defined as a global polynomial combination of design variables [71].

The mathematical definition of the model is [71]:

$$\hat{y}(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\beta} \quad (3.41)$$

in which:

- $\boldsymbol{\phi}(\mathbf{x})^T$: polynomial basis vector;
- $\boldsymbol{\beta}$: vector of unknown coefficients.

As an example, a second order polynomial is now shown. Given n variables [71]:

$$\hat{y}(\mathbf{x}) = \beta_0 + \sum_{i=1}^n \beta_i x_i + \sum_{i=1}^n \beta_{ii} x_i^2 + \sum_{i < j} \beta_{ij} x_i x_j \quad (3.42)$$

with N_{coeff} terms [71]:

$$N_{coeff} = \frac{(n+1)(n+2)}{n} \quad (3.43)$$

To estimate the coefficients, given a DOE with N samples:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \quad (3.44)$$

the classical Least Squares solution is:

$$\boldsymbol{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \quad (3.45)$$

in which:

- $\mathbf{X}$: design matrix;
- $\boldsymbol{\epsilon}$: residual error.

If $N \gg N_{coeff}$, the problem provides an acceptable solution with low computational cost. It is a differentiable function but is important to remember that it doesn't interpolate, it approximates. It can present bad conditioning or overfitting problems [71]. It is often used as baseline surrogate to check physical validity and global trends.

Surrogate Model Choice

In order to give more flexibility to the code, the surrogate model definition method has not been fixed. Each of the previously explained surrogate model methods can be the one that describes system behaviour. The choice among these three is taken by metrics evaluation: each different DOE, architecture of system can provide a different design space exploration, which requires more flexibility in order to keep a high resolution. The two metrics that drive the choice are:

- *RMSE*: Root Mean Square Error. Measures mean quadratic error between surrogate-predicted values and the reference ones (DOE) in the same units of the database [72]:

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2} \quad (3.46)$$

in which [72]:

- y_i : real (database) value;
- $\hat{y}_i$: surrogate-predicted value;
- N : number of validation samples.

It is an absolute indicator of accuracy, in which $RMSE = 0$ is the best solution, because it would mean that there's no discrepancy between surrogates prediction and simulated points;

- R^2 : Coefficient of Determination. Quantifies variance in output between surrogate models and "null models", an average of data [72]:

$$R^2 = 1 - \frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{\sum_{i=1}^N (y_i - \bar{y})^2} \quad (3.47)$$

in which [72]:

- y_i : real (database) value;
- $\hat{y}_i$: surrogate-predicted value;
- N : number of validation samples;
- $\bar{y}$: data average.

R^2 value interpretation is:

- $R^2 < 0$: the model predicts worse results than the null model;
- $R^2 = 0$: the model predicts with the exact same accuracy as null model;
- $R^2 > 0$: the model makes better predictions than the null model;
- $R^2 = 1$: perfect prediction. A unitary value is often seen as an index of overfitting. For this reason, very often $R^2 = 1$ in phases that are not model training is seen as a negative indicator by experts.

Once obtained the two values for each of the surrogate models (from the three methods used), the program chooses the one that shows a lower value of $RMSE$ and a higher value of R^2 .

Building three different surrogate models has a low computational cost, but allows having flexible surrogate models construction.

Python Implementation

All three surrogate models have been built with Python **SMT** package [68]. **SMT** is one of the most complete, reliable and easy to use tools in Python environment. It allows, with just few lines of code, building surrogate models from large database in a quasi-instantaneous calculation.

A pseudo-code regarding surrogate models construction and implementation is shown in appendix A.2.

It first loads the DOE `.csv` file and its related data (returned by former pseudo-code in section 3.4). Then, if target variables are expressed in logarithmic forms (used to give more stability to the optimizer), it translate them into the physical value. It then reads `xlimits`, the boundaries in which surrogate validity is guaranteed, and then starts the surrogate modelling process: it splits the database into 80% of points that will train the model and 20% that will test it to obtain the $RMSE$ and R^2 metrics.

Then, models training takes place: the script builds surrogate models with the three different methods and finally evaluates its metrics to choose the best one, which will become the one used by the optimizer. Indeed, the script returns the selected surrogate model and its related metadata, ready for the optimization phase of the workflow.

3.6 Surrogate-Based MDO

Adding an optimization workflow to engineering design processes allows to explore more design possibilities and drive designers focus to more convenient options [10], as shown in figure 3.32. This exploration allows to improve every aspect of the design, finding the best trade-off among several topics and disciplines.

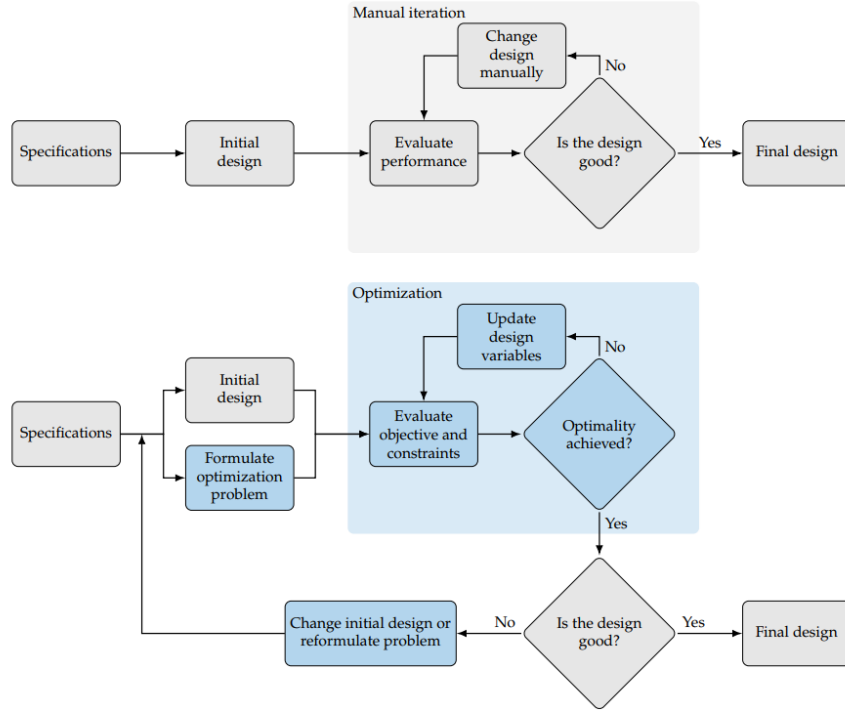


Figure 3.32: Comparison between classical design process (top) and optimized design process (bottom) [10]. Optimization adds more loops and decision blocks to the workflow

With the term "objective variables", literature usually defines the output variables that have to be minimized or maximized [10]. Applied to this workflow, objective variables are total mass and peak electrical power: both of them have to be minimized.

At the same time, some inputs must be given to the process. Inputs, or "design variables", are varied inside the design space ranges to find, with several evaluations, the best combination of objective variables. Being a bi-dimensional (two objectives) problem, the best graphical representation for results is a 2D Pareto front (figure 3.33). The plot shows, with the two objectives on the x and y axes, the explored population and the best options, usually at the borders of the region. This frontier is called "Pareto front" [10].

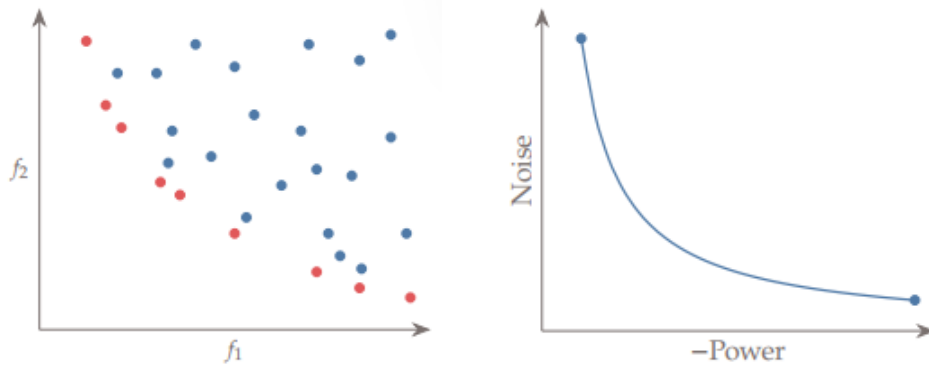


Figure 3.33: 2D Pareto plot. On the left: all the evaluated points plotted with respect to the two objectives, f_1 and f_2 . On the right, an example of Pareto front [10]

Moving to the mathematical formulation of the MDO problem, it is usually defined as [10]:

$$\begin{aligned} & \text{minimize/maximize} && f(x) \\ & \text{with respect to} && x_{i_{\min}} \leq x_i \leq x_{i_{\max}} \\ & \text{subject to} && g(x) \leq 0 \end{aligned}$$

in which:

- $f(x)$: objective function;
- x_i : design variables;
- $g(x)$: inequality constraint.

Regarding this workflow, a preliminary definition of the MDO problem is:

$$\begin{aligned} & \text{minimize} && \min \mathbf{f}(\lambda) = \begin{bmatrix} f_1(\lambda) = m(\lambda) \\ f_2(\lambda) = P(\lambda) \end{bmatrix} \\ & \text{with respect to} && \lambda_{\min} \leq \lambda \leq \lambda_{\max} \\ & \text{subject to} && \chi(\lambda) = 1 \end{aligned} \tag{3.48}$$

in which:

- λ : design variables;
- $\mathbf{f}(\lambda)$: objective functions set;

- $m(\lambda)$: total mass objective function;
- $P(\lambda)$: peak electrical power objective function;
- $\chi(\lambda)$: Boolean feasibility indicator. If equal to 1, it indicates a feasible design point.

When MDO is then applied to surrogate models (Surrogate-Based MDO), the problem formulation changes in:

$$\begin{aligned}
 & \text{minimize} \quad \min_{\lambda} \quad \hat{\mathbf{f}}(\lambda) = \begin{bmatrix} \hat{f}_1(\lambda) = \hat{m}(\lambda) \\ \hat{f}_2(\lambda) = \hat{P}(\lambda) \end{bmatrix} \\
 & \text{with respect to} \quad \lambda_{\min} \leq \lambda \leq \lambda_{\max} \\
 & \text{subject to} \quad g(\lambda) = PoV_{\min} - \hat{\pi}(\lambda) \leq 0
 \end{aligned} \tag{3.49}$$

in which:

- λ : design variables;
- $\hat{\mathbf{f}}(\lambda)$: surrogate objective functions set;
- $\hat{m}(\lambda)$: total mass surrogate objective function;
- $\hat{P}(\lambda)$: peak electrical power surrogate objective function;
- $g(x)$: inequality constraint;
- $PoV_{\min}$: PoV value threshold;
- $\hat{\pi}(\lambda)$: PoV surrogate function. It can be defined as the probability that the design point is feasible:

$$\hat{\pi}(\lambda) = \mathbb{P}(\chi(\lambda) = 1), \quad \hat{\pi}(\lambda) \in [0, 1] \tag{3.50}$$

Optimization Algorithm

After several researches among literature references, NSGA-II algorithm has been chosen. It is the same algorithm used in Coutinho work [11], confirming the suitability of the algorithm in this field.

NSGA-II algorithm (Non-dominated Sorting Genetic Algorithm II) is a genetic, multi-objective optimization algorithm [19]. It is one of the most used algorithms when objectives are in conflict among them and it is not convenient or possible using a gradient-based method [19].

It starts by creating a population of candidates and evaluating them on objectives. Missed respect of constraints is also evaluated. For the mating selection, NSGA-II uses a binary tournament:

1. it compares ranks, preferring points with lower rank;
2. for candidates with the same rank, the one with higher crowding distance is selected in order to enhance diversity among evaluated point.

From the existing solutions, the algorithm creates new candidates (genetic variation), from both combination or random modifications of the existing ones. This way, the algorithm expands its sight.

Then, the algorithm joins old and new solutions and selects the best ones to create the Pareto front. If too many solutions are similar, the most different ones are chosen. Preserving good solutions for further steps is called *elitism*. With elitism, no solution is lost and the Pareto front is not focused on just one area of the design space [19].

The main advantages of using NSGA-II in the proposed workflow are:

- is designed for multiple conflicting objectives;
- since surrogate models are not smooth and affected by noise, NSGA-II represents a valid alternative because it doesn't need gradients and it is built to be reliant to noise;
- since NSGA-II requires an elevated number of evaluations, surrogate-based optimization is the best option to reduce computational cost;
- since the workflow identifies solutions in different (and independent) steps, elitism helps in preserving good solutions and avoiding regressions;
- produces a well-defined Pareto front, helping in further validation processes.

