

TECHNISCHE UNIVERSITÄT MÜNCHEN

TUM SCHOOL OF ENGINEERING AND DESIGN

Aircraft Design Optimization Informed by Stakeholder Wisdom

Thomas Skyler Sartorius

Vollständiger Abdruck der von der TUM School of Engineering and Design der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

genehmigten Dissertation.

Vorsitzender: Prof. Dr.-Ing. Markus Lienkamp

Prüfer der Dissertation: 1. Prof. Dr.-Ing. Mirko Hornung

2. Prof. Dr.-Ing. Manfred Hajek

Die Dissertation wurde am 22.09.2020 bei der Technischen Universität München eingereicht und durch die TUM School of Engineering and Design am 21.04.2021 angenommen.

Abstract

There are several aspects of the iterative design process that present opportunities for innovation in the development of complex systems such as aircraft. The focus of the work presented here lies on leveraging un- or under-captured information for the purposes of improving design process outcomes. The three main categories of information relate to assumptions, stakeholder preferences, and system modeling. The method developed for this work not only facilitates formally capturing this information, but also putting it to use through automatic design space exploration employing an application of numerical optimization algorithms.

The method is implemented in a software framework centering on the analytical model of the system in question, a design variable class to capture information, and a design space class to execute the searching and facilitate design exploration and understanding. The searching results are a finite set of design points likely to be of interest for further design iterations, along with other artifacts to aid in gaining design insights and support decision-making. Example cases presented include the prospective clean-sheet design of a small regional airliner and another study examining a potential wing redesign with a winglet for a narrow-body airliner. Several benefits were observed related to facilitating effective design space exploration with reduced resources and reaping benefits from utilizing optimization much earlier in the design and development process than is otherwise normally practical.

Kurzfassung

Bei der Entwicklung von komplexen Systemen, wie zum Beispiel Flugzeugen, gibt es verschiedene Aspekte des iterativen Entwurfsprozesses, die Innovationsmöglichkeiten bieten. Der Fokus des hier präsentierten Forschungsprojektes liegt darin, nicht bzw. nicht ausreichend erfasste Informationen zu nutzen, um die Ergebnisse des Entwurfsprozess zu verbessern. Die drei Hauptkategorien von Informationen beziehen sich auf Annahmen, Stakeholderpräferenzen und Systemmodellierung. Die entwickelte Methode dieser Arbeit erleichtert nicht nur die formale Erfassung dieser Informationen, sondern auch deren Nutzung durch die automatisierte Untersuchung des Entwurfsraumes und die Verwendung von numerischen Optimierungsalgorithmen.

Die Methode ist in einer Softwareumgebung implementiert. Die Umgebung basiert sich auf dem analytischen Modell des betreffenden Systems, einer Objektklasse von Entwurfsvariablen zur Erfassung der Informationen und einer Designraumobjektklasse für die Durchführung der Optimierung und ermöglichen einer Parametervariation und dem Verständnis der Entwurfsergebnisse. Die Optimierung ergibt eine Menge an Designpunkten, die mit großer Wahrscheinlichkeit interessante Eigenschaften für weitere Entwurfsiterationen darstellen, sowie weitere Hilfestellungen zur Gewinnung von Einblicken in den Entwurf und zur Unterstützung von Designentscheidungen. Als Beispiel werden folgende zwei Fallstudien durchgeführt: Der Neuentwurf eines kleinen Regionalverkehrsflugzeugs und die Neugestaltung eines Flügels mit Winglet für ein narrow-body Passagierflugzeug. Hierbei zeigt die Methode Vorteile gegenüber dem Stand der Technik durch eine effektivere Entwurfsraumuntersuchung mit reduzierten Ressourcen sowie durch die frühere Anwendung von Optimierung im Verlauf des Entwurfsprozesses.

Contents

Abstract	i
Kurzfassung	ii
Contents	iii
List of Figures	vi
List of Tables	viii
Nomenclature	ix
Abbreviations and acronyms	ix
Symbols	x
1 Introduction	1
1.1 Design research domains	2
1.2 Design processes	4
1.2.1 Archetypal aircraft early design iteration process	4
1.2.2 Opportunities in the design iteration process	5
1.3 Motivation for exploring a different approach	7
1.3.1 Software-centric contemporary workflows	7
1.3.2 Un- and under-captured stakeholder wisdom	7
1.3.3 Increasing chances of product success	9
1.3.4 Focus on early design	10
1.4 Objectives	12
1.4.1 Objective 1: Facilitate formally capturing stakeholder wisdom	12
1.4.2 Objective 2: Automate the integration of captured information in design space exploration and decisions	12
1.5 Structure of this work	14
2 State of the Art	15
2.1 Requirements elicitation and analysis	16
2.1.1 Market research	16
2.1.2 Traditional aerospace and defense approach to requirements	16
2.1.3 Requirements traceability for verification and validation	18
2.2 Design exploration and decision techniques	20
2.2.1 Trade studies	20
2.2.2 House of quality	21
2.2.3 Design space understanding and visualization techniques	22
2.3 Accounting for uncertainty	28
2.4 Optimization	29
2.4.1 Single-objective search	29
2.4.2 Multi-objective search	31
2.4.3 Physical programming	33
2.5 Comparing optimization to other search techniques	36
3 Methodology	38
3.1 Applicability of approach	39
3.1.1 Early design studies	39
3.1.2 Moderate dimensionality	39
3.1.3 Mostly continuous design variables	40
3.1.4 Nominally convex design space	40
3.1.5 Tightly coupled parameters	41
3.2 Issues with optimization in early design	42
3.2.1 Imperfectly defined requirements and objectives	42
3.2.2 Low-fidelity analytical models	42

3.2.3	Immature system models.....	42
3.2.4	Rigidity of optimization approach.....	42
3.2.5	Algorithms focused on final solution	43
3.3	Types of information captured.....	44
3.3.1	Example regional airliner design for illustration.....	44
3.3.2	Assumption uncertainty	45
3.3.3	Preferences on figures of merit and design parameters	49
3.3.4	Known uncaptured system model behaviors	54
3.4	Application of optimization.....	58
3.4.1	Objective function	58
3.4.2	Introducing variation	58
3.4.3	Constraints.....	60
3.5	Understanding and decision-making	62
3.5.1	Nature of optimization results	62
3.5.2	Processing of optimization results	62
3.5.3	Iteration actions	63
4	Implementation.....	64
4.1	Overview	65
4.1.1	Implementation priorities and requirements.....	65
4.1.2	Nominal workflow.....	67
4.1.3	MATLAB as selected programming language.....	69
4.1.4	Framework components and organization	70
4.2	System model function	72
4.2.1	Syntax.....	72
4.2.2	Input and output variable types.....	74
4.2.3	Typical evolution of system model function.....	74
4.3	Design variable class	76
4.3.1	Descriptive and system model interfacing attributes.....	76
4.3.2	Attributes capturing preferences and other information	78
4.3.3	Design variable class methods	81
4.4	Design space class	85
4.4.1	User-facing design space attributes.....	85
4.4.2	Design space utility methods and attributes.....	86
4.5	WISDOM approach implementation	90
4.5.1	Batch analysis for sweeps and Monte Carlo analysis	90
4.5.2	WISDOM searching and optimization	90
4.5.3	Methods for understanding WISDOM results.....	92
5	Example Cases	95
5.1	Regional airliner	96
5.1.1	System model.....	96
5.1.2	Design variable definitions.....	97
5.1.3	Searching with WISDOM approach	98
5.1.4	Comparison to value function methods.....	101
5.2	Wing redesign with winglet.....	103
5.2.1	System model.....	103
5.2.2	Design variable definitions.....	105
5.2.3	Searching results	108
6	Conclusion	111
6.1	Expected benefits.....	112
6.1.1	Allocation of resources	112

6.1.2	Facilitating effective design iteration	112
6.1.3	Early optimization	113
6.2	Pitfalls and drawbacks.....	115
6.3	Future potential	117
6.3.1	Improvements.....	117
6.3.2	New capabilities.....	118
	References	121
	Appendix A: WISE Documentation	126
A1	README	126
A2	<code>DesignVariable</code> help block	127
A3	Select <code>DesignSpace</code> help documentation.....	129
	Appendix B: Example Case Source Code	134
B1	Regional airliner system model function.....	134
B2	Regional airliner setup script	136
B3	Wing redesign with winglet system model function	138
B4	Wing redesign with winglet setup script.....	142

List of Figures

Figure 1-1. TRL scale summary.	3
Figure 1-2. Simple depiction of “V” model of development.	4
Figure 1-3. Typical design iteration process.	5
Figure 1-4. Illustration of early payload capacity trade study.	8
Figure 1-5. A disproportionate amount of definition is set in early design.	10
Figure 2-1. DoD systems engineering process.	17
Figure 2-2. NASA systems engineering process.	18
Figure 2-3. Example stoplight chart decision matrix for an electric aircraft propulsion-empennage configuration.	20
Figure 2-4. Tiered approach to decision matrix criteria prioritization and weighting.	21
Figure 2-5. House of quality for a multirole jet fighter.	22
Figure 2-6. Example simple two-parameter plot for a solid-core UAV wing.	23
Figure 2-7. Example sensitivity analysis for a HALE ISR UAV.	23
Figure 2-8. Example constraint diagram for airplane conceptual wing and powerplant sizing visualization.	24
Figure 2-9. Example carpet plot with single figure of merit on the ordinate axis.	25
Figure 2-10. Constraint matrix for conceptual helicopter sizing.	26
Figure 2-11. Pareto set visualizer with data brushing for design space exploration.	27
Figure 2-12. Local search iteratively following steepest descent to local minimum near starting point.	30
Figure 2-13. Various roles of multi-objective methods.	31
Figure 2-14. Physical programming preference classification.	33
Figure 2-15. Physical programming class function ranges.	34
Figure 2-16. Linear physical programming class function.	35
Figure 3-1. Example of 'micro' non-convexity typical of iterative methods used in early aircraft design.	41
Figure 3-2. Regional jet transport empty weight fraction trend.	45
Figure 3-3. Assumptions set to fixed values early in analysis.	45
Figure 3-4. Triangular distributions for uncertain technical assumptions.	46
Figure 3-5. Resulting gross weight estimate distribution, propagating uncertainty.	47
Figure 3-6. PERT distributions for uncertain technical assumptions.	48
Figure 3-7. Resulting distribution when using PERT-distributed assumptions instead of triangular.	49
Figure 3-8. Regional jet gross weight versus passenger capacity.	51
Figure 3-9. Basic preference map of passenger capacity.	51
Figure 3-10. Clustering of design results driven by preferences captured in a preference map.	53
Figure 3-11. Linear preference map for gross weight, the surrogate objective to minimize.	55
Figure 3-12. Preference map for capturing known system model characteristics.	55
Figure 3-13. Gross weight as a function of aerodynamic efficiency.	56
Figure 3-14. Extents of the effect of L/D_{max} preference map on qualitative nature of system modeling behavior.	57

Figure 3-15. Flowchart of objective function integrating preference maps.	60
Figure 3-16. Flowchart of objective function with constraints.	61
Figure 4-1. Nominal workflow for WISE framework.	68
Figure 4-2. Top-level organization of the WISE framework setup of a design. .	71
Figure 4-3. Traditional MDO typical required syntax for objective function.....	72
Figure 4-4. Typical MDO required syntax for defining nonlinear constraints. ...	73
Figure 4-5. System model function syntax.....	73
Figure 4-6. System model function syntax and pseudocode with additional typical features.	75
Figure 4-7. Illustration of various possible PDFs based on <code>distribution</code> attribute.	79
Figure 4-8. Screenshot of design variable class <code>plot</code> method.	82
Figure 4-9. GUI window for the design variable <code>editmap</code> method.....	83
Figure 4-10. Design variable editing using the built-in variable editor.	84
Figure 4-11. Example code output from design variable <code>export</code> method.....	84
Figure 4-12. Name-based command-line design variable referencing in <code>DesignSpace</code> object <code>o</code>	85
Figure 4-13. Tabular display of <code>DesignSpace</code> objects or arrays of <code>DesignVariable</code> objects.....	86
Figure 4-14. The design space class <code>toggle</code> method.....	88
Figure 4-15. Routine used by <code>run</code> method for evaluating design points.	89
Figure 4-16. Operations used by <code>searching</code> method to generate a variety of optimized designs.....	91
Figure 4-17. Method for more in-depth analysis of a design point.....	94
Figure 5-1. Simple mission profile for regional airliner system model.....	96
Figure 5-2. Regional airliner design variable plots.....	98
Figure 5-3. Regional airliner results visualization with clustering.....	99
Figure 5-4. Regional airliner results visualization with data brushing.	101
Figure 5-5. Wing of Boeing 737-700 narrow-body airliner with winglet.....	103
Figure 5-6. Narrow-body airliner baseline half-wing modeled in AVL.	104
Figure 5-7. Wing redesign design variables.	105
Figure 5-8. Preference maps for root bending moment and winglet structural aspect ratio used as surrogates for structural modeling.	106
Figure 5-9. Preferences driving toward favorable structural alignment and load paths.	107
Figure 5-10. Capturing binary possibilities of gross weight increase or useful load decrease.....	107
Figure 5-11. Preferences on winglet and total span.	108
Figure 5-12. Wing redesign results visualization with clustering.....	109
Figure 5-13. Representative alternative wing geometries.....	110
Figure 6-1. Granular preference map for payload capacity of a personal-use homebuilt aircraft.....	116

List of Tables

Table 1-1. Typical categories of un- and under-captured information.	8
Table 2-1. Market research techniques overview.	16
Table 2-2. Attributes of alternative approaches to design space search.	37
Table 3-1. The basic six design variables of airplane conceptual design.	39
Table 3-2. Example design case range and payload requirements.	44
Table 4-1. Comparison of <code>DesignVariable</code> and <code>DesignSpace</code> classes.	85
Table 5-1. Mission profile parameters and assumptions.	97
Table 5-2. Table of design points representative of regional airliner clusters.	100
Table 5-3. Regional airliner design points from weighted sum method.	102
Table 5-4. Wing redesign study modeling summary.	104
Table 5-5. Data for representative alternative designs.	110

Nomenclature

Abbreviations and acronyms

14 CFR	Code of Federal Regulations Title 14
ADDAM	Aircraft Design DATA Model
AVL	Athena Vortex Lattice
CORE	Conceptual Optimization of Rotorcraft Environment
CPACS	Common Parametric Aircraft Configuration Scheme
DAL	Development assurance level
DO	Document
DoD	US Department of Defense
DUEL	Design Understanding and Exploration Library
ELECTRE	<i>Elimination Et Choix Traduisant la REalité</i> (Elimination and Choice Expressing Reality)
FAA	US Federal Aviation Administration
FOM	Figure of merit
GUI	Graphical user interface
HALE	High altitude, long endurance
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise
IDE	Integrated development environment
IFR	Instrument flight rules
ISR	Intelligence, surveillance, reconnaissance
KTAS	Knots true airspeed
MDO	Multidisciplinary design optimization
NASA	US National Aeronautics and Space Administration
OEI	One engine inoperative
OEW	Operational empty weight
PDF	Probability density function
PERT	Program Evaluation and Review Technique
SFC	Specific fuel consumption
TOGW	Takeoff gross weight
TRL	Technology readiness level
UAV	Unmanned aerial vehicle
V&V	Verification and validation
WISDOM	Well-Informed Search Design Optimization Method
WISE	Well-Informed Search Environment

Symbols

AR	Aspect ratio
b	Span
D	Drag
E	Endurance
f	Objective function
g	Inequality constraint function
h	Equality constraint function
L	Lift
LB	Lower bound
P	Power
p	Preference mapping function of parameter
R	Range
S	System model function
S	Wing planform area
T	Thrust
UB	Upper bound
V	Speed
W	Weight
W_0	Gross weight
x	Design variables
X	Random number
x^*	Design variable values at optimum
x_0	Design variable values at search starting point
α	Beta distribution shape parameter
β	Beta distribution shape parameter
λ	PERT distribution shape parameter, taper ratio
ρ	Uncertainty scaling factor
ω	Stochastic assumption parameters

1 Introduction

The aim of any research endeavor is to further the state of the art or add to the general body of knowledge. The body of work described in this document focuses on improving process outcomes in the design and development of aerospace products, with a focus on early and conceptual aircraft design. In this chapter, a discussion of the domain of development processes, a subset of the broader aerospace research categories, lends context to the motivation for the specific topic, namely exploring a new technique to leverage a software-based process to achieve improved design process outcomes.

1.1 Design research domains

When discussing research in the field of aeronautics, there are several high-level domains of exploration. One such domain is pure scientific discovery, a focus that is more closely related to the natural sciences than the engineering that aeronautics is normally associated with. This type of search for pure knowledge is characterized by discovery and seeking a better understanding of phenomena in the physical world as it is, as opposed to the creation of something new.

Another domain of research in aeronautics is that of creation, evaluation, and refinement of domain-specific analyses. These are methods of prediction and are often associated with the classic disciplines such as structures and materials, aerodynamics, stability and control, propulsion, etc. Improving prediction methods increases the likelihood that after the heavy investment of time and resources into developing an aerospace product, the product will, when the program is at a stage of verification and validation, meet requirements and expectations.

The creation and evaluation of new and novel technologies and concepts is yet another distinct domain of research within aerospace. When it is not yet certain if or under what circumstances a novel invention, configuration, or other concept is feasible or viable, research in this domain aims to mature the technology, increasing the so-called technology readiness level. The technology readiness level, or TRL, is a measure of maturity commonly used by aerospace and defense organizations, particularly government organizations. The TRL scale ranges from 1 (just an idea based on observed scientific principles, transitioning from scientific to applied research) to 9 (a proven and deployed product). The definitions of the various stages of TRL are summarized in Figure 1-1, and these definitions are similar for organizations such as the US Department of Defense (Defense Acquisition University Press, 2001), NASA (NASA, n.d.), ESA (ESA, n.d.), and ISO (ISO/TC 20/SC 14, 2013). The TRL scale correlates to research and development activities within an organization whereby investigations of low-TRL technologies and ideas are typically categorized as research, and the work on higher-TRL concepts is called development or product development.

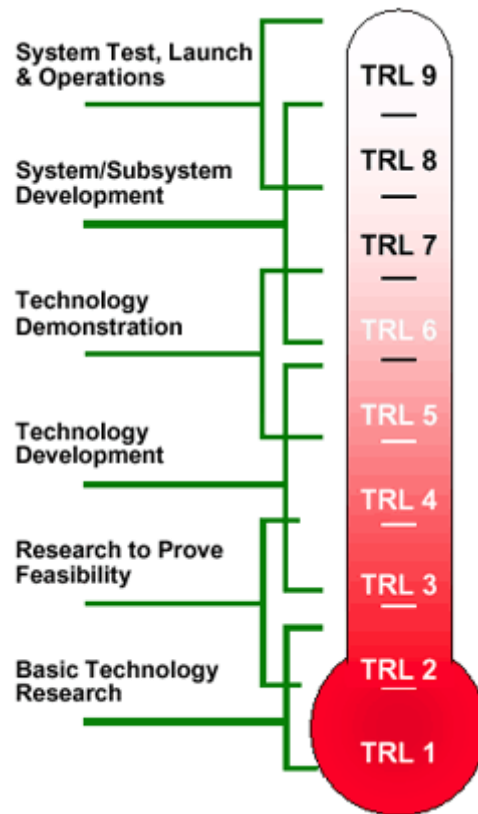


Figure 1-1. TRL scale summary. (NASA public domain image)

Finally, research in aerospace can instead focus on the processes individuals and organizations utilize. Of interest, and the focus of this body of work, are the processes used in designing new aerospace products, particularly early design of aircraft. A preliminary examination of the processes used in design exposes potential gaps or weakness, and these yield potential research opportunities.

1.2 Design processes

The aim of research in the area of aerospace design and development *processes* is essentially centered on arriving at a better resulting product and/or arriving at the same result more quickly and with fewer resources. The processes for aircraft design have the potential for exploitation for innovation.

1.2.1 Archetypal aircraft early design iteration process

The design process, particularly as it relates to early aircraft conceptual design, lives on the upper left side of the classic development “V” model (see Figure 1-2). Early design has significant overlaps with requirements and begins with initial requirements, including both elicitation of new requirements as well as analysis of existing requirements.

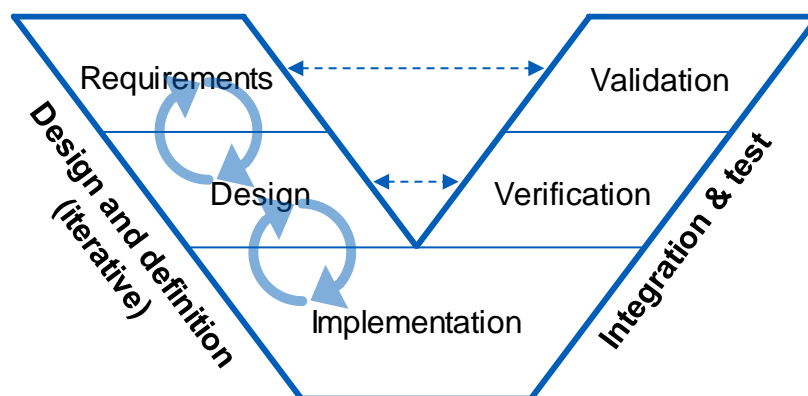


Figure 1-2. Simple depiction of “V” model of development.

Design is a continuous iterative cycle of investigations, refinements, trade studies, and decision-making. Iterating and making engineering design decisions in the early conceptual phase of design can be as much art as science in that it is non-deterministic, and results may vary depending on the individuals involved (and their creativity). While each project and every iteration within that project are unique, a typical individual generic trade study within early design can be broken down into roughly six subprocesses as labeled and defined by this author as follows:

1. The **triage** phase, for lack of a better term, is a precursor to making formal decisions. It is when the decision is made regarding which trade-off studies to investigate. This is the key step of selecting with each iteration which areas are worth investigating and which trade studies are the best use of limited resources to maintain a balanced design effort.
2. The **identification** phase is when the designer identifies alternatives as potential candidates for a final design choice. This could be identifying distinct alternatives or a continuous trade space or design domain.
3. The **analysis** phase is when alternatives are analyzed. The result is complete (or as complete as practical at the given design phase) information regarding performance of potential candidates or performance variations across the design domain.

4. The **communication/understanding** phase is when the results of the preceding phases, particularly the analysis phase, are communicated to stakeholders as well as the designer(s) themselves for the purpose of understanding the alternatives or design domain.
5. The **decision** phase is when a single alternative or design point is chosen from the many possibilities.
6. The **documentation** activity, which ideally runs continually in parallel with all the previous phases, is when all key inputs and results of the previous five phases are preserved for future reference and for inclusion in reports, proposals, etc.

These activities are depicted graphically in the flowchart in Figure 1-3.

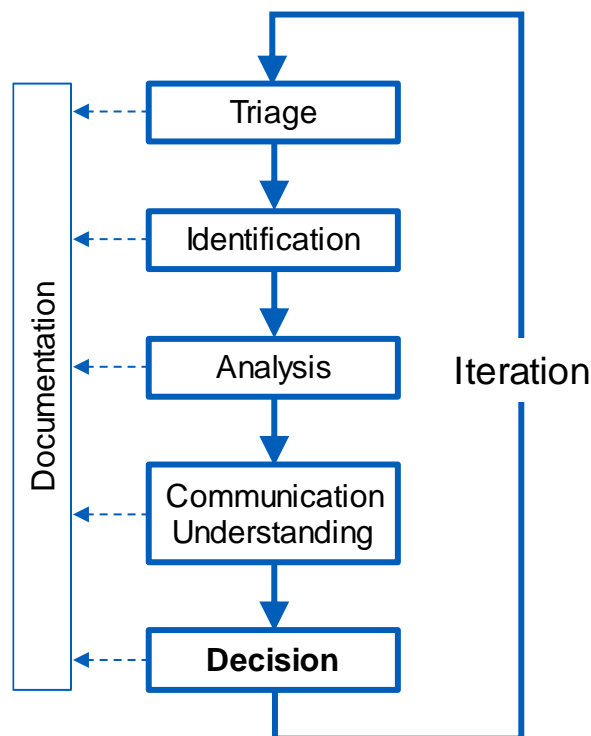


Figure 1-3. Typical design iteration process.

1.2.2 Opportunities in the design iteration process

In the context of conceptual design and the iteration process outlined above, there exist several deficiencies and shortcomings. Some of these shortcomings have potential to be overcome, creating opportunities for further advancement.

The triage phase should involve some formal processes for down-selecting trade-off studies to conduct, for example by doing some cursory parametric sensitivity analyses. This can be recursive in that the triage phase can be treated as a design decision making process in and of itself. However, there often are insufficient resources for such a formal process, and the selection of which trade-offs to investigate is often made using experience and instinct and/or by higher-ups within an organization with a different set of motivations and incentives.

The identification phase in some cases must rely heavily on the creativity and inventiveness of the designer. This is especially the case when configuration alternatives must be synthesized. Because of the reliance of the very subjective experience, meme pool, and creativity of the designer, design synthesis in the identification phase runs the risk of overlooking potentially superior alternatives.

Most shortcomings and therefore most potential advances in the analysis phase of decision making are discipline- and industry-specific. Engineers and scientists are constantly striving to make estimations of system performance more quickly, more accurately, and more robustly. One potential opportunity to improve the analysis phase that is not quite as discipline-specific lies in more efficient ways of linking analysis modules together in the larger context for purpose of, for example, speed of convergence.

The communication & understanding phase suffers from a lack of easy-to-use and accessible multi-dimensional visualization tools. More importantly, there can be significant overhead and effort involved in good communication, e.g. setting up and running parametric analyses and/or making clear, well-formatted visualizations of results. This overhead represents a barrier to creating artifacts useful for understanding and communication. An example of this would be a designer modifying input cells in a spreadsheet-based analysis tool until the resulting outputs “look good.” Doing this, despite the full analysis capability being repeatably in place as software, the designer gains only a modest understanding of the problem, and other stakeholders have no way of gaining even that much understanding.

If the understanding of the problem is thorough, a good result from the decision phase is likely. However, even with good understanding, this is not guaranteed. Decisions can be made that are not entirely justified. Sometimes the final decision is not 100% supported by the preceding objective analysis, for example to capture a qualitative preference or quantitative information not explicitly integrated into analysis. In this case, a decision is made, and a design alternative is chosen that is not directly supported by the quantitative analysis alone. Conversely, when making an objective decision based solely on analysis results, a decision is possible that goes against the intuition of an experienced designer or other stakeholder.

The documentation phase, like the communication phase, can be labor-intensive and without immediate gratification. There is significant overhead involved in producing proposal-quality text and figures, and the step of documenting decisions may be left out at the time the decision process is actually conducted. Delaying documentation introduces the risk that the justification for the decision will be forgotten.

Examining the typical iterative design process exposes some potential opportunities for novel approaches, as well as revealing some weaknesses in the process that translate into additional opportunities for improvement. These opportunities and the desire to create advances in the design process sparked the motivation for exploring new design process techniques for this work.

1.3 Motivation for exploring a different approach

Several factors combined make up the motivation and inspiration for undertaking this work and create the rationale for why the work is worthwhile.