Python Implementation

Two libraries have been used to implement MDO in Python language:

- **SBArchOpt**: Surrogate-Based Optimization tools set [47]. Library built to deal with surrogates of systems and their optimization, considering hidden constraints and speeding up the process with respect to conventional libraries;
- **pymoo**: *state of the art single and multi-objective optimization algorithms and many more features related to multi-objective optimization such as visualization and decision making* [73]. One of the most used libraries for optimization algorithms in Python language.

Python implementation (shown in appendix A.3) starts by defining the research box from previously defined `xlimits`. Design variables (lambdas) are then imported and verified with builder bounds. The following step is defining the optimization problem. This procedure is performed inside `TmsSurrogateProblem` class. This class wraps mass, power and PoV surrogates into a **SBArchOpt**-compatible environment. It translates the optimization problem formalization performed in previous sections into a Python (and optimization libraries like **SBArchOpt** and **pymoo**) understandable format, defining design variables set, objectives and constraints. Then, all surrogate functions are stored. An optimization algorithm must then be chosen.

Problem optimization is then performed: problem, algorithm and metadata are passed to `minimize` function from **pymoo**. This function, given an optimization problem, its related algorithm and some more information, performs the minimization of objectives with respect to constraints and the optimization algorithm. Results are then returned with the related problem.

3.7 Pareto Validation

Since the obtained Pareto front has been obtained from multiple approximations (surrogates), reliability of the workflow is not guaranteed until it is not proved with simulations. The following chapter will explain validation for each stage of the workflow, but the validation of Pareto front points is not intended as a verification, but instead it is intended as the final step of the process, in which only realistic results are accepted.

All Pareto points (in the case of this work 100 points) are re-simulated in Modelica, going through the previously explained filtration process, considering the three feasibility requirements. Finally, the sizing process is performed and its outputs are compared with surrogates prediction (the values that formed the optimization output). Only points below a certain threshold of relative error are accepted.

To pass this final filter, points must respect the threshold in both mass prediction and power prediction at the same time.

The threshold value has been defined starting from literature indications. As stated in chapter 2.5.3, a common threshold for validation is 5% of relative error.

Filtered and validated results are then plotted in a new Pareto front. The latter is the real output of the workflow: a set of design points that has undergone 2 DOEs, modeled into a surrogate, optimized and finally validated to ensure that the result of the optimization exists in real-world applications. The final points are a realistic and reliable estimate of a design that trades off total system mass and peak electrical power. The choice among this set of points is then left to the designer, who must consider case by case design requirements, moving towards less heavy or less power-demanding designs.

Python Implementation

The first step to implement the filter in Python language (pseudo-code shown in appendix A.4) is re-simulating all points, with a similar procedure to DOEs. The only difference in this case is that the output `.csv` reports both simulation results and surrogate predictions.

Then, once loaded the validation report, relative error is computed and compared with the threshold. Only points below the threshold (set at 5% for both mass and power) pass this filter, forming the final set of Pareto front points, plotted in the two final plots.

3.8 General Workflow Wrap-Up

The complete workflow proposed in this thesis is summarized below and in figure 3.34.

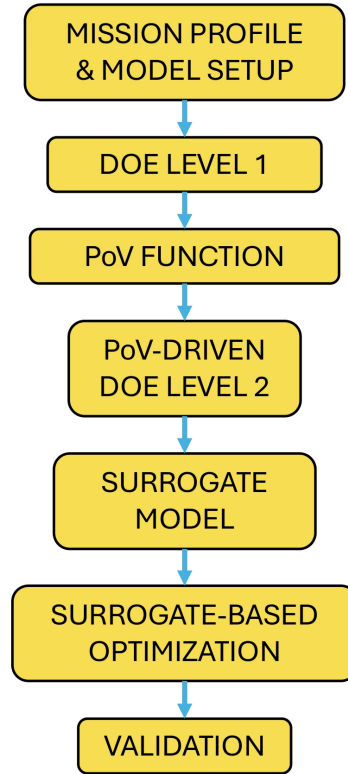


Figure 3.34: Complete workflow proposed in this thesis. It is composed by seven consecutive steps

First, mission profile is defined in terms of altitude, speed and time. Then, the architecture (and its related model) has to be chosen among the five possible ones. This step concludes the setup part of the workflow.

Since all the models present "hidden constraints" (e.g. failed simulations) and some of them take more than two hours to perform a simulation, a direct optimization of models was not a viable solution.

Surrogate-based optimization is indeed more indicated. DOE is necessary in order to build a surrogate function. Then, first DOE is built by varying selected design variables inside the design space and re-simulating the model in Modelica. To reach a sufficient number of feasible simulations a high number of simulations would be necessary. To avoid that, building a second DOE with almost only feasible

points is the best solution. PoV functions are used. PoV functions and design space restrictions are both driven by feasibility indications: simulation successfully ends, finite sizing values and builder bounds respected. Starting from these two requirements, feasible areas (for bounds narrowing) and PoV function are built and used. Then, starting from a large pool of candidates (inside the narrowed box) design points with high PoV are chosen: starting from a DOE L1 with only 20% of feasible points, a second (level 2) DOE with 70% of feasible points is obtained. PoV-driven L2 DOE represents a suitable option to build surrogate models for the two objectives: total system mass and peak electrical power. Optimization is then performed on surrogates, producing a Pareto front, the optimal points set. Pareto front is then filtered. In order to validate the optimization results, the Pareto front set is re-simulated with Modelica and only the points that present a lower relative error (surrogate-predicted points in Pareto front versus re-simulation) than a defined threshold are finally accepted as output of the process.

If the reader aims to a better understanding of the MDO process, *XDSM* diagrams [74] can be used. *XDSM* for the workflow under exam is shown in figure 3.35.

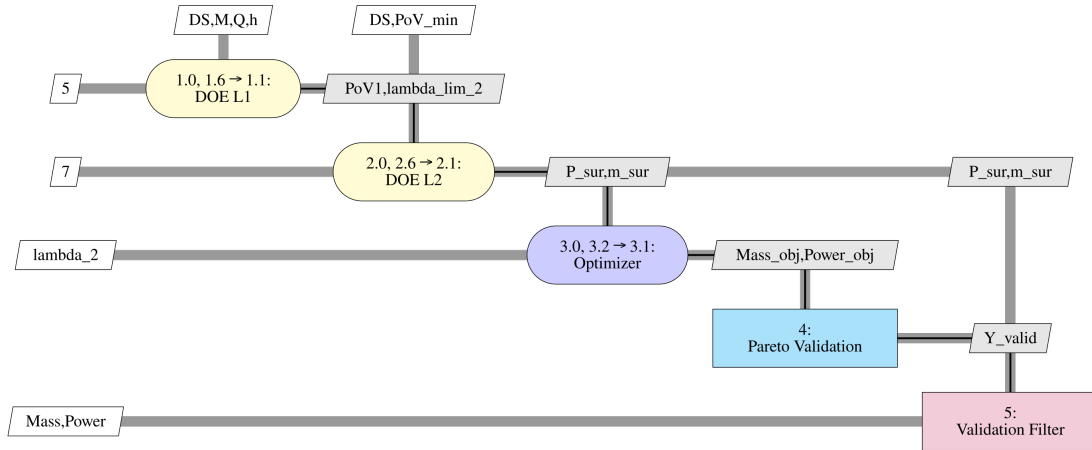


Figure 3.35: Workflow *XDSM* diagram. The two DOEs (yellow boxes), the optimizer (purple box) and the validation filter (blue box and red box) can be seen

Once performed DOE L1 and DOE L2, the optimization loop works on mass and power surrogates, with the latter and the new design space indications (lambdas) as input. After looping on these items in order to find optimized solutions, the Pareto front is built (*Mass_obj*, *Power_obj*).

Detailed *XDSM* diagrams for DOE L1, DOE L2 and MDO are shown in appendices B.1, B.2 and B.3.

Chapter 4

Validation

Each part of the workflow must be validated in order to ensure that every result that will be obtained is scientifically valid.

The aim of the validation process is to validate the entire workflow proposed in former chapters. To do so, one architecture of the five proposed has been chosen. In the case of this workflow, Architecture 5 (figure 3.24) is chosen. The choice was driven by completeness and processing time, because Architecture 5 presents the highest number of different components (it only misses the VCS cycle). Moreover, it is one of the models with the shortest runtime.

Once Architecture 5 has been validated, the reader can be sure that:

- almost all architectural components are valid (only VCS cycle is missing);
- the workflow is valid: it is the same for all architectures. Once applied some small changes to the code, it can be applied to any Modelica TMS.

DOE and MDO will not be validated as standalone tools, they already rely on validated and scientifically verified papers and tools. They will be validated instead in terms of robustness and suitability for the proposed workflow.

4.1 Modelica Models Validation

The validation of modeling process will be focused on proving that the models behaviour and composition is consistent and they may be a good starting point for the rest of the workflow. A bottom-up approach will be followed, starting from single components and then moving to the system and its behaviour.

4.1.1 Model Components

Since every component of the model has been sized starting from datasheets, the validation requires to check whether, once sized, the component behaves as described in the datasheet.

Taking as example the EGW pump, it has been sized with its datasheet. The latter states that, during a nominal working regime at the prescribed mass flow (table 3.9), it should be able to provide 0.85 *bar* of pressure differential [13].

As shown in picture 4.1, this requirement is respected during most of the flight.

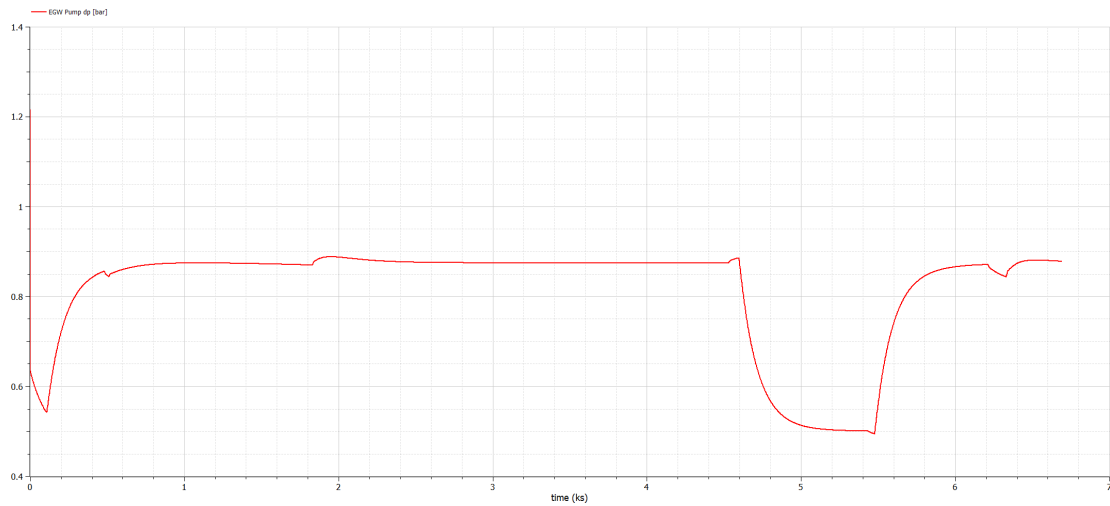


Figure 4.1: EGW pump pressure differential throughout the defined mission in OpenModelica plotting window

This procedure has been followed for every component in the model.

4.1.2 System Behaviour

System behaviour must be validated. The validation will focus on:

- temperatures and heat loads orders of magnitude;
- mission-wise coherent behaviour;
- numerical robustness.

Order of Magnitude

Starting from temperatures, figure 4.2 shows the temperatures trend of the two fluids in the loops (EGW and ATJ).

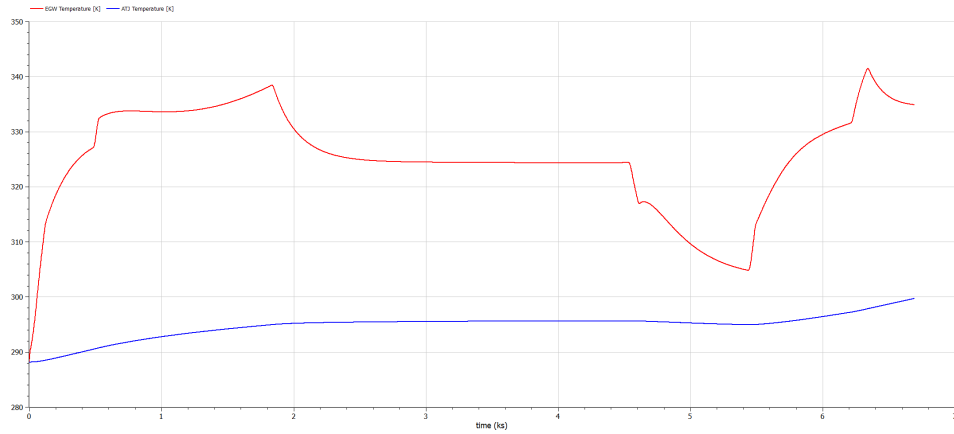


Figure 4.2: EGW (red) and ATJ (blue) temperatures during the reference mission. Plotted in OpenModelica plotting window

Focusing on EGW temperature, excluding variations due to the mission phases, it always lies between 305 K and 340 K , respecting indications from datasheet [55]. Then, focusing on ATJ, it also respects indications [62] by fluctuating around 295 K .