1.3.1 Software-centric contemporary workflows

Computers and software play an intrinsic role in contemporary aircraft design and trade studies. At each stage of a project, not only has software replaced manual calculations, but software is often written even for the simplest of tasks. From day one of any type of new technical design investigation, for example, conducting initial ‘back of the envelope’ hand calculations using pencil, paper, and a calculator has been replaced by writing formulae into a spreadsheet or a simple software script. In this way, the designer has access not only to the results of initial calculations but has concurrently created a useful tool whereby calculations are repeatable at very low cost with different inputs and assumptions.

Unfortunately, this is where much of the advantage often ends, as those low-cost repetitions of calculations are too often only repeated with very manual modification of the inputs and assumptions. This still offers a significant advantage for speed and consistency, but, crucially, the mental and decision processes of the designer remain the same as when the investigation is done without software at all. Other aspects of this workflow also remain the same, for example still only examining a very small number of parameters (usually only one or two) simultaneously.

Because software is ubiquitous in design as an instrument to be wielded by the designer, it is worthwhile to explore techniques that have the potential to augment the mutualistic and complementary relationship between designers and their software. In this way, more of the potential advantages of already software-intensive workflows can be leveraged, and further synergies between human designers and computer-based software can be realized.

1.3.2 Un- and under-captured stakeholder wisdom

Information that is un- and under-captured is an untapped resource that could be further exploited in the early design process. While the whole body of knowledge of the designers and others involved constitutes all the information available, in the context of this work, the term “stakeholder wisdom” is used to more exclusively describe the knowledge that exists among participants but that is not fully captured in the modeling and usually also not in documentation. This is similar to the term “expertise” in describing some of the less tangible knowledge that is nevertheless still respected and may still influence design results through other parts of the process such as design reviews or subjective design decisions.

There are three typical categories of information that affect the outcome of a design that are relevant to this un- and under-captured stakeholder wisdom. Assumptions are the inputs to the design and modeling that are defined by stakeholders (usually designers) other than documented requirements and specifications. Preferences are traditionally documented by customers as

requirements; slightly more nuanced preference information is commonly expressed, for example, when requirements are defined with both a threshold and an objective value. Model behavior determines the predicted performance of the system and therefore influences decisions that are based on these predictions. The typically under-captured stakeholder wisdom associated with these categories is summarized in Table 1-1 below and discussed in more detail in the context of an example in Section 3.3.

Table 1-1. Typical categories of un- and under-captured information.

Category	Typical under-captured information
Assumptions	Nuanced and richer information regarding the assumptions used for analyses.
Preferences	Preference information, not explicitly written in requirements or specifications, on figures of merit (FOMs) and other design parameters.
Model behavior	Known behaviors of the analyses in modeling the system behavior and faithfully capturing (or not capturing) the true physics of the system.

To illustrate the role stakeholder wisdom plays and the effect it has on the design process, consider a simple abstract example of an early trade study in a new aircraft design project: payload capacity, which the customer already specified as a requirement or preference, versus the operational cost per unit of payload carried for the mission distance (the notional figure of merit to be minimized to achieve the ‘best’ system), shown in Figure 1-4.

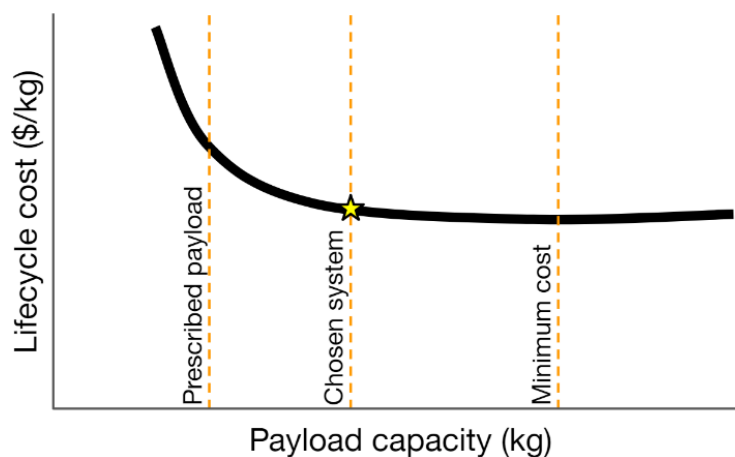


Figure 1-4. Illustration of early payload capacity trade study. (Sartorius & Hornung, 2018, p. 3)

In this situation, a significant cost improvement is possible with a system designed with a payload capacity several times larger than originally requested by the customer. In the hypothetical case that numerical optimization would be used for this study, the resulting design would either have the payload stipulated by the customer (if set as a constraint) or it would be a system several times larger than what the customer originally requested. However, the choice made by the designer was neither, with a point in between selected instead.

What preference information led to the decision, and how much of that information was easily accessible to the designer *before* the shape of the curve was known? This question is the inspiration for the technique presented in this work.

The answer is that a significant amount of this information is present early in design but is simply not captured in a conventional design workflow. However, it is there, waiting to be exploited. Capturing this stakeholder wisdom in a formal manner should therefore be low-hanging fruit for taking advantage of. With the right software-enabled techniques, we can not only harvest that information, which has hitherto been out of reach, but we can also put it to use in meaningful ways that can improve design development and product outcomes.

1.3.3 Increasing chances of product success

Innovation in design processes increases the chances of a product or program being successful by reducing time and resources needed for converging on an acceptable result or by increasing the quality or refinement of the results of a given iteration or trade study. By exploring a technique that captures un- and under-captured information, there is potential for significant benefits to the design process for the reasons discussed here.

1.3.3.1 Access to unexplored areas of the design space

It is often the case that as design progresses, certain design variables, parameters, requirements, and decisions transition from a free, undetermined state to being set to a fixed value. This is often necessary for the design to move forward. However, every time a parameter is fixed, the design space is effectively pruned of possibilities. This means that there is a potential for excellent design possibilities to be undiscovered and unexplored. By maintaining richer information about design parameters for longer in the design process instead of locking them in to fixed values prematurely, the pruning process is delayed for as long as possible, and the chances of discovering better solutions improves.

1.3.3.2 Delaying locking in requirements

This pruning of possibilities also applies to requirements. Product failure often results from requirements that are locked in prior to fully understanding the costs and compromises of those requirements. Maintaining flexibility on the requirements until later in design avoids this. However, doing so normally carries an extreme burden and significantly slows the pace of development, so there are strong incentives to fix requirements as early as possible. By facilitating carrying uncertainty in the requirements deeper into the design and development instead of prematurely pruning the product space, the development can adapt and make necessary changes in reaction to new insights and information.

1.3.3.3 Design decision freedom, tracked and justified

As illustrated in the example payload capacity shown in Figure 1-4 above, it is possible for a given trade study to result in a spectrum of possible decisions. In this case, two decisions are easily justified: either the decision is conforming to

a provided specification or it is based upon optimizing an objective, quantitative parameter. It is all the possibilities in between where decision-making is not always easily justified. Conversely, sometimes the objective analyses and data drive towards a decision that goes against the wisdom of the designer or other stakeholders. This wisdom should not be undervalued, and if it is, the pressure to make easy-to-justify decisions can prevent better design possibilities from being selected. By explicitly capturing the information in a formal way that is documented and traceable, designers have the justification in hand to enable the freedom to make decisions that will lead to better results.

1.3.3.4 Process efficiency

One consequence of stakeholder wisdom sometimes being undervalued is that research or other efforts are spent only to reach a conclusion that could have been found using only the existing knowledge on hand amongst the designers and stakeholders. By enabling and leveraging the use of that knowledge, some of those superfluous efforts can be short-circuited, making better use of resources in design and accelerating development or, conversely, accomplishing the development with fewer resources spent.

1.3.4 Focus on early design

In a typical aircraft development program, the decisions that have the greatest effects on the ultimate outcome are made early on before the majority of the effort has been spent. In other words, only a relatively small early investment has a disproportionate impact on the overall program (see Figure 1-5), and there is quickly decreasing flexibility to make changes to the design definition that is set in this early time. This means that new processes and approaches that can yield gains will have a greater relative impact when the improvements are realized in the earlier design states. Therefore, the focus of this work is on early design studies for superior returns in making better decisions when so much is in flux and the development is less hindered by high programmatic inertia.

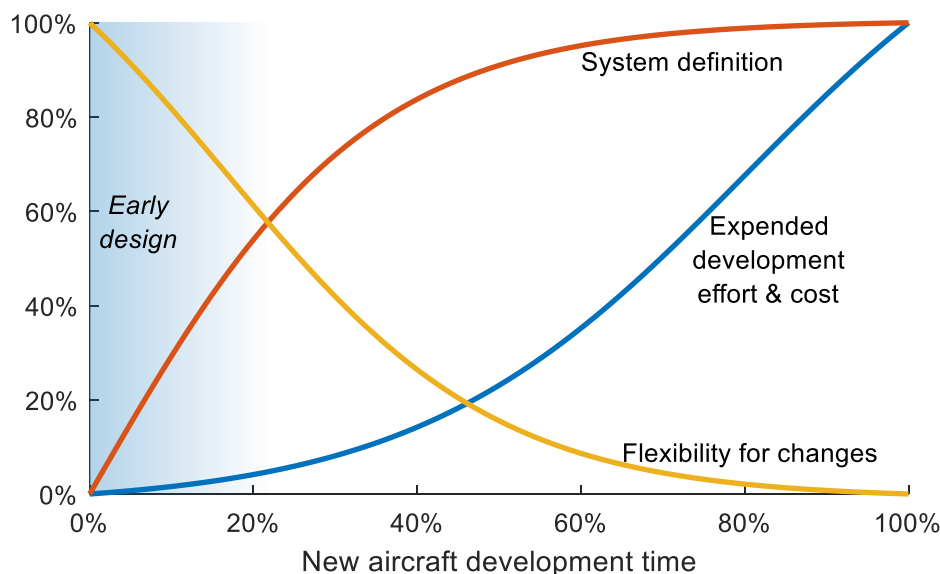


Figure 1-5. A disproportionate amount of definition is set in early design.

For conceptual aircraft design, 'early design' would typically fall prior to a conceptual design review for a larger program, prior to a preliminary design review for a small program in a small company, and prior to the use of formal requirements and change management tools and processes. In the context of this work, 'early design' also spans into market analysis, requirements elicitation, and other activities sometimes conventionally thought of as preceding the design phase. That said, not all 'early design studies' take place in the early phases of a program like in a conceptual design phase. The term as used here applies to any effort that has a combination of the following attributes or characteristics:

- The study is part of the early stages of any trade study or exploration of the design space.
- It is part of the initial investigations into some aspect of an aircraft or system that has not been previously given significant consideration in the overall development.
- Project requirements and/or objectives are unlikely to be extremely well defined and understood.
- The analytical models in use have a lack of maturity and/or fidelity.
- There is a high likelihood, or even a near certainty, that the design will change in future iterations.
- The designer and other stakeholders are more interested in understanding the design space and the decisions bringing the design closer to a *good* area of the design space, rather than a single best and final design decision.

These motivations coalesce to inspire the undertaking that is the focus of this work, where there is surmised to be a potential for a new approach to leverage the already software-centric workflows to capture additional information and put it to use to improve the results of early design processes.

1.4 Objectives

The objectives of this new approach are split into two main parts:

1.4.1 Objective 1: Facilitate formally capturing stakeholder wisdom

The first and primary objective of the work is to create a method for formally and quantitatively capturing the aforementioned un- and under-captured information and knowledge. This makes available preferences and other types of information that are present *a priori* in early design but not normally explicitly used. The aim of the technique is to capture as much as possible of the preference information designers normally do have *a priori* but typically only use to make design decisions after completing activities that make the design space and trade-offs better understood.

This requires first devising a format that can effectively capture often qualitative information in a way that can be at least roughly quantified for use by automatic processing methods. There is also the important aspect of making sure that the approach to capturing is as low effort and low friction as possible. Since the information for the most part already exists in the knowledge base of the stakeholders, if the approach is ever to see practical real-world usage, it is crucial that the process for capturing the information is relatively low overhead and does not present an unpalatable burden to stakeholders.

1.4.2 Objective 2: Automate the integration of captured information in design space exploration and decisions

Simply capturing the stakeholder information has limited usefulness by itself. There are many potential ways to put the captured information to good use. The primary aim here is to use the information in such a way as to automate some of the design space exploration and decision making. In this fashion, it is possible to accelerate the design and decision-making process with the automated search process, but with an automated search that is informed with as much available information and knowledge as possible. This essentially adds automation where the additional captured information makes it possible but keeps the designer and other stakeholders in the loop for decisions where it makes sense to do so.

This work explores an approach to use the power of computational tools, specifically existing optimization search algorithms, to put the information to good use for automating design space exploration. The stakeholder wisdom information involved here is, by definition, not normally used in multidisciplinary design optimization (MDO), so to meet this objective, it is necessary to capture this type of often nuanced and nonlinear information and knowledge in a quantitative way that can enable leveraging the power of these search algorithms to provide a useful result. Thus, the aim is to accelerate the design and decision-making processes and accelerate design iteration cycles through automatic searching of the design space that is as informed as possible to let the search algorithms, at every step, to be driven toward similar design directions that a human-in-the-loop designer would be.

Capturing knowledge and putting that captured information to good use were the two primary objectives of the work, which came with certain anticipated benefits. However, in executing the project, certain unplanned secondary benefits were also realized, which will also be discussed.

1.5 Structure of this work

The present thesis describes the specific approach used to achieve the primary objectives for improving design processes. A theoretical discussion is supplemented by specific discussion of implementation and experiments with example design cases.

Chapter 2 discusses the state of the art related to capturing stakeholder preferences and information, particularly in the domain of requirements elicitation. Some prior work is also summarized regarding design processes and techniques for conducting trades studies and visualizing and understanding the design space, including techniques for accounting for uncertainty and numerical optimization.

Chapter 3 begins to go deeper into the applicability of the approach and the issues to resolve in developing the new technique, called the Well-Informed Search Design Optimization Method (WISDOM). The specific types of information targeted for formal capturing and implementation are illustrated with a simple regional airliner sizing example, and then the approach to implementation of that information in an optimization workflow is presented.

Because this technique is meant to be a relatively integral part of a design and development process, the specific implementation in software is important. Chapter 4 lays out the workflow, architecture, and algorithms involved, along with some of the reasoning behind certain implementation decisions.

To further solidify the illustration of the usage of the technique in design processes, example cases are laid out in Chapter 5. The regional airliner from Chapter 3 is used as one example case, and the second example case explores the possible redesign of a narrow-body airliner wing with a winglet.

Chapter 6 concludes with a discussion on how the technique performs against the objectives of this work, some of the unexpected side benefits that were realized, and also some drawbacks and areas where future improvements or new capabilities may be worthwhile to build.

2 State of the Art

The typical aerospace program process begins with requirements, and there are established approaches for eliciting, formulating, and analyzing requirements. During the iterative development of the system, design decisions must be made, and there are many techniques proposed and in use for formally structuring trades studies and decision-making. A large subcategory of decision-making approaches is the broad field of multidisciplinary design optimization. Finally, there are additional approaches and techniques in development that aim to account for various types of uncertainty in design and optimization.

2.1 Requirements elicitation and analysis

One primary objective of this work is to facilitate formally capturing stakeholder knowledge. Though the focus is on capturing the types of knowledge and other information that is often un- or under-captured in typical design, there are still many areas where information *is* formally captured, particularly in developing requirements. It is therefore worthwhile to take stock of some of the established approaches used for capturing stakeholder preferences and knowledge in the creation of system requirements.

2.1.1 Market research

Briefly, market research is any effort that seeks to gain insights about customers or potential customers in order to support business and product decisions. In the context of the commercial aerospace industry, this applies to sectors such as airliners, business jets, general aviation, and even experimental kit planes and ultralights. In these industry sectors, market research translates into figuring out *what* the company should build that will have a market fit and profitable sales. The aforementioned “what” translates into top-level requirements. In other words, market research seeks to determine the requirements for the next product, and there are several categories of techniques available in the domain of market research (McQuarrie, 2016), as summarized in Table 2-1.

Table 2-1. Market research techniques overview.

Category	Example technique
Archival	Secondary research: Search existing information collected for another purpose.
	Big data analytics: Special kind of secondary research characterized by large datasets and software-based analysis.
Qualitative / interview	Customer visits: Researcher and colleagues visit multiple customers.
	Focus groups: Multiple customers are brought to the researcher at one facility.
Quantitative	Descriptive survey: Collecting data from multiple respondents using specific, quantifiable (e.g. multiple choice) questions.
	Experimentation: Multiple ‘treatments’ (e.g. product features) applied to different groups.
	Conjoint analysis: Multiple treatments applied to an individual.

2.1.2 Traditional aerospace and defense approach to requirements

Many requirements formulation approaches that are advocated for in the aerospace and defense domain acknowledge that requirements, at least to some extent, must be allowed to iterate, for example in response to the program progressing and new information becoming available. However, as larger programs progress, the cost of any changes to requirements makes changes

infeasible. This is especially true of top-level customer requirements and system goals, and this is one reason that, for example, the US Department of Defense systems engineering process, shown in Figure 2-1, which evolved for the development of very large, complex, and expensive programs, features only a one-way flow of customer requirements and needs into the process.

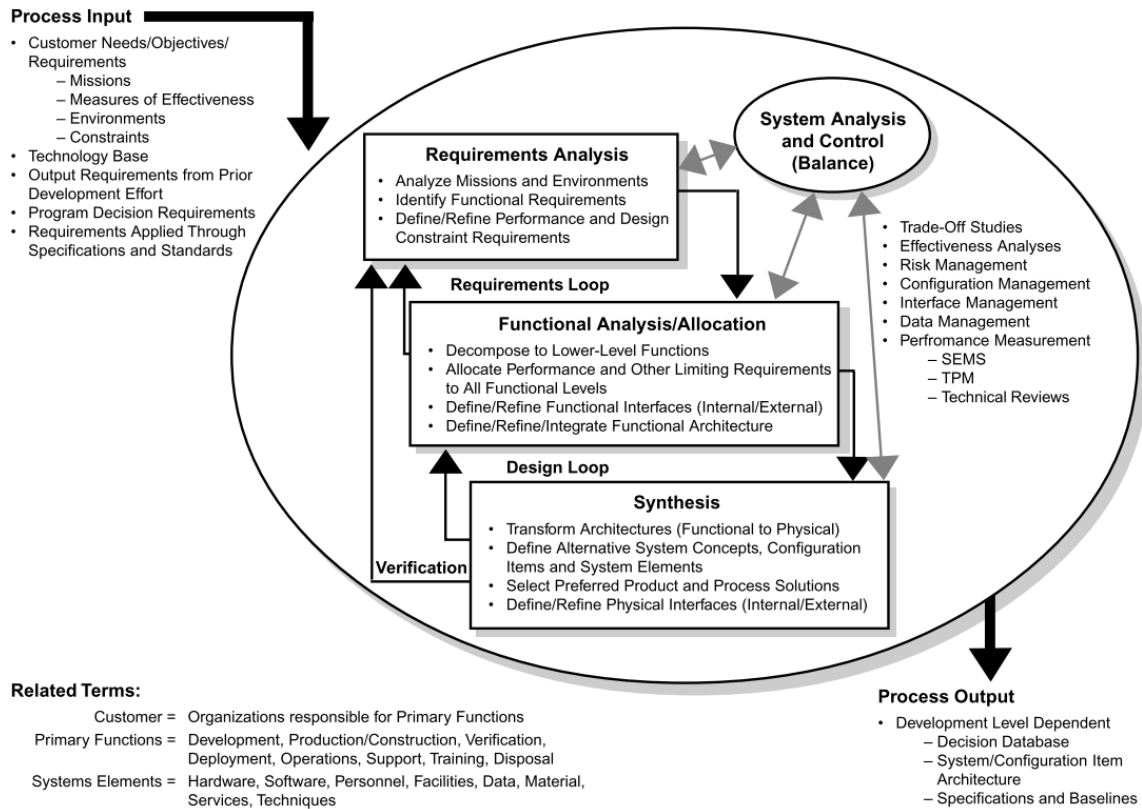


Figure 2-1. DoD systems engineering process. (Defense Acquisition University Press, 2001, p. 31)

Many of the requirements applied to aircraft certified for civil aviation are enshrined in law and regulation and are understandably also taken as immutable inputs. The prevalent recommended practice for development and certification of civil aircraft and an accepted means of compliance with key regulations, ARP4754A (SAE S-18, 2010), focuses primarily on safety assurance in a “development” phase that is a mostly separate follow-on to a “concept” phase. However, it is during this concept phase, which “determines the overall aircraft performance and configuration” (SAE S-18, 2010), when top-level requirements would have an opportunity for analysis and iteration.

The NASA systems engineering process, shown in Figure 2-2, also features a primarily one-way flow of customer requirements (also called “mission requirements” or “stakeholder expectations”) and constraints into the design and implementation process. However, it also explicitly advocates for an iterative feedback loop that includes involving stakeholder expectations.

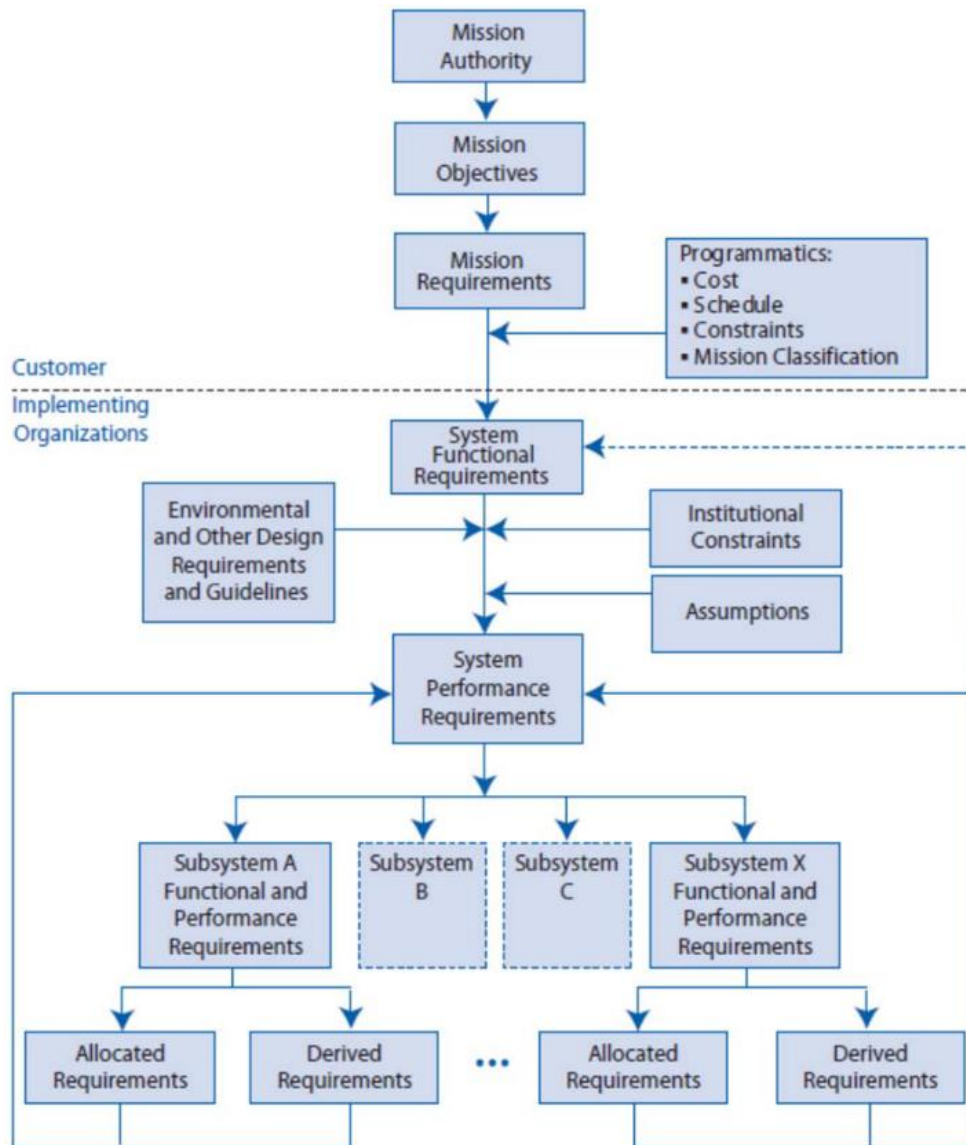


Figure 2-2. NASA systems engineering process. (Hoffpauir, 2017, p. 68)

Since the flow of information from customer top-level requirements all the way down to detailed design and implementation is often unidirectional, it is important that the process of formulating and deriving requirements has some structure and robustness. With primary focus on the development of requirements for software-intensive systems such as avionics and other real-time embedded systems, the FAA's *Requirements Engineering Management Handbook* (Lempia & Miller, 2009) prescribes formulating qualitative system goals and highly structured derived use cases. Since the top-level system goals are less likely to change, high quality requirements are more likely due to the strict adherence to linking and deriving a) use cases from goals, b) high-level requirements from use cases, and c) low-level requirements from higher-level requirements.

2.1.3 Requirements traceability for verification and validation

Early in a program or for smaller projects, it is straightforward to record requirements in a text document or other simple format such as a spreadsheet

in such a way that an individual can have both a broad and deep understanding of all requirements. As development progresses, further low-level requirements are derived, and the system is designed and built to the requirements (left side of the classic development “V” model). When progressing up the right side of the development “V” model, however, two activities add additional burden to the formats and tools used for recording and managing requirements: a) the verification that the system conforms to specifications and b) validation that the system meets objectives, together known as verification and validation (V&V).

Additional needed capabilities for requirements management led to the development of specialized requirements management tools. A prevalent tool for aerospace applications is IBM’s Rational DOORS software, which facilitates tracking of both requirements and compliance with requirements (IBM, 2016). Complex or software-intensive systems present additional challenges for requirements traceability and V&V, particularly those that must conduct a rigorous safety assessment process, e.g., for certified civil aircraft, follow ARP4761 (SAE S-18, 1996), or comply with a rigorous development assurance level (DAL), such as by following DO-178C (RTCA SC-205, 2011). These processes, when traditionally implemented, have many manual steps required for maintaining requirements traceability.

2.2 Design exploration and decision techniques

Development activities, i.e., iteratively moving down the left side of the development “V” model, consist of constantly making a series of design decisions. The second objective of this work is to use the formally-captured information in a useful way such to facilitate better design decision making. Better decisions lead to better results (as measured by system performance, but also by reduced risk, cost, schedule, etc.), so to establish context for the WISDOM technique that is the subject of this work, a sampling of common established approaches to trade studies and decision-making is summarized here.

2.2.1 Trade studies

A trade study is a process to support decision making. It is an objective comparison of as many realistic alternatives as possible while considering as many figures of merit as is reasonable. A good trade study will lead to a sound engineering design decision. Not only must the trade study be well executed using appropriate tools and methods for both the trade study process and technical analysis, but it should also be well communicated such that the results are clear to the designer, decision-maker, and other stakeholders.

Trade studies and other design decision techniques may be applicable to discrete decisions (choosing between two or more distinct alternatives), selection of one from an infinite number of design alternatives on a continuous design domain, or a discrete-continuous combination of the two. In many cases, the technique for conducting a basic discrete trade study is some variation on the decision matrix. One of the simplest forms of the decision matrix is the fully qualitative stoplight chart, an example of which is shown in Figure 2-3, with a row for each alternative and a column for each figure of merit. The stoplight chart is a visual representation of pros and cons of alternatives against various figures of merit.