The difference in trends is due to the positioning of the two loops: while EGW is in direct contact with the thermal load, ATJ receives the heat from EGW. EGW acts as a "filter" for abrupt thermal load changes.

Moving to heat loads, figure 4.3 shows the trends for the heat load, the Ram-Air Heat Exchanger (RHX), the Fuel Heat Exchanger (FHX) and the Skin Heat Exchanger.

Trends are perfectly coherent with the imposed heat load, shown in figure 3.27. This means that the heat flux is brought from input port (red line) to the output

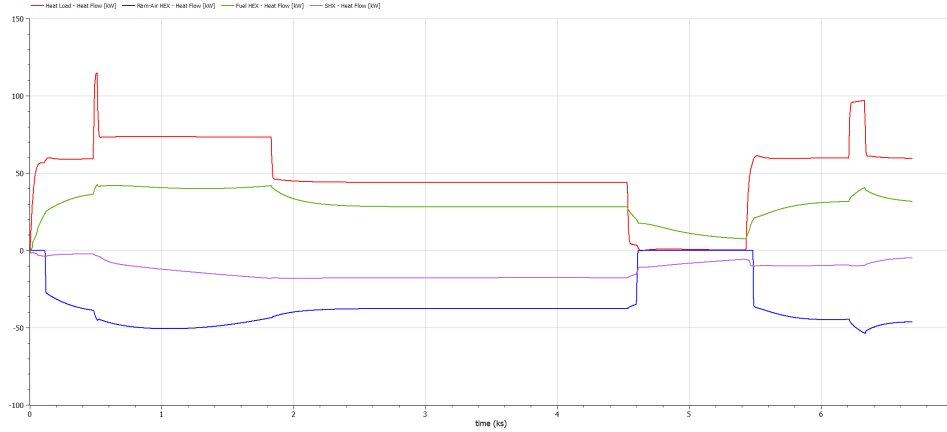


Figure 4.3: Heat load Heat Flow (red), RHX Heat Flow (blue), FHX Heat Flow (green), SHX Heat Flow (purple) throughout the reference mission. Plotted in OpenModelica plotting window

heat ports (blue and purple lines), passing from intermediate exchanges (green line). Values sign is also respected: heat ports that absorb heat have positive heat flows, while heat ports that reject heat show negative values.

Finally, orders of magnitude are respected too, with the values moving between 20 kW and 110 kW (positive or negative depending on absorbed/rejected heat).

In conclusion, it has been proven that the heat load is efficiently absorbed, managed without exceeding temperature bounds and rejected.

Mission-wise Behaviour

Mission-wise behaviour validation wants to focus on the response of the system to the different mission phases. Taking as reference the standard mission profile, the aim is to verify that a change of flight phase (and consequent heat load change) produces a reasoned change in system conditions. To perform this analysis, the main reference are figures 3.25, 3.26, 3.27 and table 3.2. This way, the mission profile is fully described. To validate the model, it is expected to associate trend changes in main system variables with mission profile data (and consequently phase) change.

The variables to be checked are:

- fluid temperatures (figure 4.2): they present peaks around 500 s , 1800 s and 6300 s . This is perfectly corresponding to take-off, cruise start and approach;
- heat flow (figure 4.3): they present peaks around 500 s , 1800 s and 6300 s . This is perfectly corresponding to take-off, cruise start and approach;

- fluid pressures (figure 4.4): EGW Pump shows the same peaks at 500 s, 1800 s and 6300 s. This behaviour follows heat load profile, while ATJ Pump doesn't. This feature shouldn't worry because, as said in former pages, ATJ loop sees a strongly "filtered" heat load, with less abrupt variations. ATJ loop relies also on tanks, which rise the total fluid amount, increasing fluid-dynamic inertia. The straight blue line is due to the previous features, making ATJ cycle a more stable process.

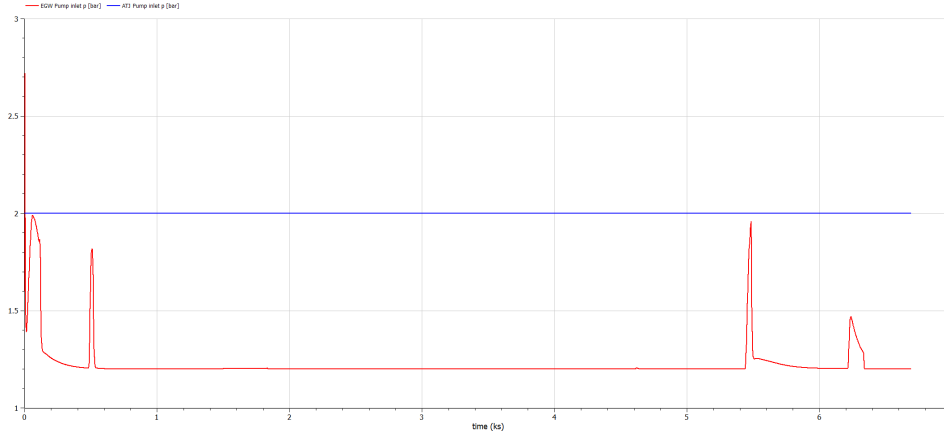


Figure 4.4: EGW Pump inlet pressure (red) and ATJ Pump inlet pressure (blue) throughout the reference mission. Plotted in OpenModelica plotting window

It is now possible to conclude that the system is sensitive to the mission profile definition.

Numerical Robustness

Finally, numerical robustness of the model must be evaluated. This check is already embedded in the Python code: for any failed simulation or `NaN/inf` value, the program discards the outputs and the entire design point. The simulation is also reproducible, because even by simulating multiple times the same design point, the outcome and the results will always be the same. Nonetheless, OpenModelica itself performs several checks on the numerical side: warnings are printed for stiff systems, unsolved differential equations and nonsense values, ensuring the user about its numerical robustness.

4.2 Surrogate Models Validation

Surrogate models represent local approximations performed inside a defined design space. They must not be used outside design space bounds to avoid loss of reliability.

Therefore, the first check to be performed is that every generated data lies inside the design space. This is automatically proven since the only Python variable used for every operation is `xlimits`, which corresponds exactly to the design space bounds.

Secondly, surrogates reliability must be proven. The workflow can prove it in two different ways:

- error metrics;
- error distribution.

Error Metrics

As explained in Methodology chapter, two error metrics ($RMSE$ and R^2) are used in this workflow to evaluate surrogates. A percentage of 20% of DOE points is only used for surrogate testing. This means that they don't train the surrogate: the latter didn't have any interaction with the test points before. Thanks to this, $RMSE$ and R^2 represent an independent verification on surrogates reliability. Each Python run shows on the prompt error metrics data. The error metrics values lie around:

$$RMSE \approx 1e - 2, \quad R^2 \approx 0.98$$

describing a very reliable surrogate function.

Error Distribution

Error distribution is strongly connected with surrogates testing. To prove surrogates validity, surrogates have been tested on DOE points and results have been compared with DOE outputs (directly from Modelica). The measurement has been performed in terms of relative error (figure 4.5). The latter is expressed as a percentage, representing the normalized difference between the prediction and the sizing result.

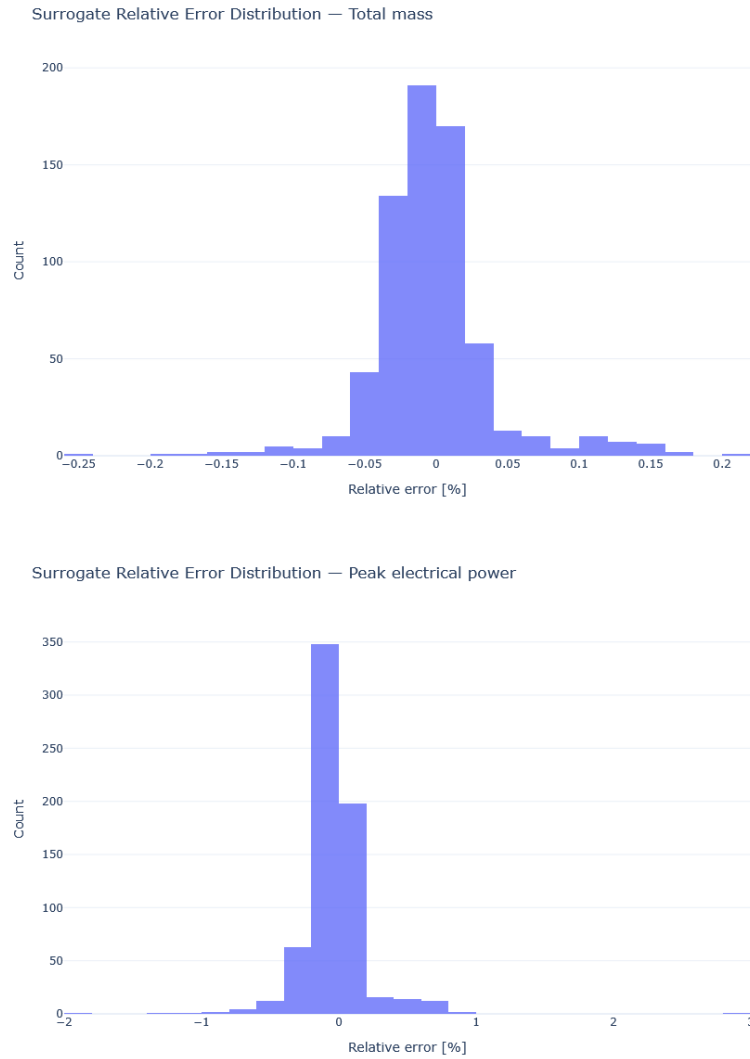


Figure 4.5: Mass surrogate (top) and power surrogate (bottom) relative error distribution histogram. In both cases, relative errors converge to 0%

Relative errors clearly converge around 0%.

MDO command lines in Python are intrinsically validated since they come from peer-reviewed libraries: `pymoo` [73], `SBArchOpt` [47].

4.3 Results Validation

Validation of final Pareto front is already embedded in the workflow (section 3.7). This validation method serves both as final workflow step and as validation test. When considered inside the workflow, it is the step that back-transforms results from surrogate models world to MBD world, giving real application to the predicted points. When considered as a validation test, it gives a proof that the predicted results are, within a defined confidence interval, adherent to simulations in the same design points.

Pareto front set is obtained as optimization output. It then must be validated in order to check whether it corresponds to a real physical optimum region. The Pareto set is simulated in Modelica. Results are then compared and relative error for each couple is calculated. Only values with a lower relative error than the threshold (5%) in both mass and power prediction pass the test.

Since this procedure is an embedded step in the workflow, results will be shown in the results chapter.

Chapter 5

Application Case and Results

The entire workflow will be reviewed, and a mathematical outcome, mostly in the form of plots, will be provided for each main step.

Modelica single simulations are not considered as steps of the workflow, they represent the case study for an application of the workflow. The first results to be analyzed are the ones related to the Level-1 DOE. Then, Level-2 DOE and resulting surrogate functions will be reviewed. Finally, the optimization process will be evaluated with results set and validity considerations on them.

5.1 Level-1 DOE

Level-1 DOE result is a scatter map (figure 5.1) that represents all the points produced and simulated for the DOE L1. They are differentiated by colour: purple points are unfeasible (e.g. failed simulation, non-finite sizing results or builder bounds not respected), while yellow points are feasible.