Tail Type	Prop Position	Drag	Weight	Control	Noise	Tail Fatigue	Prop Fatigue	Looks
Conv. Tail	Tractor	Red	Green	Yellow	Green	Yellow	Green	Yellow
	Pusher	Red	Yellow	Green	Yellow	Green	Yellow	Yellow
T-Tail	Tractor	Green	Yellow	Red	Green	Red	Green	Yellow
	Pusher	Green	Yellow	Yellow	Red	Green	Red	Green

Figure 2-3. Example stoplight chart decision matrix for an electric aircraft propulsion-empennage configuration. (Atanasov, 2011, p. 20)

The next level of decision matrix quantifies or assigns a weighting to each of the figures of merit and uses a quantitative score in lieu of qualitative color coding, yielding a quantitative measure for each alternative considered. Further variations on the decision matrix approach to trade studies delve deeper into the various elements, for example a tiered approach for criteria prioritization and

weighting, illustrated in Figure 2-4, or accounting for team dynamics or uncertainty (Ullman & Spiegel, 2006).

Criteria	Sub-Criteria	Percentages	Ratios
Performance	-----	50.0%	3.3
---	speed	---	3.0
---	acceleration	---	3.0
---	payload	---	2.0
---	fuel consumption	---	1.0
---	ceiling	---	1.0
Reliability	-----	15.0%	1.0
---	MTBF	---	1.0
Safety	-----	15.0%	1.0
---	Safety	---	1.0
Logistics	-----	20.0%	1.3
---	Supply Lines	---	1.0
---	Spare Parts	---	1.0
---	Crew Training	---	2.0
		100.0%	100.0%

Figure 2-4. Tiered approach to decision matrix criteria prioritization and weighting. (Felix, 2004, p. 5)

2.2.2 House of quality

A further variation on the decision matrix approach, the house of quality is a semi-quantitative graphical technique that is applicable to early development while also spanning the divide between requirements development and facilitating making early design decisions. It is often used as a means to facilitate discussions amongst various stakeholders regarding priorities and desirability of various design features and attributes (Hauser & Clausing, 1988; King, 1987).

In the house of quality technique, customer needs are prioritized and weighted. For each design feature, its correlation with a customer need is recorded in the central matrix region of the house of quality. Each level of correlation (e.g. strong, medium, weak, or none) is assigned a value, which when multiplied by the customer priority weighting and summed yields a quantitative guidance on design feature priorities.

The house of quality has a role in early aircraft design for providing diverse and multidisciplinary teams and stakeholders a common artifact to reference for a basis of discussion. Figure 2-5 shows an example house of quality for a multirole jet fighter.

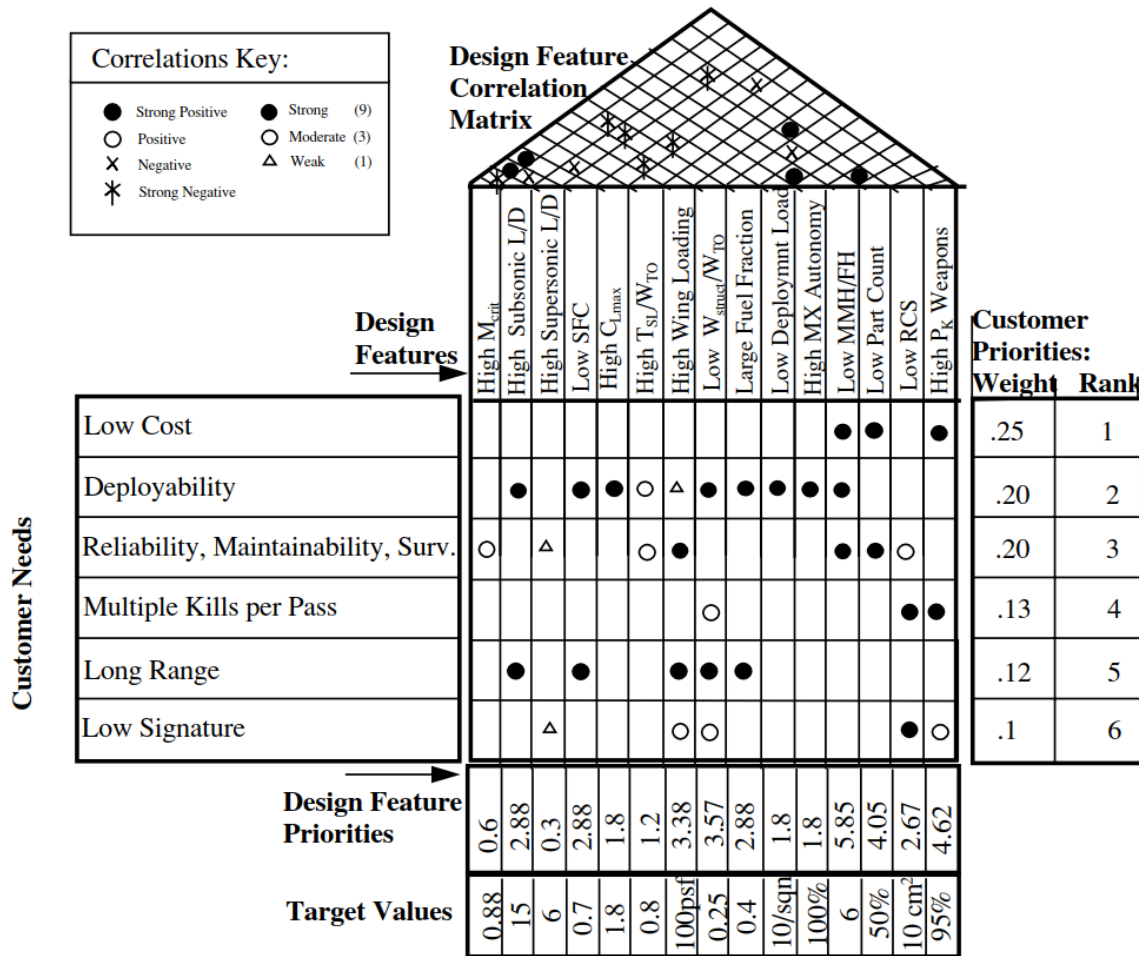


Figure 2-5. House of quality for a multirole jet fighter. (Brandt et al., 2004, p. 32)

One drawback of the house of quality is that it “lacks an explicit indication of the cost of each design priority and decision” (Brandt et al., 2004, p. 31). On the other hand, somewhat mitigating this and a somewhat underrated feature is the design feature correlation matrix in the ‘attic’ of the house of quality. Despite having no quantitative impact on the bottom-line results of the house of quality technique, the design feature correlation matrix in the attic serves to offer a common basis for discussion on the constant compromises that must be made in aircraft design.

2.2.3 Design space understanding and visualization techniques

One way that trade studies and other design decision aides and techniques could be described is simply as activities in expressing a complex situation in a way that a decision-maker can understand in order to make well-informed decisions. To this end, there are several different ways that designers create visual artifacts in order to better understand the design space. These types of plots are most common for examining and better understanding trade studies on a continuous domain.

Figure 2-6 shows an example of one of the simplest forms of design space visualization: a plot in cartesian coordinates examining the effects of a single design parameter upon a sole other parameter or figure of merit of interest. In

this case it is an examination of the effect of wing aspect ratio on the overall weight of a UAV wing built with a solid foam core and composite skin and spar.

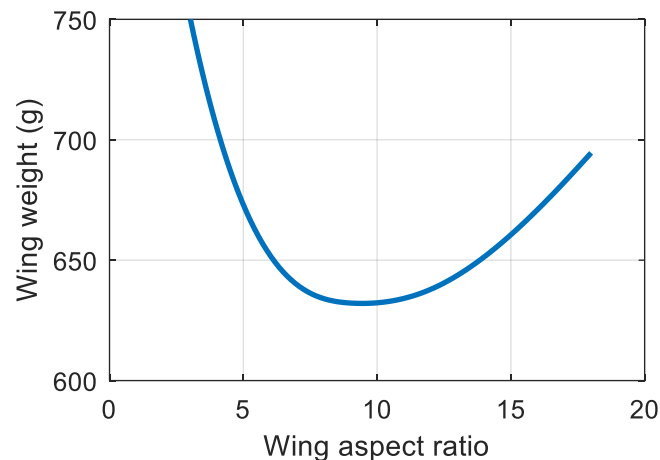


Figure 2-6. Example simple two-parameter plot for a solid-core UAV wing.

An extension of the simple sweep of a single parameter is examining the effects of multiple parameters. This, with an example shown in Figure 2-7, is commonly called a sensitivity analysis, and it is an extremely useful technique for understanding which parameters have the strongest effect on the design. Not only does this information help inform immediate design changes or adjustments, but it also provides benefits in directing limited resources in future design iterations and trade studies.

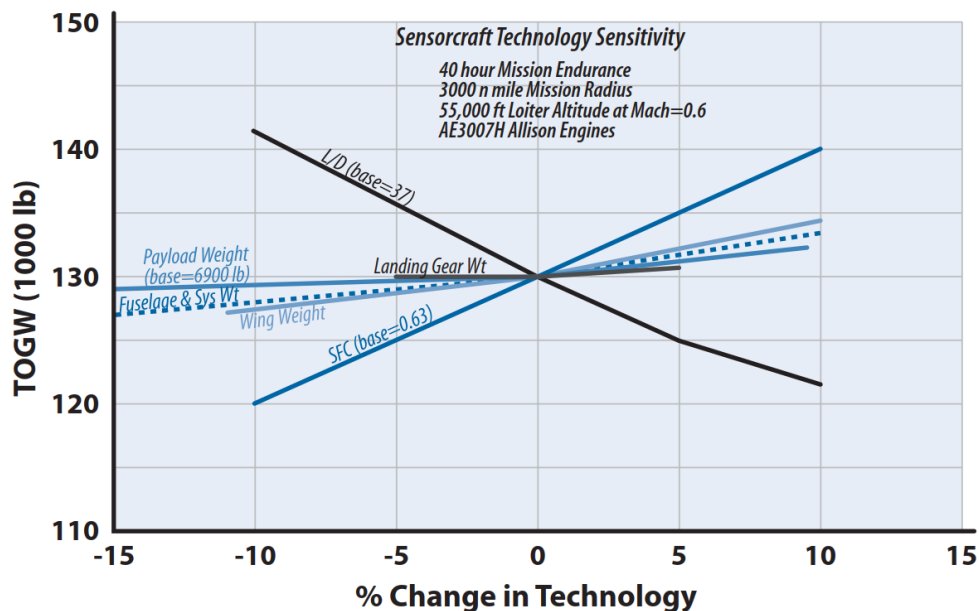


Figure 2-7. Example sensitivity analysis for a HALE ISR UAV. (Nicolai & Carichner, 2010, p. 662)

There are other common visualizations used in early aircraft design to better understand the design space. A classic example is the very common constraint diagram, which shows various design constraints (dictated by requirements) on

a field of all the possibilities of wing size and powerplant size, for example in terms of wing loading and inverse power loading, as shown in Figure 2-8.

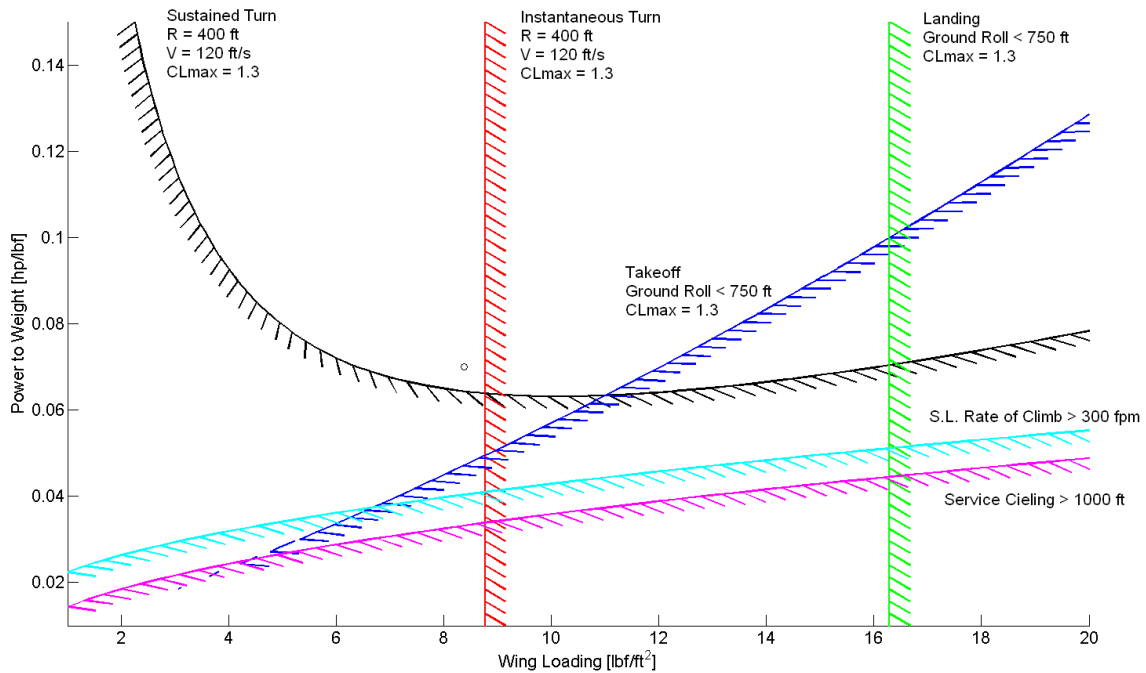


Figure 2-8. Example constraint diagram for airplane conceptual wing and powerplant sizing visualization.