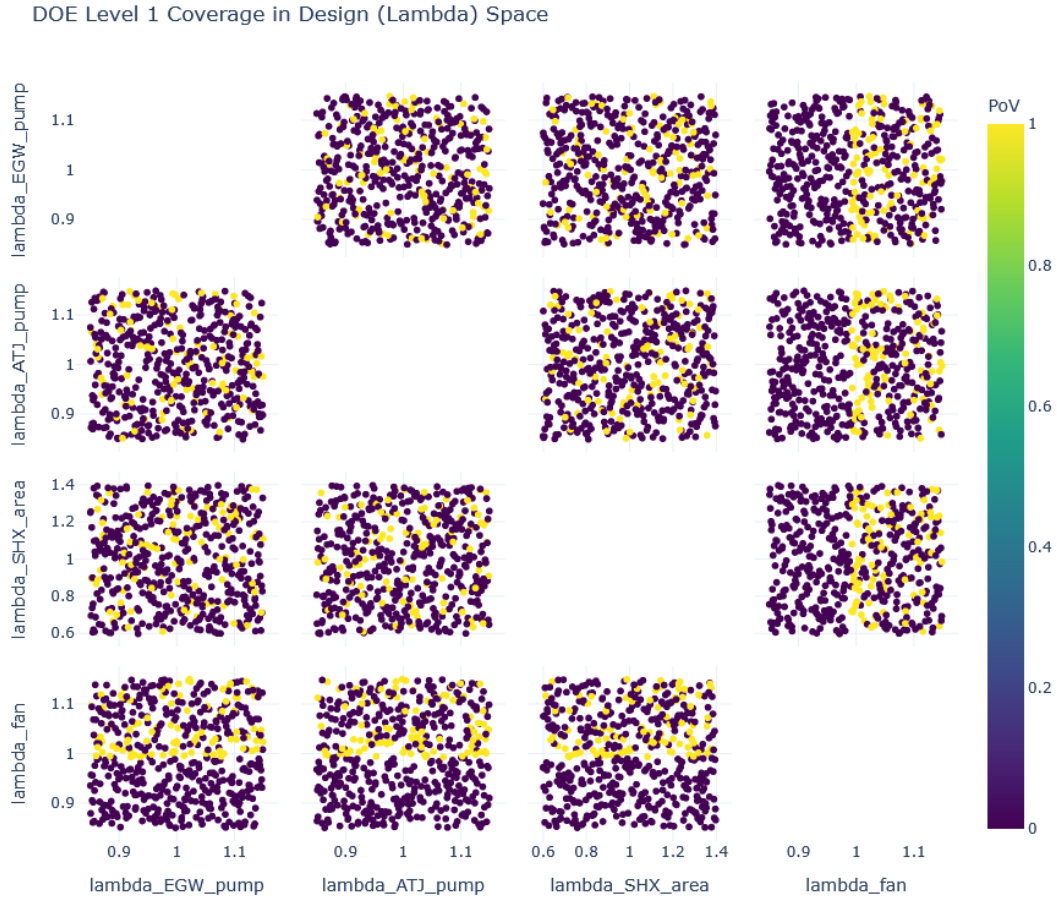


Figure 5.1: Level-1 DOE distribution plot. Yellow points are the feasible generated design space points, purple points are the unfeasible ones

The unfeasible points predominance, 407 of 500 (around 80% of the total) is clearly visible. This plot is helpful in understanding why the workflow didn't directly optimize on simulations. A larger number of points and simulations would have been necessary. With a 20% success rate, using a single DOE would have required 2500 simulations to reach 500 feasible points. This result marks the

necessity of further design space explorations and evaluations.

Another relevant aspect from the plot must be underlined. Feasible points are evenly distributed in almost all the scatter boxes, with presence of more dense local clusters. Only one component shows a strong predominance of some areas of the design space: the ram-air fan. The DOE distribution of the latter shows a clear area with feasible points around $\lambda = 1$ which also marks a border with another area for $\lambda < 1$, completely unfeasible. This result has been largely investigated during the work, and the result has been that the nominal value for the fan is already very low and close to an optimum. This unfeasible area clear marking is used by the design space boundaries restriction function, explained in Methodology chapter. This function revealed that, below a value of $\lambda_{fan} = 0.963$, no feasible simulations are found. Design space exploration for further steps has been consequently restricted to the values shown in table 5.1.

Table 5.1: Design space boundaries after DOE L1

DESIGN VARIABLE	LAMBDA RANGE AFTER DOE L1
EGW Pump	[0.850, 1.150]
ATJ Pump	[0.850, 1.150]
SHX	[0.600, 1.400]
RA Fan	[0.963, 1.150]

5.2 Level-2 DOE

PoV-Driven DOE (or Level-2 DOE) has then been performed. As expected, the success rate has risen from around 20% to around 70%.

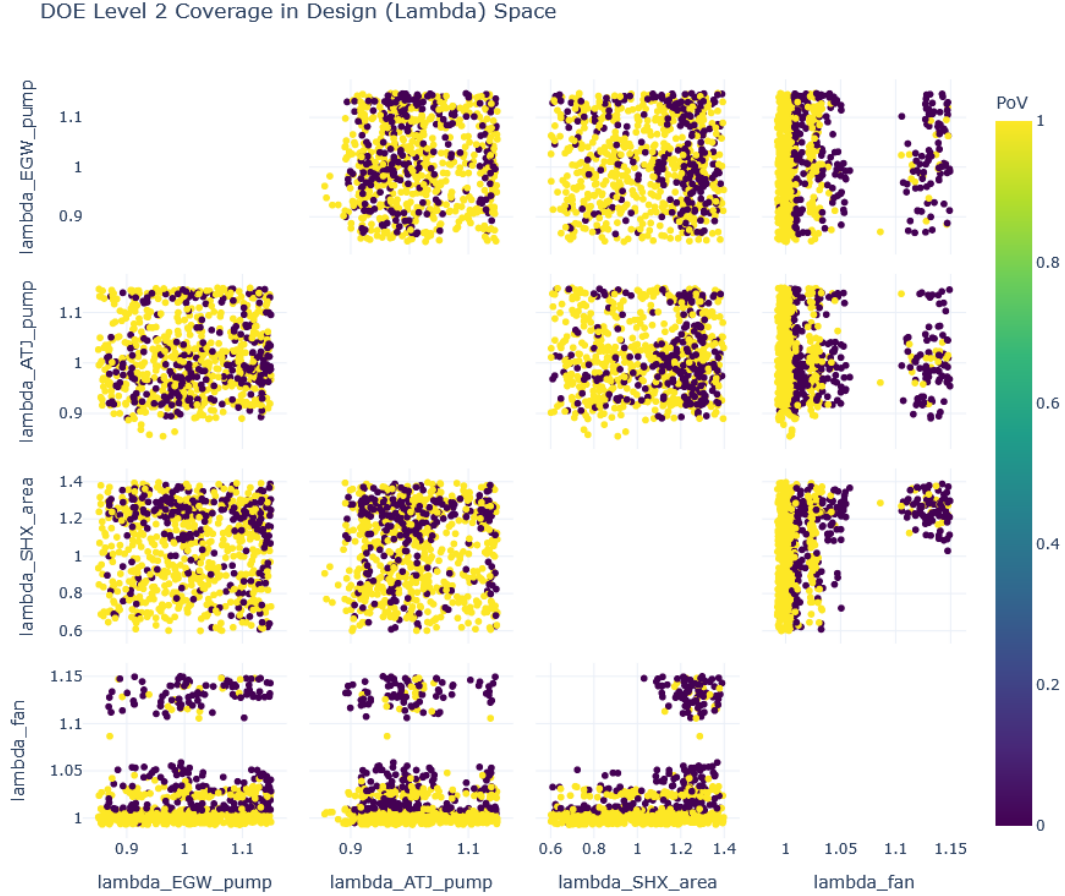


Figure 5.2: Level-2 DOE distribution plot. Yellow points are the feasible generated design space points, purple points are the unfeasible ones

DOE L2 points feasibility is shown in the same way (figure 5.2 as DOE L1. Out of 1000 points, 675 are now feasible. This is the result of filtering a larger pool with the PoV function, developed starting with DOE L1 results. Ram-air fan values range is reduced coherently with boundaries restriction, while other components scatter maps still show some feasible points cluster and some less feasible areas, allowing now the surrogates models to be trained on both feasible and unfeasible points.

SHX scatter map shows an interesting fact. When λ_{SHX} is high, unfeasible points density increases. For systems in which the reliance on the SHX is stronger, sudden changes are managed worse. This happens because SHX is not in direct contact with the heat load, but there is at least one (in Architecture 5 case two) cooling cycle between them, with mechanical components and fluids with their own thermal inertia. This shows how dynamic simulations are essential in a design process, since static simulations (like the ones performed in Coutinho's works [11], [5]) don't present any change in heat loads or thermal behaviours during time. Therefore, not performing a dynamic simulation "hides" some hidden constraints, possibly leading designers into unfeasible design space areas.

5.3 Optimization

As mentioned in earlier chapters, the output from an optimization process is a Pareto front (figure 5.3).

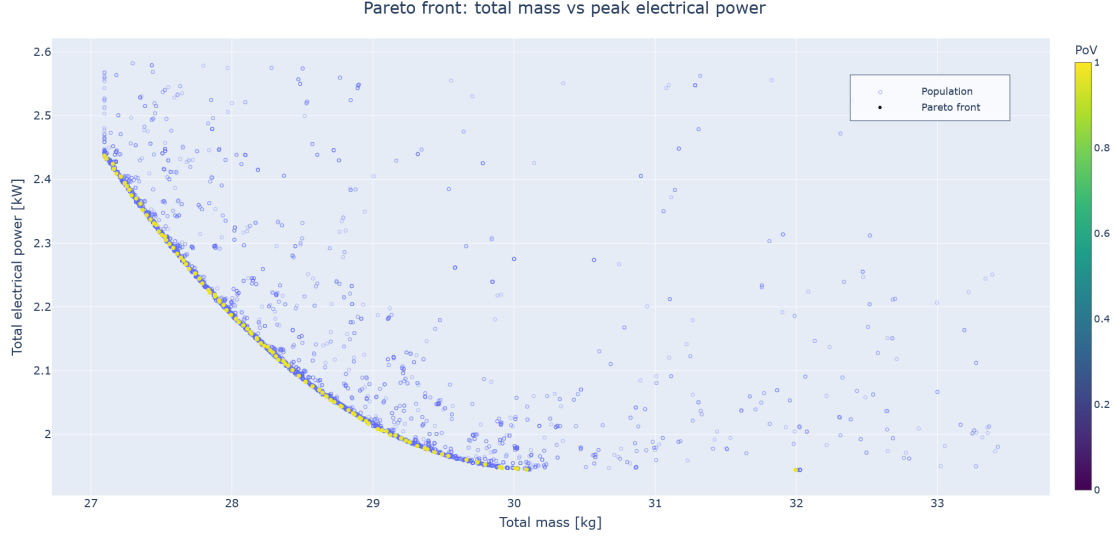


Figure 5.3: Resulting Pareto front. The front is indicated with filled points whose color depends on PoV value (right hand side colour bar). The rest of the generated population is plotted with empty points

A Pareto front set is formed by the optimal solutions. This work focuses on a bi-dimensional optimization since it has two objective functions. For this reason, the resulting Pareto front is a 2D plot. Peak electrical power is shown on y axis, while total system mass is shown on x axis.

Table 5.2: Pareto front values range

Objective	Unit	min	max
Peak electrical power	$[kW]$	1.94	2.44
Total system mass	$[kg]$	27.10	32.00

As shown in table 5.2, resulting objective ranges mark a difference of 5 kg and 0.5 kW between the two extreme points.

Once the Pareto front is produced, the choice among its points depends on designers needs. They are all non-dominated points, therefore there is not a better point of each of them in both objectives. They all represent possible trade-offs

between mass and power. If, for example, the designer can accept to have higher electrical power peaks (and therefore size a more powerful electrical system), he can move to the left side of the front and have lower masses. On the other hand, if the design can be heavier with no significant penalties, it is possible to achieve lower electrical power peaks.

It is also interesting to evaluate whether the optimization produced better results than the design space exploration alone. The minimum value for mass and power has been selected from DOE-L2 output file and compared with the Pareto points in table 5.3.

Table 5.3: Minimum values comparison for Level 2 DOE and Pareto set

Objective	Min - DOE L2	Min - Pareto	Variation
Peak electrical power	2.00 <i>kW</i>	1.94 <i>kW</i>	-2.97%
Total system mass	33.87 <i>kg</i>	27.11 <i>kg</i>	-19.99%

Total mass is strongly reduced with optimization. The minimum for system mass found in design space exploration is approximately 34 *kg*. The optimization process found, inside the same design space, a combination of design variables that produced a feasible point with just 27 *kg* of mass, reducing the previous value by 20%.

The power variation between DOE and Pareto front is lower: the value is reduced by just 3%. This could mean that the design space exploration already performed a good mapping, exploring points close to the optimum.

5.4 Pareto Front Filter

The final validation filter aims to give a validated feedback to the obtained Pareto set.

As explained in Methodology chapter, Pareto points are simulated and their output is compared with the surrogate prediction that produced the Pareto front in figure 5.3. To pass the filter, validated points must have a relative error below a defined threshold (figure 5.4). The typical threshold for high-fidelity results is 5% in relative error [50].