Many variations on design space visualizations exist. A relatively common variation on the constraint plot adds information in the form of contours of some objective function or figure of merit. Transforming the axes such that this FOM is on the ordinate axis can yield a simple carpet plot, shown in Figure 2-9 for a similar wing size and powerplant size visualization. Variations on the carpet plot can add further independent and/or dependent variables to the visualization (for example by using the abscissa for either), and the type of visualization can be used to understand a myriad of different aspects of a design or trade-off.

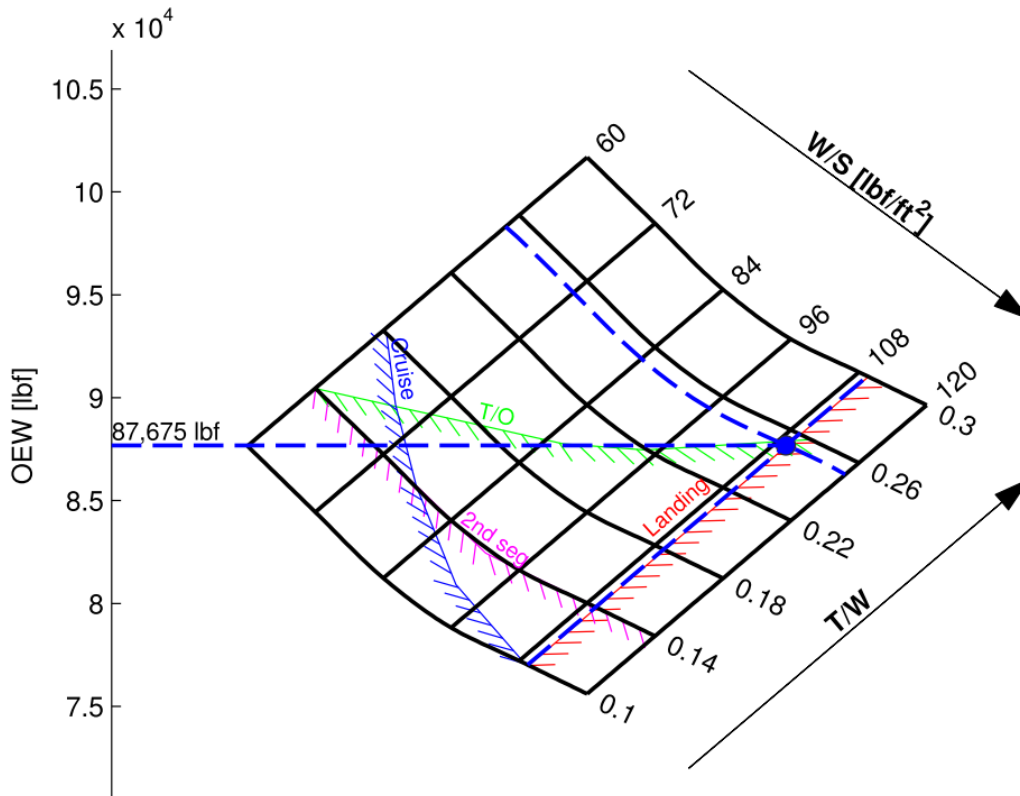


Figure 2-9. Example carpet plot with single figure of merit on the ordinate axis.
(Oberhauser, 2013, p. 32)

Visualizations of the design space become more challenging when a higher number of variables, or design space dimensions, is involved. It is common to examine two or three design variables at a time for early aircraft design because usually these two variables (commonly wing size and powerplant size) have an oversized and dominant effect on the design's performance and outcome. However, there are many applications, within and outside of simple airplane design, where understanding more parameters simultaneously is advantageous. Three-dimensional plots, color, marker size, and other tools can be used to capture higher dimensions, but one slightly more scalable method is to use a matrix of two-dimensional plots, effectively visualizing planar slices of the design space taken about some baseline point. Two visualizations that leverage this approach are shown in Figure 2-10 and Figure 2-11. Both are in the context of initial helicopter sizing and design space understanding. The nature of rotary wing aircraft means that it is very difficult to make effective design decisions examining only two design parameters simultaneously.

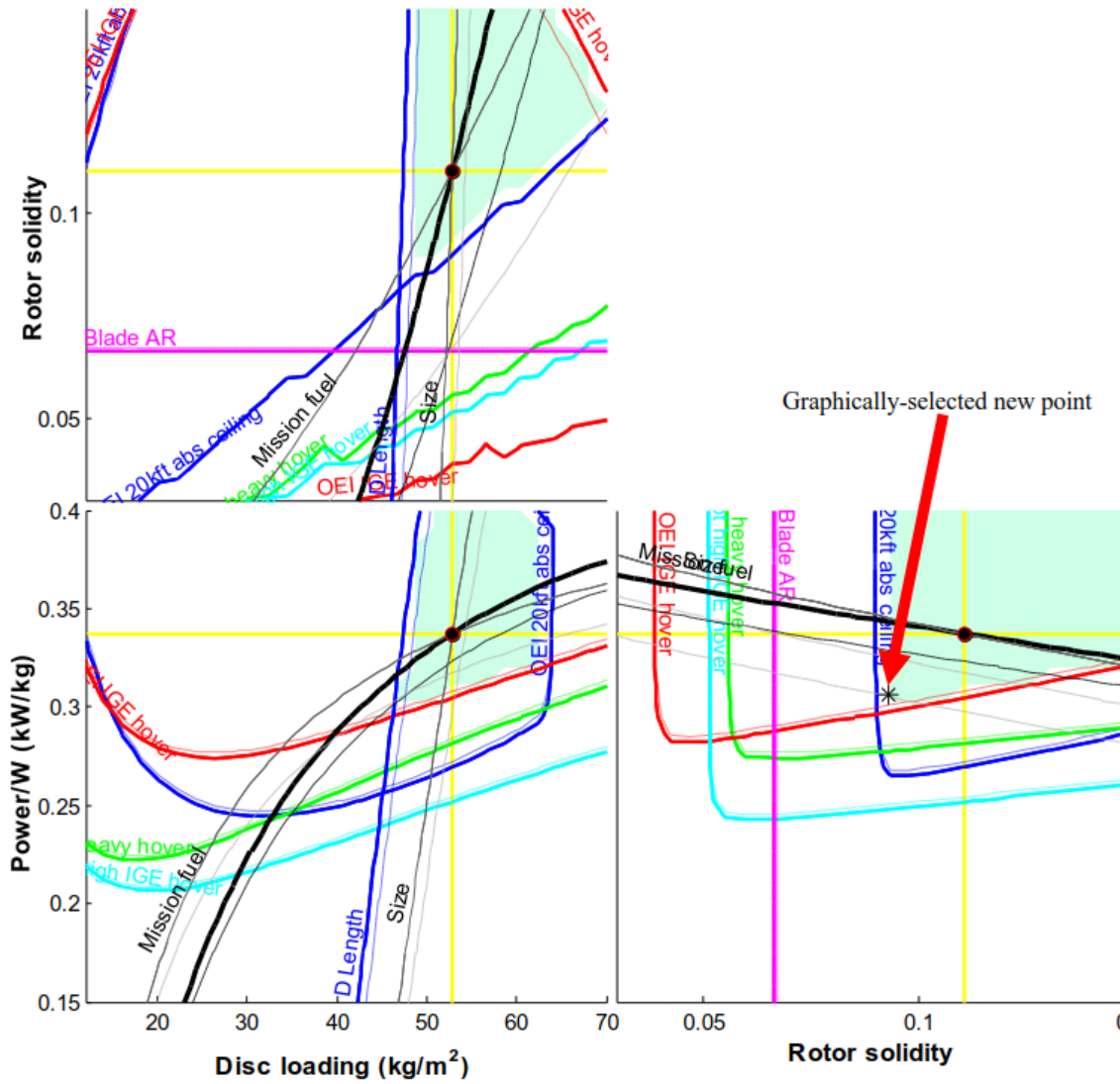


Figure 2-10. Constraint matrix for conceptual helicopter sizing. Three design variables (cardinal axes), six constraints with 1% margin contours (colored lines), and objectives with sensitivity (gray). (Sartorius, 2011c, p. 8)

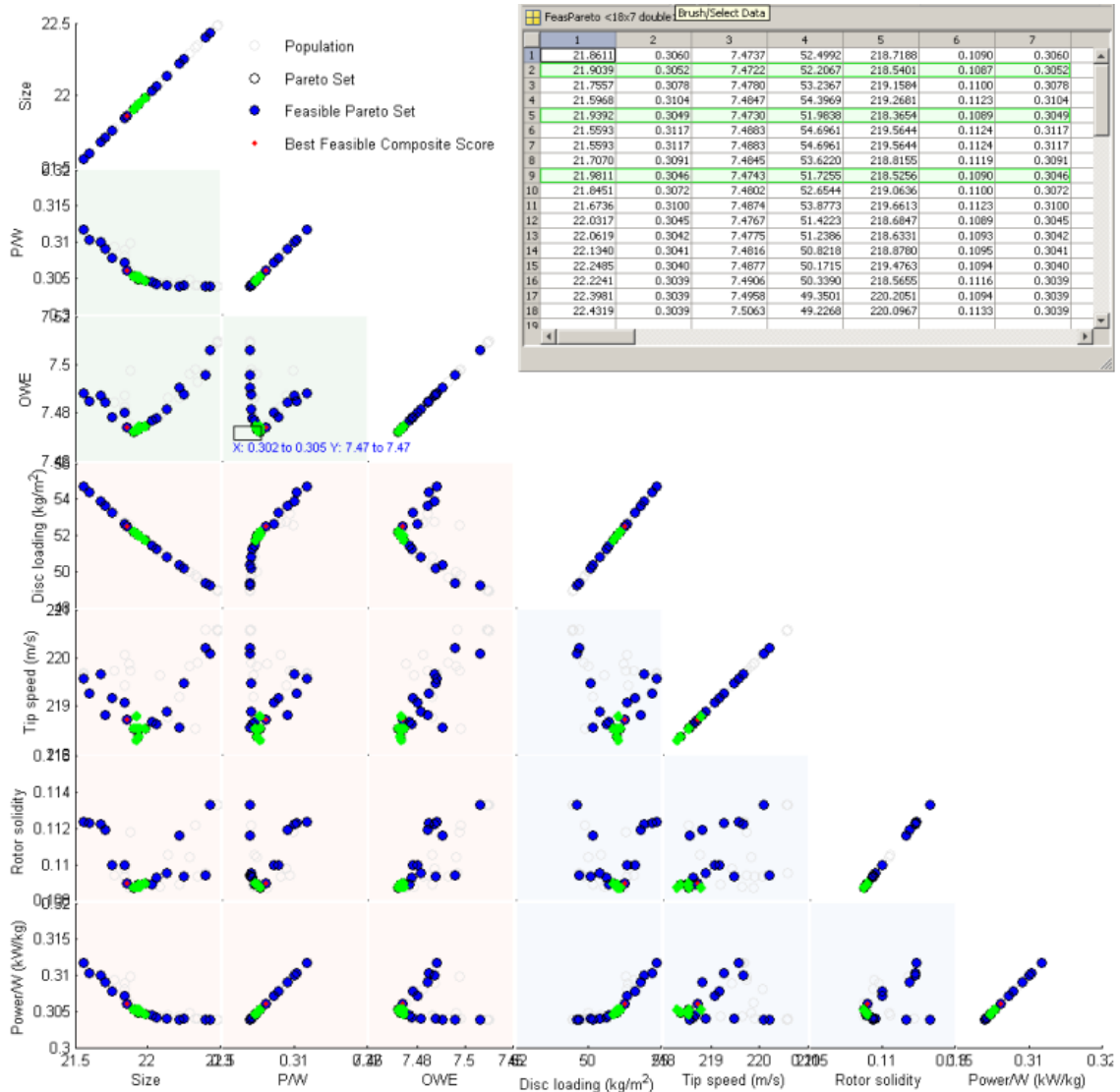


Figure 2-11. Pareto set visualizer with data brushing for design space exploration. (Sartorius, 2011c, p. 7)

2.3 Accounting for uncertainty

While it is possible, with the understanding that comes from the design space visualization techniques discussed above in Section 2.2.3, to account for some uncertainties in calculations, add margins, and similar, there are also techniques to integrate information regarding uncertainty more directly into engineering analysis and calculations.

Normally, design and analysis calculations are executed using certain variables and parameters as inputs that take on a singular value. The yield stress of a material, for example, may be taken as 50,000 psi (345 MPa). However, it is known that this value will not be exact. Instead of executing design and analysis calculations with these singular values, it is possible to perform the calculations using distributions with probabilistic design and analysis (Haugen, 1980). This allows the yield stress input to instead be, for example, a normal (Gaussian) distribution. Similarly, instead of defining the input variables and parameters as a probability distribution, fuzzy sets can be used to define the inputs to calculations (Wood et al., 1992), which has been shown to have some computational advantages for preliminary design over probabilistic design and analysis (Wood et al., 1989).

The concept of accounting for uncertainty can be extended to the field of optimization as well (optimization is discussed in Section 2.4 below) in the field of robust optimization. A survey of robust optimization is presented by Sözüer and Thiele (2016), which contains a broad survey that includes a discussion of applications of robust optimization beyond just engineering design.

A basic form of robust optimization simply introduces worst-case tolerances to input parameters to ensure feasibility of resulting designs. This can be overconservative in applications where performance is demanding or margins must be kept small, so statistical tolerances, an extension of the concepts of probabilistic design and analysis, can be used instead. These types of robust optimization approaches are typically to ensure feasibility. However, another type of robustness involves minimizing the sensitivity of the design solution's performance to variations, either in the input parameters, the design variables, or both (Parkinson et al., 2018, Chapter 9). Fuzzy optimization is a similar extension of the concept of using fuzzy sets for engineering calculations applied to optimization to account for uncertainty (Rao, 2009, Chapter 13).