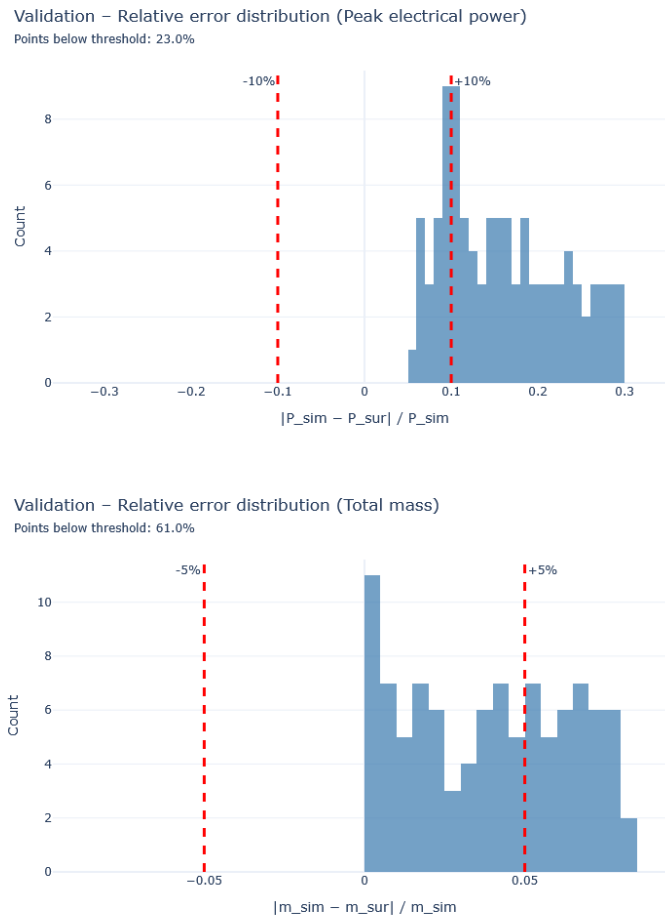


Figure 5.4: Total mass (top) and peak electrical power (bottom) validation error distribution histogram. red dashed lines mark the applied acceptance region

Focusing on mass relative error distribution in figure 5.4, 63% of Pareto front points have an acceptable relative error for the 5% threshold. Moreover, 100% of points have a lower relative error than 10%. This confirms the surrogates modeling and optimization ability to effectively predict optimal solutions for the system mass.

Moving to peak electrical power, results are slightly worse. None of the points fall under the prescribed 5% threshold. If the threshold is moved (considering the higher uncertainty seen in power surrogates) to 10%, 23 points pass the filter.

Focusing on validation outputs, they have been compared in figure 5.5, in which every Pareto front point is plotted with as x value the surrogate prediction and y value the Modelica output. The desirable condition would be having all points on the red dashed line, meaning that the predicted value perfectly matches the simulation value.

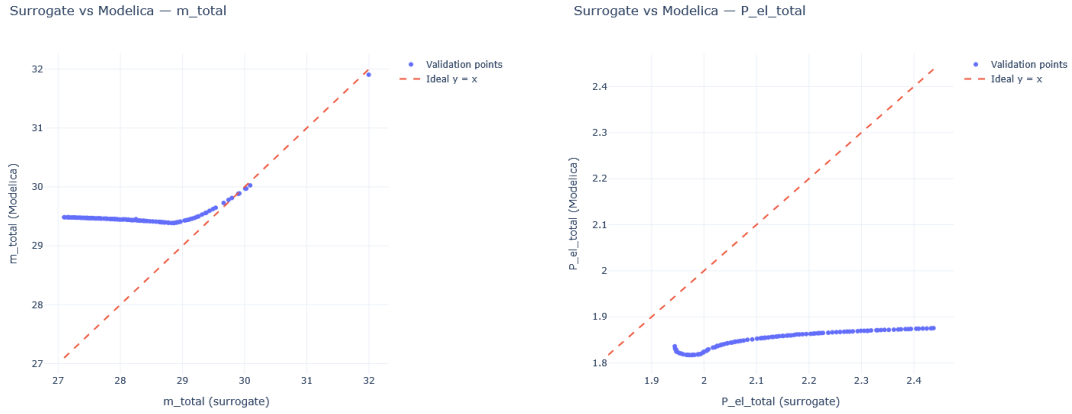


Figure 5.5: Surrogate vs Modelica values comparison: mass (top) and power (bottom). Blue points coordinates are x : surrogate-predicted value and y : value obtained from Modelica re-simulation. Red dashed line marks the ideal optimum ($x = y \Rightarrow x/y = 1$)

Analyzing the output for the mass surrogate, lower mass points seem unreliable. They start being reliable from around 28 *kg*, meaning that the optimizer brought the surrogate into a region (at the borders of the design space) in which the latter starts losing reliability.

On the other hand, low-power predictions are actually the best ones, while high powers are not reached in Modelica simulations. In this case, the power surrogate is predicting high variability and sensitivity for power, while in reality it is subject to smaller changes. The blue points curve presents a growing trend, but sensibly slower than the predicted one.

Accepting the threshold change to a slightly higher tolerance value, 23 points are available from the optimization and validation process.

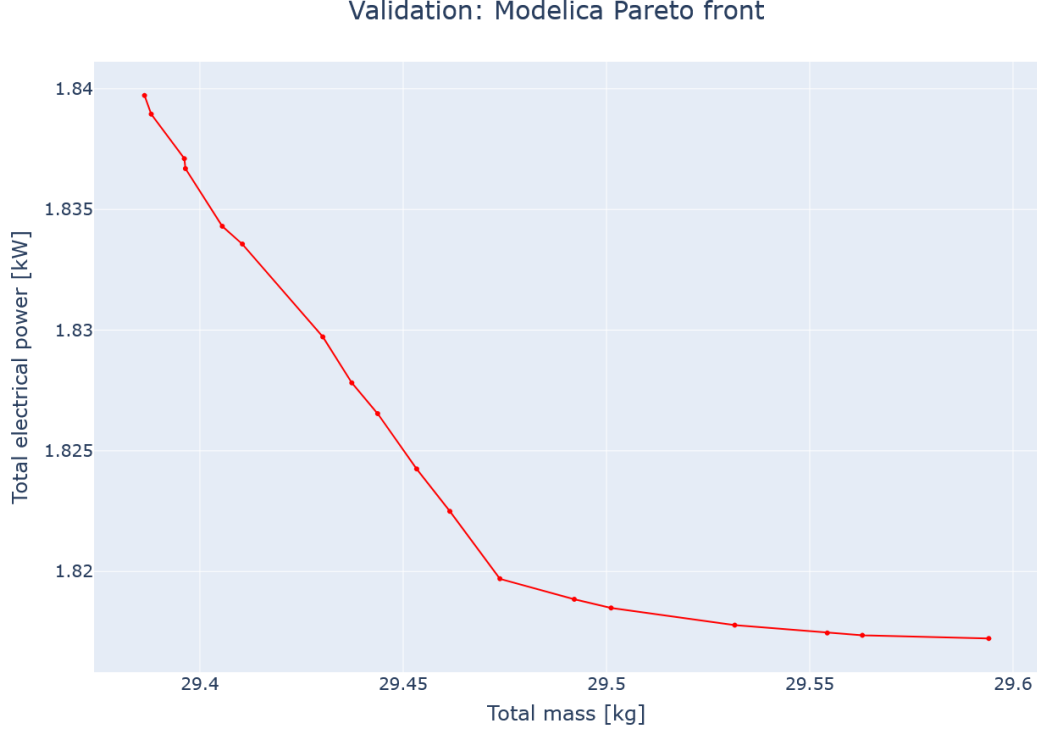


Figure 5.6: Pareto front from validation results. Pareto points coordinates indicate peak electrical power and total system mass obtained from Modelica re-simulations

Figure 5.6 shows the validated and filtered points. They form a new Pareto front, this time validated, composed by 23 points. The values span has been significantly reduced to the center section of the original Pareto front from figure 5.3, meaning that the extremities of the latter were less accurate.

Validated points values are reported in table 5.4.

The resulting trend shows that pumps should be maximized in their sizes while, on the other hand, SHX and ram-air fan should be minimized. Indeed, λ_{EGW} and λ_{ATJ} are almost matching their maximum allowed value, while λ_{SHX} and λ_{fan} are close to their respective minimums for all the validated points. This design choice prioritizes moving more coolant and faster to efficiently dissipate heat instead of having more exchangers. This fact is due to the higher mass penalty that SHX area increase brings.

Table 5.4: Validated points from final Modelica validation

λ_{EGW}	λ_{ATJ}	λ_{SHX}	λ_{fan}	m_{sur} [kg]	P_{sur} [kW]	m_{sim} [kg]	P_{sim} [kW]
1.149922	1.149942	0.654113	0.991787	31.9957	1.9442	31.9055	1.8364
1.149922	1.149809	0.600241	0.991641	30.0881	1.9448	30.0246	1.8313
1.149987	1.149987	0.600065	0.990668	29.3138	1.9816	29.5316	1.8178
1.149980	1.149945	0.600033	0.990816	29.4353	1.9719	29.5940	1.8172
1.149931	1.149802	0.600009	0.991205	29.7404	1.9560	29.7810	1.8206
1.149983	1.149997	0.600067	0.991096	29.6625	1.9601	29.7273	1.8189
1.149998	1.149923	0.600001	0.991399	29.8971	1.9489	29.8822	1.8243
1.149953	1.149931	0.600029	0.990595	29.2591	1.9878	29.5011	1.8185
1.149992	1.149889	0.600003	0.991265	29.7942	1.9530	29.8115	1.8216
1.149995	1.149972	0.600007	0.990523	29.1988	1.9939	29.4738	1.8197
1.149998	1.149596	0.600005	0.990882	29.4900	1.9688	29.6236	1.8175
1.149998	1.149616	0.600007	0.990935	29.5336	1.9663	29.6481	1.8177
1.149924	1.149660	0.600030	0.990396	29.0959	2.0047	29.4437	1.8265
1.149947	1.149897	0.600008	0.991410	29.9147	1.9473	29.8888	1.8247
1.149964	1.149855	0.600001	0.990473	29.1589	1.9986	29.4615	1.8225
1.149999	1.149855	0.600002	0.990440	29.1228	1.9991	29.4533	1.8242
1.149965	1.149855	0.600131	0.990562	29.2322	1.9904	29.4921	1.8189
1.149974	1.149864	0.600048	0.990335	29.0347	2.0083	29.4302	1.8297
1.149999	1.149979	0.600019	0.990746	29.3852	1.9765	29.5629	1.8174
1.149733	1.149857	0.600011	0.991556	30.0093	1.9468	29.9699	1.8289
1.149999	1.149979	0.600009	0.990727	29.3657	1.9786	29.5543	1.8175
1.149991	1.149660	0.600013	0.990372	29.0625	2.0062	29.4374	1.8278
1.149925	1.149857	0.600004	0.991556	30.0251	1.9463	29.9692	1.8287

Another interesting aspect visible from results is that, while Modelica validation increased mass values to $29.3 - 29.6 \text{ kg}$, simulation-based power values are lower, around $1.82 - 1.84 \text{ kW}$. This fact further reduces the optimal power peaks. It is clearly visible in figure 5.7, where the Modelica-based Pareto lies on the right-bottom side of the plot with respect to the original one, meaning that x axis (mass) values are higher and y axis (power) values are lower.

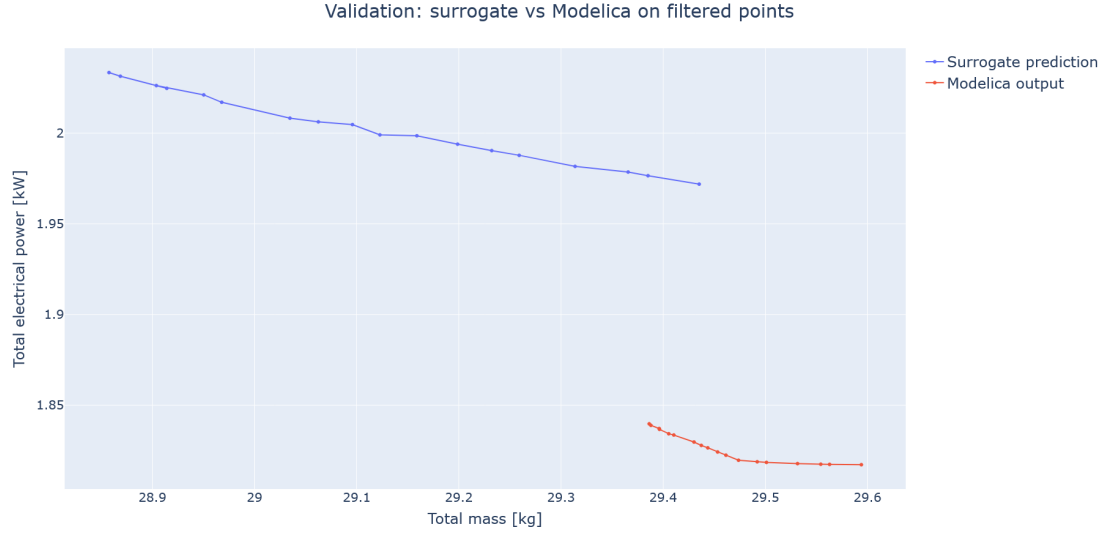


Figure 5.7: Surrogate-predicted (blue line) vs Modelica-based (red line) Pareto front. the shift to right-bottom side is clearly visible: re-simulating in Modelica gives lower peak electrical powers and higher total system masses

Chapter 6

Conclusions

The aim of this thesis is providing an integration of systems models dynamic simulations with an optimization workflow. Starting from Thermal Management Systems virtual models, the aim was to integrate them in an optimization process. In literature, this integration is mostly made using static simulations. Static simulations rely on constant heat load and boundary conditions, not revealing possible simulation hurdles like transients and cycles due to abrupt load and boundaries changes. Latter hurdles are also a matter of concern in data management: in order to perform an efficient sizing, peak values must be found, chosen and managed. In static simulations, a 100% simulation success rate is common. Once defined a feasible design space, no hidden constraint (e.g. failed simulation) shows up. On the other hand, dynamic simulations often present hidden constraints inside the design space. This problem requires a careful and dedicated management, mostly because the optimization tool produces a huge number of design points: relying on mostly failed simulations can lead to unreliable results. Another feature of the problem, the size of the design space, can become a concern. Since TMS are complex aircraft systems, optimization must be performed on a large number of design variables. This high dimensionality of the problem leads to even more hidden constraints. Surrogates are often used to partially address the problem, but a good training is essential. Therefore, a sufficient number of feasible points must be found within the design space.