2.4 Optimization

Many capabilities are available to designers by the simple presence of the nearly-free repetitions of a given analysis inherent to computers and software being so central to modern development work. One prominent example is optimization, which employs a variety of search algorithms to automatically search the design space to find an optimal design. Optimization is an excellent tool that has unique capability for handling design problems with numerous variables of interest that all interact, which is the normal situation in aeronautical design.

While it is straightforward to fully understand a problem and make an informed design decision using plots and visualizations of only a few design variables and a handful of dependent parameters, most optimization algorithms easily scale up to larger problems that a human designer can only grasp a few dimensions at a time. Consequently, optimization also enables exploring these higher-dimension design spaces, not only to find optimal solutions, but also often enabling or accelerating discovery of feasible solutions in cases of highly constrained problems and identifying and gaining insight into driving design constraints.

There exist a variety of algorithms, techniques, and off-the-shelf software toolboxes for optimization. Various approaches are typically either deterministic or stochastic in nature. There is also a categorization of approaches into methods that minimize a single objective function versus those that account for multiple objective functions.

2.4.1 Single-objective search

The basic form of optimization is a minimization of a single function of one or more design variables, typically of the form

$$\mathbf{x}^* = \min_{\mathbf{x}} f(\mathbf{x}), \quad (1)$$

where \mathbf{x} is a vector containing n design variables (also known as decision variables),

$$\mathbf{x} = \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix}, \quad (2)$$

f is a function returning a single objective value, and \mathbf{x}^* is an optimized design solution.

It is typical in engineering optimization to also have many constraints that must be satisfied for a solution to be valid, feasible, and meet requirements. The most basic constraints are the simple lower or upper bounds (LB and UB , respectively), also called side constraints, on the values of the design variables themselves,

$$LB_i \leq x_i \leq UB_i, \quad i = 1, 2, 3, \dots, n. \quad (3)$$

It is quite common to also have nonlinear inequality constraints,

$$g_j(x) \leq 0, \quad j = 1, 2, 3, \dots, m, \quad (4)$$

and nonlinear equality constraints,

$$h_k(x) = 0, \quad k = 1, 2, 3, \dots, p. \quad (5)$$

Sometimes simple linear constraints are also imposed, either equality constraints ($Ax = b$) or inequality constraints ($Ax \leq b$).

There are many valid approaches and algorithms for single-objective search, as well as a variety of associated methods for handling constraints. While there is no single taxonomy of single-objective optimization algorithms, it is common to label algorithms as deterministic versus stochastic, local versus global, and sometimes also unimodal versus multimodal. Further differentiation between search methods is often associated with how constraints are addressed and integrated into the algorithm.

Deterministic algorithms will yield the same result for a given objective function, constraints, search starting conditions, and algorithm parameters. Stochastic approaches, on the other hand, are less dependent on starting conditions but are not guaranteed to yield consistent, repeatable results due to the intentional introduction of randomness.

A common theme in local search methods is, from a given starting point in the design space, to determine the direction of steepest change of the objective function and step in that direction. This is done iteratively until certain convergence criteria are reached. Local search methods, following some sort of steepest descent direction of search from a starting point, often result in finding one of the minima near the search starting point (Figure 2-12).

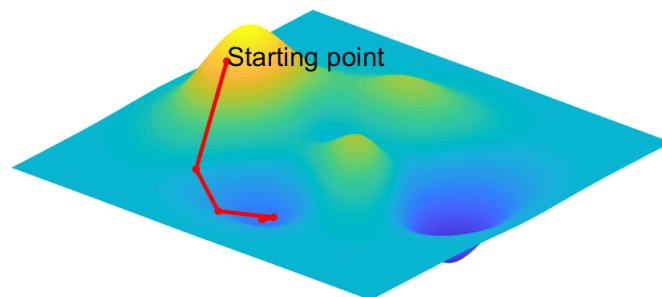


Figure 2-12. Local search iteratively following steepest descent to local minimum near starting point.

Global search methods, on the other hand, are designed to seek the global optimum design point in a way that is somewhat independent of the starting conditions. One of the simplest approaches to global search is to execute local search at multiple (often random) starting points. There is a correlation between local and deterministic search methods, and likewise a correlation between global and stochastic search methods such as evolutionary algorithms or simulated annealing. References such as Vanderplaats (2007), Rao (2009), and Parkinson et al. (2018) provide more in-depth descriptions of the various common single-objective optimization algorithms.

Instead of finding a local minimum near a starting point or a single global optimum, sometimes it is desirable to instead find multiple local minima for multimodal functions. These are objective functions that may have several peaks and valleys in the design space and therefore several local minima that may be interesting and valid design solutions. Because of the similarity to a global search task, multimodal search algorithms are often variations on global search algorithms such as genetic algorithms and other evolutionary algorithms (Wong, 2015).

2.4.2 Multi-objective search

It is often the case in real-world engineering development, especially in early aircraft design, that there will be several competing interests and therefore compromises to be made. In many cases, a single-objective optimization formulation with constraints will not adequately represent the design challenges at hand. In these situations, a formulation is needed that seeks to simultaneously minimize multiple objective functions,

$$f_1(x), f_2(x), \dots, f_k(x) . \quad (6)$$

The optimization task therefore becomes

$$\min_x F(x) , \quad (7)$$

where $F(x)$ is the vector of all objective functions, and the optimization is still subject to the same side, inequality, and equality constraints as with single-objective optimization.

Multi-objective optimization and other multi-criteria decision methods can support the designer in several ways: Many approaches select a single design point, some rank various alternatives, others use clustering to classify alternatives and sort them into groups, and others compare and prioritize between two or more alternatives. These method roles are summarized in Figure 2-13.

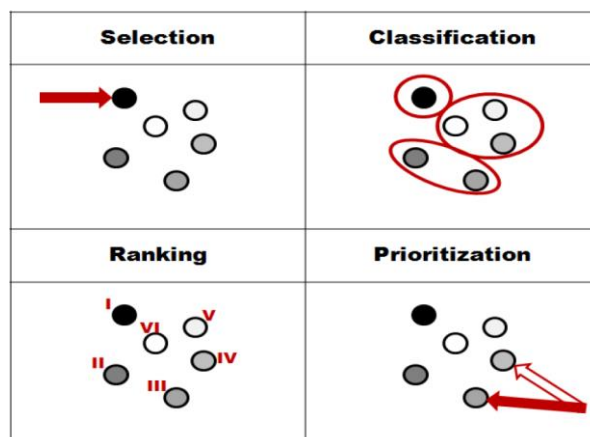


Figure 2-13. Various roles of multi-objective methods. (Schneiderbauer, 2013, p. 3)

It is rarely the case that a single design point, x , can simultaneously minimize all objective functions, and compromises must therefore be made by trading off

one objective for another. If an objective cannot be improved from a given design point without a detriment to another objective, then that design point is considered Pareto optimal. It is therefore desirable for multi-objective methods to yield Pareto optimal results. *A priori* methods typically require additional inputs prior to using the algorithm with the aim of identifying and selecting a single optimum design point. *A posteriori* methods, on the other hand, more typically focus on identifying a full Pareto set, and after the algorithm is run is when the designer or decision maker will integrate further information to make a design selection from this set. See Loehr (2013) and Schneiderbauer (2013) for further discussion of more multi-criteria and multi-objective search and decision-making methods and their taxonomies.

It is common with multi-objective optimization to create a single objective function that is itself some function of the multiple objectives. This normally involves creating a single all-encompassing value function to represent overall goodness of a design as a single figure of merit. One of the most common of these value function methods is the simple weighted sum model, whereby, as the name suggests, weights, w , are assigned to each objective and the sum of the results is the new single objective function to minimize, as in

$$\mathbf{x}^* = \min_x \sum_{i=1}^k w_i f_i(\mathbf{x}). \quad (8)$$

This and similar methods, such as the less common weighted product method,

$$\mathbf{x}^* = \min_x \prod_{i=1}^k (f_i(\mathbf{x}))^{w_i}, \quad (9)$$

require significant and precise *a priori* knowledge of design preferences, which is rarely available in early design. There are also mathematical issues that can sometimes make compromise solutions unlikely to be found in cases of simple formulations of value functions.

Lexicographic ordering, another *a priori* method, requires the designer to rank objectives in order of absolute importance. Often in real-world cases, the method results in ignoring all objectives except for the most important one. The method also requires foreknowledge of the absolute importance of objectives, which is not always the case in early design.

Methods such as Nash arbitration, goal programming, achievement scalarizing, and similar methods integrate more specific knowledge and preferences regarding objectives, but only in the form of a single goal or reference value for each objective. This can result in ignoring any benefits of exceeding goal values and/or effectively over- or under-weighting objectives based on the distance from the reference value.

The ELECTRE (*Elimination Et Choix Traduisant la REalité* / Elimination and Choice Expressing Reality) family of multi-criteria decision methods is more of a decision analysis method than a multi-objective optimization. However, it does facilitate selection, ranking, or sorting (depending on which ELECTRE method)

of a set of alternatives based multiple factors. Some ELECTRE methods use multiple thresholds (e.g. indifference, preference, and veto thresholds) for each criterion to mimic real decision-maker preferences. Another feature of the method is that it allows for non-binary comparison of alternatives (two alternatives can be “just as good” or “incomparable” in addition to simply “better” or “worse”), which accounts for common real-world decision-maker preference. (Schneiderbauer, 2013)

2.4.3 Physical programming

Physical programming, introduced by Messac (1996), along with applications in aircraft design (Messac & Hattis, 1996), is the multi-objective optimization method that carries most similarities to the new technique presented in this work for capturing and using preference information in early design. Physical programming captures richer preference information for each objective, with four different classes of information that could be associated with a given objective and an associated ‘hard’ or ‘soft’ mapping of preferences to the objectives. Figure 2-14 shows the classification of preference, \bar{g}_i , to each design objective, g_i .

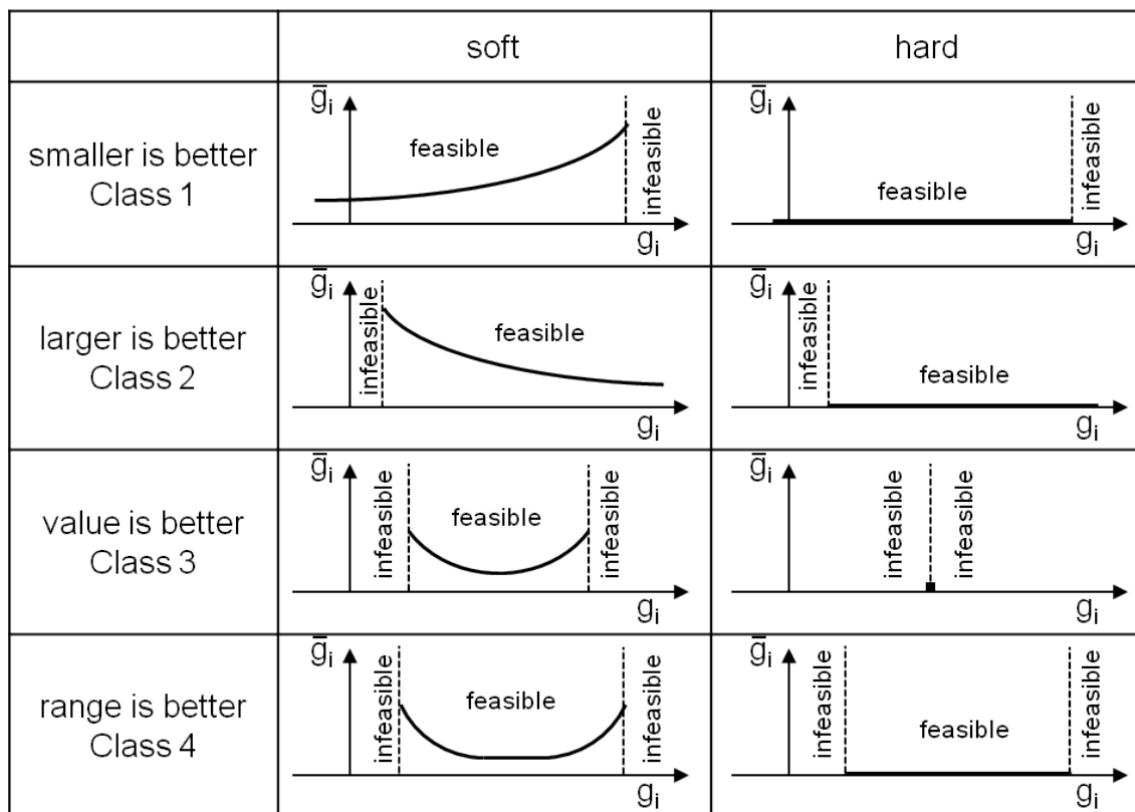


Figure 2-14. Physical programming preference classification. (Loehr, 2013, p. 20)

One of the weak points of several multi-objective methods is the need to adjust relative weights or goals for each objective. Physical programming addresses this by mapping distinct qualitative attributes (highly desirable, desirable, tolerable, undesirable, highly undesirable, and unacceptable), which can be applied somewhat consistently across objectives, to various values of each objective, shown for the soft classifications in Figure 2-15. Furthermore,

application of physical programming techniques has shown to be effective in both robust design and in the generation of full and well-distributed Pareto optimal solutions (Chen et al., 2000; Messac & Mattson, 2002).