RESEARCH QUESTION

How can Model-Based Design (MBD) and dynamic simulations be integrated into a Multi-Disciplinary Optimization (MDO) framework?

This thesis started by modeling the architectures in Modelica language, simulating them in time domain with time-dependant conditions and taking the outputs to perform sizing. The first DOE described an unreliable design space, filled with hidden constraints. By performing a PoV-driven DOE it has been possible to achieve a higher number of useful (feasible) simulations, enough to train surrogates and perform optimization. As last step, validation with re-simulation of optimums has been performed. Only validated design points are accepted as final results.

By using DOEs and surrogate models, with the essential use of PoV functions, it is possible to build a more reliable and smooth environment for the optimizer. Therefore, optimization results keep their reliability while strongly reducing computational cost and avoiding hidden constraints limitations.

The most important technical takeaways are:

- two-level design space exploration allowed to drastically reduce computational cost while keeping reliable results;
- PoV function significantly improved results quality and computational cost by driving the design space exploration and representing a constraint for the optimizer;
- surrogate models are a reliable and precise mathematical tool, able to predict with high fidelity the system behaviour;
- surrogate-based optimization led to significant reductions in computational time without losing accuracy;
- validation gave a 100% reliable set of optimums within a low relative error threshold.

The listed takeaways and the integration of dynamic simulations (for this kind of workflow) can lead to a more integrated design process. The workflow can be applied to any system with only small variations and, if applied to high fidelity systems models, it has been proven that it will produce high fidelity results.

6.1 Limitations

Four main limitations from the presented work are:

1. Mission profile: a standard mission profile has been used. It describes a typical flight, without accounting for possible variations like loitering, rejected landing/takeoff or engine failures. This is a possible limitation but it is partially addressed by the safety factor applied in sizing process. Further analyses should be conducted to find the least computationally expensive solution for integrating six or seven possible mission profiles without having to perform all of them for each design point, leading to a larger number of simulations.
2. Sizing: an approximated approach has been used for sizing. By using empirical and scaling laws, an approximation has been made. While this approximation can be a problem, the workflow structure is completely independent from the latter: once more reliable sizing tools will be available, they can easily be integrated into the sizing Python code by substituting the approximated laws. For the same reason, heat exchangers sizing (excluding SHX) has not been performed. Many literature studies are focusing on the topic, but none of them produced yet a simple relationship that could be used for sizing. This is because heat exchangers are the most complex components in the system. For this reason, sizing has not been performed on heat exchangers, waiting for the integration of other tools.
3. Components wear: components wearing has not been taken into account. During their life cycle, components lose some of their technical features. This is partially solved with the safety factor used in sizing process.
4. Surrogates training: despite their proven reliability, surrogates have been trained on a localized area of the design space. The area has been defined around the best combination of values found. This is a strong assumption: it's impossible to exclude that some other areas of the design space cannot provide better results.

6.2 Further Steps

Starting from general conclusions and limitations, some future steps can be drawn from this research work. Three main steps have been identified:

1. Improve optimization results: surrogate-based optimization is notoriously a difficult topic. Surrogate models introduce an approximation in the workflow. This approximation must not interfere with optimization results quality. Further studies are needed to improve the current 23% pass rate of the validation filter. Furthermore, being surrogate models a kind of interpolation, they must be resilient to optimizers' tendency to find unrealistic optimal areas. Since an optimization process aims to find areas of the design space in which the optimization functions are maximized or minimized, sometimes the optimizer finds some "weak spots" of the function that look like optimums, but in reality they are points where the surrogate model is weaker and less trained. An interesting future development would be addressing this typical problem by integrating better boundaries and definitions.
2. Control circuits optimization: as seen in Modelica models (figures 3.20, 3.21, 3.22, 3.23, 3.24), around 50% of the model itself is represented by control circuits (PID, gains and filters). This part of the model hasn't been optimized. Further studies on the topic would be interesting since it manages all the physical components behaviour and response, with a strong influence on output variables.
3. Vehicle-level integration: once the workflow has been tested and validated on Thermal Management Systems, it could be applied to every aircraft system and subsystem design. It would need more flexibility because design variables, components and objectives will be different, but as shown in this thesis the workflow is sufficiently independent from the system. It would only be necessary a re-adjustment of design variables and optimization problem definitions.

Appendix A

Python Implementations

Appendices in this section:

- **A.1 DOE Implementation**
- **A.2 Surrogate Modeling**
- **A.3 Surrogate-Based Optimization**
- **A.4 Pareto Validation**

A.1 DOE Implementation

```

1 INPUT:
2   - arch_name           : selected architecture
3   - N_L1                : number of Level-1 DOE samples
4   - N_L2                : number of Level-2 DOE samples
5   - out_dir             : base output directory
6   - vars_to_export      : Modelica variables to export
7   - q_shrink            : padding factor for focused bounds
8
9 OUTPUT:
10  - doe_L1_csv           : Level-1 DOE summary CSV path
11  - doe_L2_csv           : Level-2 DOE summary CSV path
12  - pov_model            : PoV classifier trained on L1 data
13
14 # 0. Setup
15 set_global_seed(GLOBAL_SEED)
16 dv_arch = get_design_variables_for_architecture(arch_name)
17 xlimits_global = build_xlimits(dv_arch)
18
19 # 1. LEVEL-1 DOE (global exploration in active lambda space)
20 X_L1, lambda_dicts_L1_full = generate_doe_for_architecture(
21     arch_name, N_samples=N_L1)
22
23 # 2. Run Modelica simulations for L1
24 for each lambda_dict_full in lambda_dicts_L1_full do
25     p_mods = build_modelica_params_from_lambdas(
26         lambda_dict_full, arch_name)
27     sim_out = run_simulation(arch_name, p_mods, vars_to_export)
28
29     sizing = run_component_sizing(sim_out)
30     feasible = compute_feasibility(sim_out, sizing)
31     log_row_to_L1_summary(lambda_dict_full, sim_out, sizing,
32         feasible)
33 end for
34 doe_L1_csv = save_L1_summary_csv(out_dir, arch_name)
35
36 # 3. Train PoV model on L1 data (classification: feasible vs
37     infeasible)
38 df_L1 = read_csv(doe_L1_csv)
39 X_pov = df_L1[active_lambda_columns(arch_name)]
40 y_pov = df_L1["feasible"].astype(int)
41 pov_model = train_pov_classifier(X_pov, y_pov)

```

```

37 pov_threshold = 0.7
38
39 # 4. Select feasible points and compute focused bounds for
    Level-2 DOE
40 df_feas = df_L1[df_L1["feasible"] == 1]
41 xlimits_focus, dv_arch = build_focused_xlimits_from_feasible(
42     arch_name=arch_name,
43     df_feas=df_feas,
44     q_shrink=q_shrink,
45     lambda_nominal={name: 1.0 for name in active_lambda_names(
        arch_name)}
46 )
47
48 # 5. LEVEL-2 DOE (PoV-driven sampling inside focused box)
49 X_L2 = empty()
50 lambda_dicts_L2_full = empty()
51 while len(X_L2) < N_L2 do
52     x_try = LHS_sample_one(xlimits_focus)
53     p_feas = pov_model.predict_proba(x_try)[feasible_class]
54     if p_feas >= pov_threshold:
55         X_L2.append(x_try)
56         lambda_dict_partial = vector_to_xdict(x_try, dv_arch)
57         lambda_dict_full = fill_non_active_with_ones(
            lambda_dict_partial)
58         lambda_dicts_L2_full.append(lambda_dict_full)
59     end if
60 end while
61
62 # 6. Modelica simulations for DOE L2
63 for each lambda_dict_full in lambda_dicts_L2_full do
64     p_mods = build_modelica_params_from_lambdas(
        lambda_dict_full, arch_name)
65     sim_out = run_simulation(arch_name, p_mods, vars_to_export)
66
67     sizing = run_component_sizing(sim_out)
68     feasible = compute_feasibility(sim_out, sizing)
69     log_row_to_L2_summary(lambda_dict_full, sim_out, sizing,
        feasible)
70 end for
71 doe_L2_csv = save_L2_summary_csv(out_dir, arch_name)
72
73 # 7. Outputs
74 return {doe_L1_csv, doe_L2_csv, pov_model, xlimits_global,
        xlimits_focus}

```

A.2 Surrogate Modeling

```

1 INPUT:
2     - csv_path   : DOE summary file
3     - target     : output quantity to be approximated
4
5 OUTPUT:
6     - surrogate_info : trained surrogate model and metadata
7
8 # 1. Load DOE data
9 arch_name, lambda_names, X, y = load_doe(csv_path, target=
    target)
10
11 # 2. Target transformation (if needed)
12 is_log = target in ("m_total", "P_el_total")
13     if is_log:
14         eps = 1e-6
15         y_clipped = np.clip(y, eps, None)
16         y_train_vec = np.log(y_clipped)
17     else:
18         y_train_vec = y
19
20 # 3. xlimits for surrogate validity
21 xlimits[j] = [min(X[:,j]), max(X[:,j])]
22
23 # 4. Split train/test points
24     if n < 5:
25         # Not enough points: use all for training, no test set
26         train_idx = np.arange(n)
27         test_idx = np.array([], dtype=int)
28     else:
29         perm = np.random.permutation(n)
30
31         # Use test_size fraction for test set, with basic
32         safety checks
33         test_frac = float(test_size)
34         if not (0.0 < test_frac < 1.0):
35             test_frac = 0.2 # fallback
36
37         n_test = max(int(round(test_frac * n)), 1)
38         n_test = min(n_test, n - 1) # ensure at least 1 train
39         sample
40         n_train = n - n_test

```

```
39     train_idx = perm[:n_train]
40     test_idx = perm[n_train:]
41
42     X_train, y_train = X[train_idx], y[train_idx]
43     X_test, y_test = (X[test_idx], y[test_idx]) if test_idx.
size > 0 else (None, None)
44
45 # 5. Train surrogate models
46 for model in {Kriging, RBF, LeastSquares} do
47     Train(model, X_train, y_train)
48     Evaluate RMSE, R^2 on training set
49     Evaluate RMSE, R^2 on test set (if available)
50 end for
51
52 # 6. Select best surrogate model
53 if X_test is not None and X_test.size > 0:
54     y_pred_test = sm.predict_values(X_test).ravel()
55     rmse_test = rmse(y_test, y_pred_test)
56     r2_test = r2_score(y_test, y_pred_test)
57
58 results[name] = {
59     "model": sm,
60     "rmse_train": rmse_train,
61     "r2_train": r2_train,
62     "rmse_test": rmse_test,
63     "r2_test": r2_test,
64 }
65
66 if any(not np.isnan(r["rmse_test"]) for r in results.values()):
67     best_name = min(
68         results.keys(),
69         key=lambda k: results[k]["rmse_test"]
70         if not np.isnan(results[k]["rmse_test"])
71         else float("inf"),
72     )
73 best = results[best_name]
74
75 # 7. Return surrogate metadata
76 surrogate_info = {
77     architecture, lambda_names,
78     selected_model, performance_metrics,
79     xlimits, is_log
80 }
```

A.3 Surrogate-Based Optimization

```

1 INPUT:
2     - arch_name      : architecture identifier (e.g., "Arch_3")
3     - sur_m          : mass surrogate dict
4     - sur_P          : power surrogate dict
5     - sur_pov        : PoV surrogate dict
6     - pov_min        : minimum PoV threshold (feasibility)
7     - pop_size       : NSGA-II population size
8     - n_gen          : number of generations
9     - max_evals      : max evaluations (optional)
10    - xlimits        : bounds override (optional)
11
12 OUTPUT:
13    - res            : pymoo result (Pareto set, obj, history)
14    - problem        : instantiated optimization problem
15
16 # 1. Build research box (optimizer bounds)
17 if xlimits is None:
18     dv_arch        = get_design_variables_for_architecture(
19         arch_name)
20     xlimits_base = build_xlimits(dv_arch)
21 else:
22     xlimits_base = as_array(xlimits)
23
24 dv_arch          = get_design_variables_for_architecture(
25     arch_name)
26 lambda_nominal = { dv.name : 1.0 for dv in dv_arch }
27
28 xlimits_opt = intersect_xlimits_with_builder_bounds(
29     arch_name      = arch_name,
30     xlimits        = xlimits_base,
31     lambda_nominal = lambda_nominal
32 )
33
34 # 2. Instantiate surrogate optimization problem
35 problem = TmsSurrogateProblem(
36     arch_name = arch_name,
37     sur_m     = sur_m,
38     sur_P     = sur_P,
39     sur_pov   = sur_pov,
40     pov_min   = pov_min,
41     xlimits   = xlimits_opt

```

```

40 )
41
42 # 3. Define evaluation
43 Given a batch X_all (shape: n_points * n_vars):
44     X_m   = X_all[:, idx_m]
45     X_P   = X_all[:, idx_P]
46     X_pov = X_all[:, idx_pov]
47
48     # Predict objectives from surrogates
49     m_pred = sur_m.model.predict_values(X_m)
50     P_pred = sur_P.model.predict_values(X_P)
51
52     # Log-space inverse transform (if configured)
53     if sur_m.is_log: m_pred = inv_log(m_pred, base=sur_m.
log_base)
54     if sur_P.is_log: P_pred = inv_log(P_pred, base=sur_P.
log_base)
55
56     # Predict PoV
57     pov_pred = predict_pov(sur_pov.model, X_pov)
58     pov_pred = clip(pov_pred, 0, 1)
59
60     # Objectives (minimize)
61     f0 = m_pred
62     f1 = P_pred
63
64     # Constraint (feasible if <= 0)
65     g0 = pov_min - pov_pred
66
67 # 4. Configure NSGA-II (pymoo) optimization algorithm
68 algo = NSGA2(
69     pop_size           = pop_size,
70     sampling           = FloatRandomSampling(),
71     crossover          = SBX(prob=0.9, eta=15),
72     mutation           = PM(eta=20),
73     eliminate_duplicates = True
74 )
75
76 if max_evals is not None:
77     termination = ("n_eval", max_evals)
78 else:
79     termination = ("n_gen", n_gen)
80
81 # 5. Run optimization
82 res = minimize(

```

```
83     problem      = problem,
84     algorithm     = algo,
85     termination   = termination,
86     seed          = GLOBAL_SEED,
87     save_history  = True,
88     verbose       = False
89 )
90
91 # 6. Return results
92 return res, problem
```

A.4 Pareto Validation

```

1 INPUT:
2   - val_csv           : validation CSV (validated points)
3   - lambda_names      : ordered lambda list (problem.
   lambda_names_all)
4   - thr_m, thr_P      : relative error thresholds on (m_total,
   P_el_total)
5
6 OUTPUT:
7   - filtered_set       : validated points with low surrogate
   error
8   - filtered_pareto    : Pareto-efficient subset of
   filtered_set (Modelica-based)
9
10 # 1. Load validated results
11 df_val = read_csv(val_csv)
12 idx_all = df_val["idx"]
13 X_all   = df_val[lambda_names]
14 m_sur   = df_val["m_sur"]
15 P_sur   = df_val["P_sur"]
16 m_sim   = df_val["m_sim"]
17 P_sim   = df_val["P_sim"]
18
19 # 2. Compute relative surrogate errors (Modelica as reference)
20 err_m = abs(m_sim - m_sur) / max(abs(m_sim), eps)
21 err_P = abs(P_sim - P_sur) / max(abs(P_sim), eps)
22
23 # 3. Threshold filtering (accuracy screen)
24 thr_m = 0.05
25 thr_P = 0.05
26 mask_good = (err_m < thr_m) AND (err_P < thr_P)
27
28 idx_good = idx_all[mask_good]
29 X_good   = X_all[mask_good]
30 F_good   = stack_columns(m_sim[mask_good], P_sim[mask_good])
   # simulation objectives
31
32 # 4. Pareto-clean the filtered set
33 mask_pf   = pareto_mask_from_objectives(F_good) # True if
   non-dominated
34 idx_pf    = idx_good[mask_pf]
35 X_pf      = X_good[mask_pf]

```

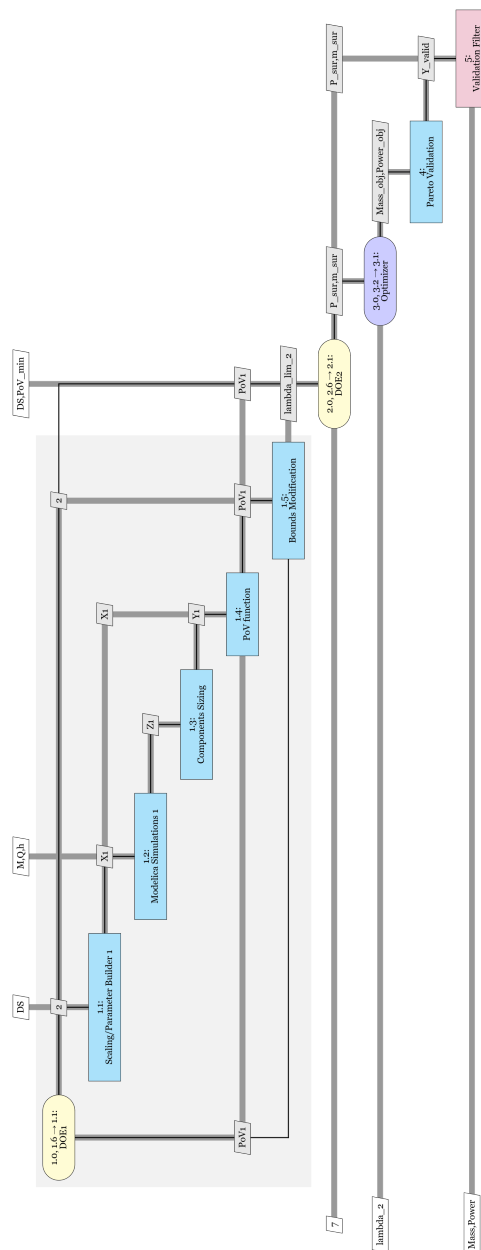
```
36 F_pf          = F_good[mask_pf]
37
38 # 5. Save/plot
39 save_csv("pareto_front_L2_filtered.csv", idx_good, X_good,
          m_sur[mask_good], P_sur[mask_good], m_sim[mask_good], P_sim
          [mask_good])
40 save_csv("pareto_front_L2_filtered_pareto_clean.csv", idx_pf,
          X_pf,  m_sur[mask_good][mask_pf], P_sur[mask_good][mask_pf]
          ],
          m_sim[
          mask_good][mask_pf], P_sim[mask_good][mask_pf])
41
42
43 plot_2d(F_good, title="Validated points with low surrogate
          error (filtered)")
44 plot_2d(F_pf,   title="Pareto front after filtering (Modelica-
          based, pareto-clean)")
```

Appendix B

Workflow Detailed Diagrams

Appendices in this section:

- **B.1 DOE-L1 Focused Diagram**
- **B.2 DOE-L2 Focused Diagram**
- **B.3 MDO Focused Diagram**



136

Bibliography

- [1] International Energy Agency (IEA). CO2 emissions in aviation in the Net Zero Scenario, 2000-2030, 2025. IEA, Paris. Licence: CC BY 4.0. Accessed October 2025. <https://www.iea.org/data-and-statistics/charts/co2-emissions-in-aviation-in-the-net-zero-scenario-2000-2030>.
- [2] International Energy Agency (IEA). Global total commercial air passenger traffic, 2019-2023, 2025. IEA, Paris. Licence: CC BY 4.0. <https://www.iea.org/data-and-statistics/charts/global-total-commercial-air-passenger-traffic-2019-2023-2>.
- [3] Clean Aviation Joint Undertaking. Clean Aviation: We innovate for the future of aviation. <https://www.clean-aviation.eu/>, 2026. Accessed: 2026-02-04.
- [4] ODE4HERA Project. Open Digital Environment for Hybrid-Electric Regional Aircraft (ODE4HERA), 2025. Accessed October 2025. <https://www.ode4hera.eu/>.
- [5] Maria Maia Coutinho, Frederico Afonso, Alain Souza, David Bento, Ricardo Gandolfi, Felipe R. Barbosa, Fernando Lau, and Afzal Suleman. A Study on Thermal Management Systems for Hybrid-Electric Aircraft. *Aerospace*, 10(9):745, 2023.
- [6] Manuel A. Rendón, Carlos D. Sánchez R., Josselyn Gallo M., and Alexandre H. Anzai. Aircraft Hybrid-Electric Propulsion: Development Trends, Challenges and Opportunities. *Journal of Control, Automation and Electrical Systems*, 32:1244–1268, 2021.
- [7] Dirk Zimmer. Modelica Workshop. Slides from Workshop on Modelica language. DLR Institute of Robotics and Mechatronics. 2024-10.
- [8] Dirk Zimmer, Michael Meißner, and Niels Weber. The DLR ThermoFluid Stream Library. *Electronics*, 11(22), 2022. Open access, CC BY 4.0.

- [9] M. Rutheravan. Summary of Safety Criteria in Design. Technical report, Automotive Engineering Research Group (AERG), Faculty of Mechanical Engineering, Universiti Malaysia Pahang, Pekan, Pahang, Malaysia, 2016. Overview of factor of safety definitions and typical recommended ranges for engineering applications.
- [10] Joaquim R. R. A. Martins and Andrew Ning. *Engineering Design Optimization*. Cambridge University Press, Cambridge, UK, January 2022.
- [11] Maria Maia Coutinho. On the Design of Thermal Management Systems for Hybrid-Electric Aircraft. Master’s thesis, Universidade de Lisboa, December 2022. Master of Science in Aerospace Engineering.
- [12] International Civil Aviation Organization. *Manual of the ICAO Standard Atmosphere: Extended to 80 Kilometres (262 500 Feet)*. Doc 7488/3. International Civil Aviation Organization, Montreal, third edition, 1993. Also published in French and Spanish as *Manuel de l’Atmosphère Type OACI* and *Manual de la Atmósfera Tipo de la OACI*.
- [13] Pierburg GmbH. *CWA400 Electric Water Pump (PWM Version)*. Pierburg GmbH, n.d. Brushless electric water pump; specifications and PWM control characteristics.
- [14] Parker Hannifin Corporation. *28V DC Brushless Fuel Pump Product Family*. Parker Aerospace, Fluid Systems Division, Elyria, Ohio, USA, n.d. Model families 11C12/11C13, 13C1, 14C1.
- [15] ebm-papst Mulfingen GmbH & Co. KG. *EC Centrifugal Module RadiCal, Backward-Curved, with Housing (Type K3G175-RC05-03)*. ebm-papst Mulfingen GmbH & Co. KG, Mulfingen, Germany, November 2018. Item No. 53479-5-9980; EC centrifugal fan with integrated electronics; air performance curves and electrical characteristics.
- [16] BITZER Kühlmaschinenbau GmbH. *Hermetic Scroll Compressors ORBIT, ORBIT+ and ORBIT FIT*. BITZER Kühlmaschinenbau GmbH, Sindelfingen, Germany, September 2019. Technical data, performance maps and application limits for hermetic scroll compressors; suitable for vapour compression systems.
- [17] Relly Victoria Virgil Petrescu, Raffaella Aversa, Bilal Akash, Ronald Bucinell, Antonio Apicella, and Florian Ion Tiberiu Petrescu. History of Aviation - A Short Review. *Journal of Aircraft and Spacecraft Technology*, 1(1):30–49, 2017. Open access (CC BY 3.0).

- [18] International Energy Agency (IEA). Aviation - Tracking, 2025. Accessed October, 2025. <https://www.iea.org/energy-system/transport/aviation#tracking>.
- [19] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.
- [20] International Air Transport Association. Aircraft Technology: Net Zero Roadmap. Technical report, IATA, Montreal, Canada, 2023. Complementary to the Waypoint 2050 and Fueling Net Zero analyses.
- [21] Benjamin J. Brelje and Joaquim R. R. A. Martins. Electric, Hybrid, and Turboelectric Fixed-Wing Aircraft: A Review of Concepts, Models, and Design Approaches. *Progress in Aerospace Sciences*, 104:1–19, 2019.
- [22] C. Pornet and A. T. Isikveren. Conceptual design of hybrid-electric transport aircraft. *Progress in Aerospace Sciences*, 79:114–135, 2015.
- [23] Amir S. Gohardani, Geoffrey Doulgeris, and R. Mark Merrett. A Review of Concepts, Benefits, and Challenges for Aircraft Propulsion by Electric Power. *Progress in Aerospace Sciences*, 57:84–111, 2013.
- [24] Erik J. Stalcup, Timothy P. Dever, and Jonah J. Sachs-Wetstone. Thermal Management Key Performance Parameter Development and System Analysis for the SUSAN Electrofan Aircraft. Research report, NASA Glenn Research Center, 2025. Key performance parameters (KPP) for SUSAN Electrofan TMS; sizing & mission analysis.
- [25] Jaap van Muijden and Vincent J. E. Aalbers. Thermal management principal analysis of hybrid-electric aircraft with turboelectric propulsion using distributed propulsors. In *9th European Conference for Aeronautics and Space Sciences (EUCASS)*, 2022. Presented within the IMOTHEP project (Investigation and Maturation of Technologies for Hybrid-Electric Propulsion).
- [26] Soheil Jafari and Theoklis Nikolaidis. Thermal Management Systems for Civil Aircraft Engines: Review, Challenges and Exploring the Future. *Applied Sciences*, 8(11):2044, 2018.
- [27] John A. Harris, Michael A. Santi, Richard P. Shively, and Matthew P. Dwyer. Aircraft Thermal Management: Practices, Technology, System Architectures, and Future Challenges. *Progress in Aerospace Sciences*, 127:100779, 2021.

- [28] Walter Affonso Jr., Ricardo Gandolfi, Ricardo Jose Nunes dos Reis, Carlos Roberto Ilário da Silva, Nicolas Rodio, Timoleon Kipouros, Panagiotis Laskaridis, Andrei Chekin, Yury Ravikovich, Nikolay Ivanov, Leonid Ponyaev, and Dmitry Holobtsev. Thermal Management Challenges for Hybrid-Electric Aircraft - FUTPRINT 50. In *IOP Conference Series: Materials Science and Engineering*, volume 1024, page 012075. 10th EASN Conference, IOP Publishing, 2021.
- [29] Yu-Feng Mao, Yun-Ze Li, Ji-Xiang Wang, Kai Xiong, and Jia-Xin Li. Cooling Ability/Capacity and Exergy Penalty Analysis of Each Heat Sink of Modern Supersonic Aircraft. *Entropy*, 21(3):223, February 2019.
- [30] Hagen Kellermann, Michael Lüdemann, Markus Pohl, and Mirko Hornung. Design and Optimization of Ram Air-Based Thermal Management Systems for Hybrid-Electric Aircraft. *Aerospace*, 8(1):3, December 2020.
- [31] Iara Figueiras, Maria Coutinho, Frederico Afonso, and Afzal Suleman. On the Study of Thermal-Propulsive Systems for Regional Aircraft. *Aerospace*, 10(2):113, January 2023.
- [32] Hagen Kellermann, Ana Luisa Habermann, and Mirko Hornung. Assessment of Aircraft Surface Heat Exchanger Potential. *Aerospace*, 7(1):1–19, 2020.
- [33] Anaïs Luisa Habermann, Ankit Khot, David Emanuel Lampl, and Christof Perren. Aerodynamic Effects of a Wing Surface Heat Exchanger. *Aerospace*, 10(5):407, April 2023.
- [34] Sergio Chiesa. *Impianti di bordo per aeromobili: Impianto idraulico*. CLUT, Torino, 1988. Technical handbook about aircraft hydraulic system.
- [35] Sergio Chiesa. *Impianti di bordo per aeromobili: Impianto combustibile*. CLUT, Torino, 1988. Technical handbook about aircraft fuel system.
- [36] S. L. Dixon and C. A. Hall. *Fluid Mechanics and Thermodynamics of Turbomachinery*. Butterworth-Heinemann, Oxford, 6th edition, 2014. Sixth Edition.
- [37] Georgiana-Claudia Negoita, Jean-Charles Mare, Marc Budinger, and Nicolae Vasiliu. Scaling-laws Based Hydraulic Pumps Parameter Estimation. *U.P.B. Scientific Bulletin, Series D: Mechanical Engineering*, 74(2):199–208, 2012. Study on similarity laws applied to axial piston pumps from Rexroth and Parker data.
- [38] National Aeronautics and Space Administration. NASA Systems Engineering Handbook. Technical Report NASA/SP-2016-6105 Rev. 2, NASA Office of

- the Chief Engineer, Washington, DC, 2016. Supersedes NASA/SP-2007-6105 Rev. 1.
- [39] National Aeronautics and Space Administration. NASA Space Flight Program and Project Management Requirements. Technical Report NPR 7120.5F, NASA Office of the Chief Engineer, Washington, DC, August 2021. Effective Date: August 3, 2021; Expiration Date: August 3, 2026.
- [40] European Space Agency. Margin Philosophy for Science Assessment Studies. Technical Note SRE-PA/2011.097, ESA-ESTEC, Noordwijk, The Netherlands, June 2012. Issue 1, Revision 3.
- [41] Daniel P. Raymer. *Aircraft Design: A Conceptual Approach*. AIAA Education Series. American Institute of Aeronautics and Astronautics, Washington, DC, second edition, 1992.
- [42] Joaquim R. R. A. Martins and Andrew B. Lambe. Multidisciplinary Design Optimization: A Survey of Architectures. *AIAA Journal*, 51(9):2049–2075, 2013.
- [43] Timothy W. Simpson, Dennis K. J. Lin, and Wei Chen. Sampling Strategies for Computer Experiments: Design and Analysis. *International Journal of Reliability and Applications*, 2(3):209–240, 2001.
- [44] M. D. McKay, R. J. Beckman, and W. J. Conover. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, 21(2):239–245, 1979.
- [45] Jasper H. Bussemaker, Paul Saves, Nathalie Bartoli, Thierry Lefebvre, and Björn Nagel. Surrogate-Based Optimization of System Architectures Subject to Hidden Constraints. *arXiv preprint*, 2025.
- [46] Sébastien Defoort, Mathieu Balesdent, Philippe Klotz, Pascal Schmollgruber, Jérôme Morio, Jérôme Hermetz, Cédric Blondeau, G. Carrier, and Nicolas Bérend. Multidisciplinary Aerospace System Design: Principles, Issues and Onera Experience. *AerospaceLab*, 1(4), may 2012.
- [47] Jasper H. Bussemaker. SBArchOpt: Surrogate-Based Architecture Optimization. *Journal of Open Source Software*, 8(89):5564, 2023.
- [48] Nestor V. Queipo, Raphael T. Haftka, Wei Shyy, Tushar Goel, Raj Vaidyanathan, and P. Kevin Tucker. Surrogate-based Analysis and Optimization. *Progress in Aerospace Sciences*, 41(1):1–28, 2005.

- [49] Sparsh Garg, Raúl García Sánchez, Jasper H. Bussemaker, Luca Boggero, and Björn Nagel. Dynamic Formulation and Execution of MDAO Workflows for Architecture Optimization. In *AIAA AVIATION 2024 Forum*. American Institute of Aeronautics and Astronautics, 2024.
- [50] Mohammad Reza Setayandeh. Surrogate Model-Based Robust Multidisciplinary Design Optimization of an Unmanned Aerial Vehicle. *Journal of Aerospace Engineering*, 34(4):04021029, 2021.
- [51] Modelica Association. Modelica Language, 2025. Accessed October 2025. <https://modelica.org/language/>.
- [52] Modelica Association. *Modelica Language Specification, Version 3.6*. Modelica Association, 2023. Official language specification.
- [53] Peter Fritzson, Adrian Pop, Karim Abdelhak, Adeel Ashgar, Bernhard Bachmann, Willi Braun, Daniel Bouskela, Robert Braun, Lena Buffoni, Francesco Casella, Rodrigo Castro, Rüdiger Franke, Dag Fritzson, Mahder Gebremedhin, Andreas Heuermann, Bernt Lie, Alachew Mengist, Lars Mikelsons, Kannan Moudgalya, Lennart Ochel, Arunkumar Palanisamy, Vitalij Ruge, Wladimir Schamai, Martin Sjölund, Bernhard Thiele, John Tinnerholm, and Per Östlund. The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development. *Modeling, Identification and Control*, 41(4):241–295, 2020.
- [54] Dirk Zimmer, Michael Meißner, and Niels Weber. The DLR Thermofluid Stream Library, July 2021. <https://github.com/DLR-SR/ThermofluidStream>.
- [55] Mokon. *Heat Transfer Fluid - Ethylene Glycol: Technical & Engineering Data*. Mokon Thermal Fluid Systems, 2150 Elmwood Ave., Buffalo, NY 14207, USA, n.d. Technical data sheet for Mokon Ethylene Glycol heat transfer fluid.
- [56] Frank P. Incropera, David P. DeWitt, Theodore L. Bergman, and Adrienne S. Lavine. *Fundamentals of Heat and Mass Transfer*. John Wiley & Sons, 6 edition, 2007.
- [57] Reiner Tillner-Roth and Hans Dieter Baehr. An International Standard Formulation for the Thermodynamic Properties of 1,1,1,2-Tetrafluoroethane (HFC-134a) for Temperatures from 170 K to 455 K and Pressures up to 70 MPa. *Journal of Physical and Chemical Reference Data*, 23(5):657–729, 1994.
- [58] Infraser v GmbH & Co. Höchst KG. R-134a (1,1,1,2-Tetrafluoroethane) - Refrigerant Data Sheet, 2024. Accessed October 2025. <https://>

[//www.infraserv.com/en/services/facility-management/expertise/f-gas/refrigerant/specific-refrigerant/r-134a.html](http://www.infraserv.com/en/services/facility-management/expertise/f-gas/refrigerant/specific-refrigerant/r-134a.html).

- [59] European Parliament and Council of the European Union. Regulation (EU) No 517/2014 of the European Parliament and of the Council of 16 April 2014 on Fluorinated Greenhouse Gases and Repealing Regulation (EC) No 842/2006, 2014. Official Journal of the European Union, L 150, 20 May 2014, pp. 195–230. Accessed October 2025.
- [60] ServiceTitan. What is Superheat in Refrigeration? How to Measure, May 2025. Accessed November 2025. <https://www.servicetitan.com/blog/what-is-superheat>.
- [61] ATR - Avions de Transport Régional. *ATR 72-600 Factsheet*, 2019. Manufacturer data sheet. Engines: Pratt & Whitney Canada PW127M/N; Propellers: Hamilton Standard 568F.
- [62] Clifford Moses. Review of Bulk Physical Properties of Synthesized Hydrocarbon: Kerosenes and Blends. Interim Report AFRL-RQ-WP-TR-2017-0091, Air Force Research Laboratory, Aerospace Systems Directorate, June 2017. Delivery Order 0006: Airbreathing Propulsion Fuels and Energy Exploratory Research and Development (APFEERD), Subtask: Review of Bulk Physical Properties of Synthesized Hydrocarbon: Kerosenes and Blends. Approved for public release (88ABW-2017-2895).
- [63] Plotly Technologies Inc. Collaborative data science, 2015.
- [64] SKYbrary Aviation Safety. International Standard Atmosphere (ISA), 2026. Accessed: 2026-01-14.
- [65] Goodfellow. Dural[®] Aluminum / Copper / Magnesium Alloy. Material datasheet, n.d. High-strength aluminum alloy datasheet; density, mechanical, thermal, and electrical properties.
- [66] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.
- [67] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [68] P. Saves, R. Lafage, N. Bartoli, Y. Diouane, J. Bussemaker, T. Lefebvre, J. T. Hwang, J. Morlier, and J. R. R. A. Martins. SMT 2.0: A Surrogate Modeling

- Toolbox with a focus on Hierarchical and Mixed Variables Gaussian Processes. *Advances in Engineering Software*, 188:103571, 2024.
- [69] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 11 2005.
- [70] Robert Schaback. Multivariate Interpolation by Polynomials and Radial Basis Functions. *Constructive Approximation*, 18(3):349–366, 2002.
- [71] Ben Adcock. Optimal Sampling for Least-Squares Approximation. *arXiv preprint*, 2025.
- [72] Davide Chicco, Matthijs J. Warrens, and Giuseppe Jurman. The coefficient of determination R^2 is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis. *PeerJ Computer Science*, 7:e623, 2021.
- [73] J. Blank and K. Deb. pymoo: Multi-Objective Optimization in Python. *IEEE Access*, 8:89497–89509, 2020.
- [74] Joaquim R. R. A. Martins and Andrew B. Lambe. Multidisciplinary Design Optimization: A Survey of Architectures. *AIAA Journal*, 51(9):2049–2075, 2013.