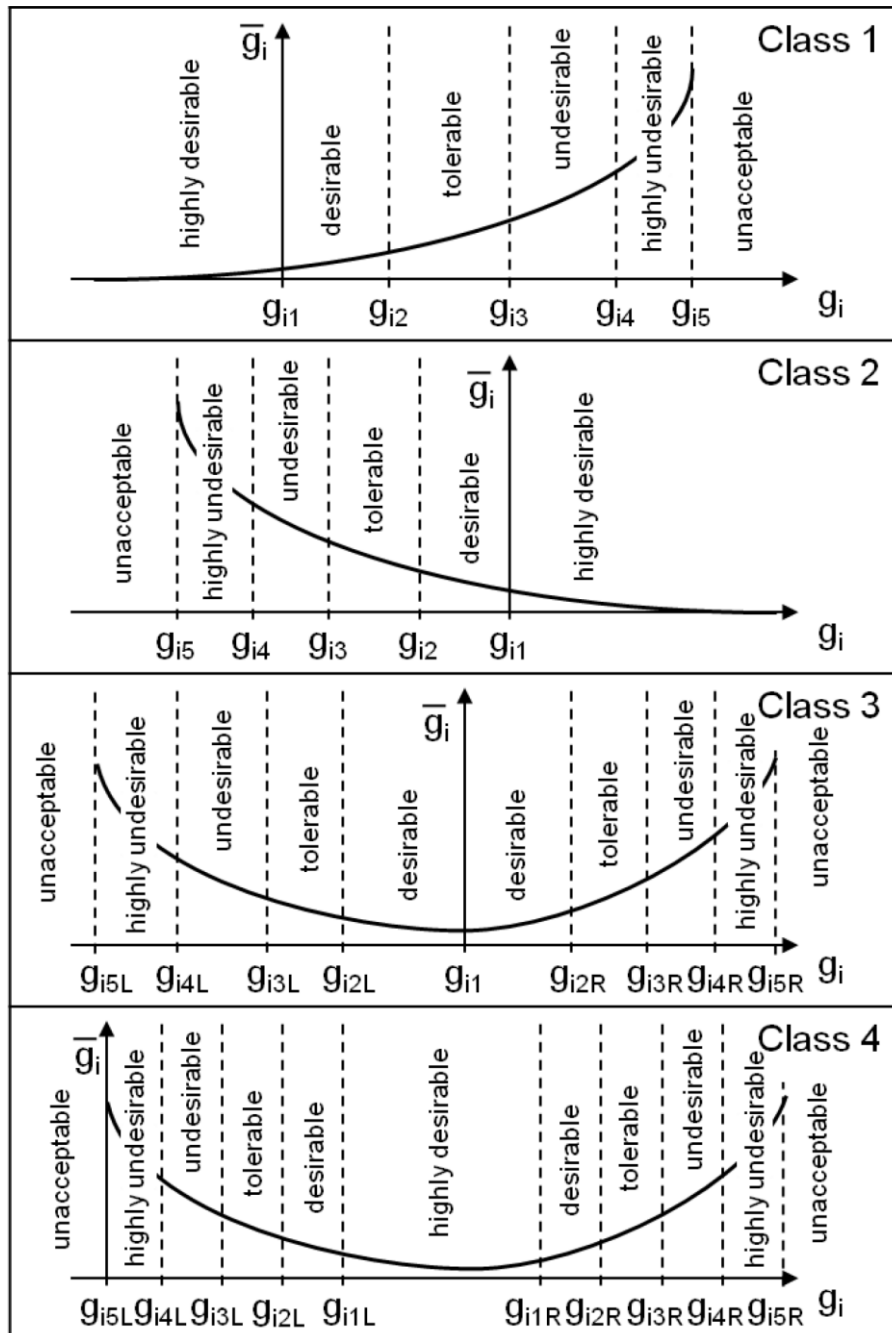


Figure 2-15. Physical programming class function ranges. (Loehr, 2013, p. 21)

Variations of physical programming have been explored. The physical programming described above is sometimes called nonlinear physical programming to distinguish it from the more common linear physical programming whereby instead of using a type of smooth spline function, a simple linear piecewise function is used for the class functions, for example as shown in Figure 2-16.

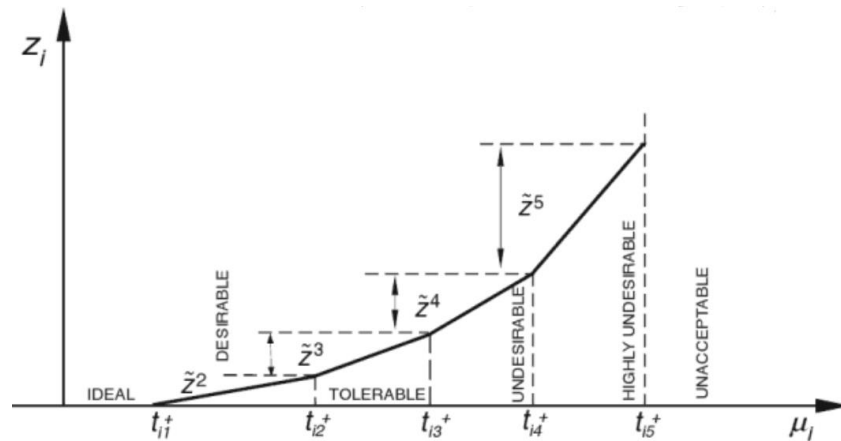


Figure 2-16. Linear physical programming class function. (Messac, 2015, p. 431)

There are other variations as well, such as fuzzy and global physical programming, though the majority of applications of physical programming use linear physical programming, and the plurality of applications are in the field of engineering design (Ilgin & Gupta, 2012). The exercise presented by Ilgin (2019) is a recent example of employing linear physical programming in aerospace for selecting an aircraft for an airline fleet.

In the work of Yatsuka et al. (2018), the linear physical programming concept of a preference class function is used as a way to non-linearize a goal programming multi-objective optimization problem. The authors then used this approach to capture preference functions for multiple stakeholders with competing interests in what they call multi-player multi-objective decision making. This creates a method whereby *a priori* information from the stakeholders is used to automate arriving at compromise solutions. Furthermore, robust optimization was applied to capture uncertainty information to allow minimization of the worst case under uncertainty, mitigating effects of biases.

2.5 Comparing optimization to other search techniques

The design space exploration and decision techniques discussed in Section 2.2 and particularly Section 2.2.3 – the so-called ‘classical’ design space search and decision methods – have been practiced since before design and engineering with modern computing speed and power was possible. Tools for this classical approach include the decision matrix, house of quality, two-, three-, and four-variable plots, etc. These classical techniques essentially encompass the identification, analysis, communication and understanding, and decision phases of the archetypal aircraft design iteration process discussed in Section 1.2.1. Done well, documentation is also done simultaneously as part of the process.

The essential advantage of these classical approaches is that the designer and other stakeholders retain the freedom to intervene and deviate from the mathematical models if deemed necessary. This means that the designer has the option to challenge design requirements, compensate for deficiencies in analysis tools and methods, and account for a value function or figure of merit that does not capture all value information or preferences. The designer can also use experience to arbitrarily add margins to the process as needed.

The disadvantages of the classical approaches derive from the inability to simultaneously examine a trade space with many dimensions. Being unable to examine all relevant parameters simultaneously means that it is difficult or impossible to find the absolute (mathematically) best alternative. As the complexity of the problem increases, the advantages of the classical approaches further diminish to the point that establishing a full understanding of the design space becomes difficult to the extent that an optimal or Pareto optimal solution is not guaranteed, even for smooth, differentiable, convex problems.

Optimization methods, discussed in Section 2.4, by leveraging automated analysis tools to arrive at an optimal solution, address the analysis phase and the decision phase of the archetypal aircraft design iteration process discussed in Section 1.2.1. Unfortunately, this means that the communication and understanding phase is mostly or entirely omitted when implementing optimization methods.

The main advantage of optimization methods, and the main reason that they are currently used effectively in aerospace development and other fields, is that it is likely that an optimal design solution can be found that is superior in every way to the best solution findable with the ‘classical’ methods alone. However, this feature comes at a cost. With optimization methods, many of the advantages of the classical approach are lost, including the ability to challenge requirements, add margin, account for analysis tool deficiencies, and integrate more subtle preference information. Extra effort must be exerted as well in order to formulate the optimization problem in a way that the analyses and the search algorithm are compatible. In addition, the designer must have some level of optimization expertise in addition to discipline-specific expertise to effectively wield optimization as a tool and avoid potential pitfalls such as those caused by noisy data, discontinuities, etc.

A rough qualitative comparison between the classical approach and the optimization methods are shown below in Table 2-2, with optimization's strength lying in the attributes at the top of the table and the classical approach's strengths falling primarily in the attributes at the bottom of the table.

Table 2-2. Attributes of alternative approaches to design space search.

Approach attribute	Approach	
	Classical	Optimization
Examine many parameters simultaneously	●	●
Find feasible domain	●	●
Find precisely optimal solution(s)	●	●
Understand problem and design space	●	●
Account for known modeling deficiencies	●	●
Identify design drivers	●	●
Challenge requirements	●	●
Deviate from established figures of merit	●	●
Account for extra preference information	●	●

Key: Strong ● ● ● Weak

The newly developed approach and supporting tools described in this work strive to retain most of the advantages of the classical approach while being able to use numerical optimization and other techniques requiring intense computation to find good design solutions. The approach allows for finding optimal solutions while integrating more preference information in a methodical, justifiable fashion but without the need to formulate an elaborate and precisely informed value function or objective weightings. It thereby helps achieve greater understanding of the problem, account for known or discovered analysis deficiencies, and retain the ability to challenge requirements and take into consideration the cost of requirements. The method semi-quantitatively captures preference information that, for technical or other reasons, cannot be integrated into an all-encompassing value function. The lack of said value function is accounted for by keeping the designer in the loop.

3 Methodology

The two primary objectives of this work are to a) formally capture stakeholder knowledge and b) put that knowledge to good use in the design and development cycle. For each of these objectives, both in isolation and in conjunction, there are multiple possibilities for how to approach the execution. This chapter describes the specific method used for this work, called the Well-Informed Search Design Optimization Method (WISDOM), based primarily on utilizing preference maps coupled with numerical optimization. The WISDOM approach described in this chapter is relatively independent from the specific software implementation of the approach.

3.1 Applicability of approach

It is important to first discuss the conditions under which this approach is and is not appropriate to apply as intended.

3.1.1 Early design studies

This work is primarily developed with early design studies in mind. As discussed in Section 1.3.4, the term 'early design' as used here refers to the early stages of a trade study or exploration of the design space. This could typically be the first sizing estimate made in the first days after either encountering a new set of air vehicle requirements or a decision to pursue internal development of a new product. More broadly, however, an early design study is simply the initial investigation into some vehicle or system aspect that has not been previously given significant attention in the overall development effort.

3.1.2 Moderate dimensionality

Early aircraft design studies involve multiple degrees of freedom. However, because it is early design, the total number of design variables that are truly of interest is usually limited. Very early airplane design, for example, may only be interested in two: powerplant size and wing size (thrust-to-weight ratio and wing loading, e.g.). Even as conceptual design progresses quite far, there are still only six basic design variables that are the most important in conceptual design (Raymer, 2002), with the wing geometry being the main focus:

Table 3-1. The basic six design variables of airplane conceptual design.

Design variable	Description
T/W or P/W	Engine size
W/S	Wing loading
AR	Wing aspect ratio
λ	Wing taper ratio
Λ	Wing sweep
t/c	Wing airfoil thickness ratio

Unlike airplanes, for rotorcraft there is typically no phase early enough where only examining two main design variables is appropriate. For traditional single main rotor, single tail rotor helicopters, instead of the dominant two parameters for an airplane, it may be appropriate for the very earliest investigations for the helicopter to examine a minimum of four parameters relating to rotor disc area, rotor speed, rotor solidity (the reference area of the blades in relation to the disc area), and the engine size. The typical analogue to the most important conceptual design parameters for airplanes may also include the number of blades and the twist of the rotor blades. Finally, for a compound helicopter, the typical set of conceptual design parameters also includes the lift-compounding wing area and aspect ratio and the thrust-compounding forward propulsion system size (Sartorius, 2011a), for a total of seven to nine design variables that would be in the earliest design studies.

Since most early design studies for aircraft will likely only examine perhaps two to six design variables simultaneously, the approach here is intended only to be used for design studies with up to about a dozen design variables (plus about

twice that many constraints, measures of merit, and other parameters of interest). Problems with higher dimensionality are not necessarily infeasible with the approach, but they are not the intended application.

3.1.3 Mostly continuous design variables

In early design of air vehicles, massive discontinuities in the design space are rare and are usually small in comparison to the scale of the overall design space. One cause of this is that there are not too many discrete parameters in early design. Discrete design variables that do come up are often a countable number that is quite small, with just a small and manageable number of distinct alternatives, for example a design variable for the number of engines or number of rotor blades. For early design studies, many other discrete parameters can be effectively 'smeared' into a continuous parameter for purposes of early design iterations, for example the number of cells in a battery pack for electric propulsion. So, while the approach should be able to handle some discrete variables, it is more as an exception and not as the norm.

3.1.4 Nominally convex design space

Like enormous discontinuities, a design space with multiple major 'humps,' i.e. qualitatively multiple highly disparate but valid and viable local optima, are rare in the design experience of this author. Non-convex and non-smooth design spaces can often be present at a much smaller scale, however. An example source of this is the prevalence of analytic methods that rely on discretization (for example for piecewise integration), methods that have limited precision (for example limited significant figures from the output of a separate integrated tool that the design has no control over), and/or methods that rely on iteration to find solutions for analyses that are difficult or impossible to invert into a closed-form equation. An example of this type of 'micro' non-convexity is shown in Figure 3-1 for a case where gross weight of an aircraft is solved as a function of fuel fraction using an uninvertible equation for empty weight fraction as a function of gross weight. At the macro level, the method appears smooth and continuous but in fact, due to the iterative method of solving, has significant nonconvexity and discontinuity at the 'micro' scale. This type of 'noise' must be kept in mind as an expected and likely phenomenon to encounter when applying the design space exploration approach discussed in this work. However, because the approach uses off-the-shelf existing optimization tools and algorithms that may have trouble handling this kind of 'micro' noise, the approach, just like most applications of MDO, is expected to perform better when the analytical tools used are truly smooth and continuous.

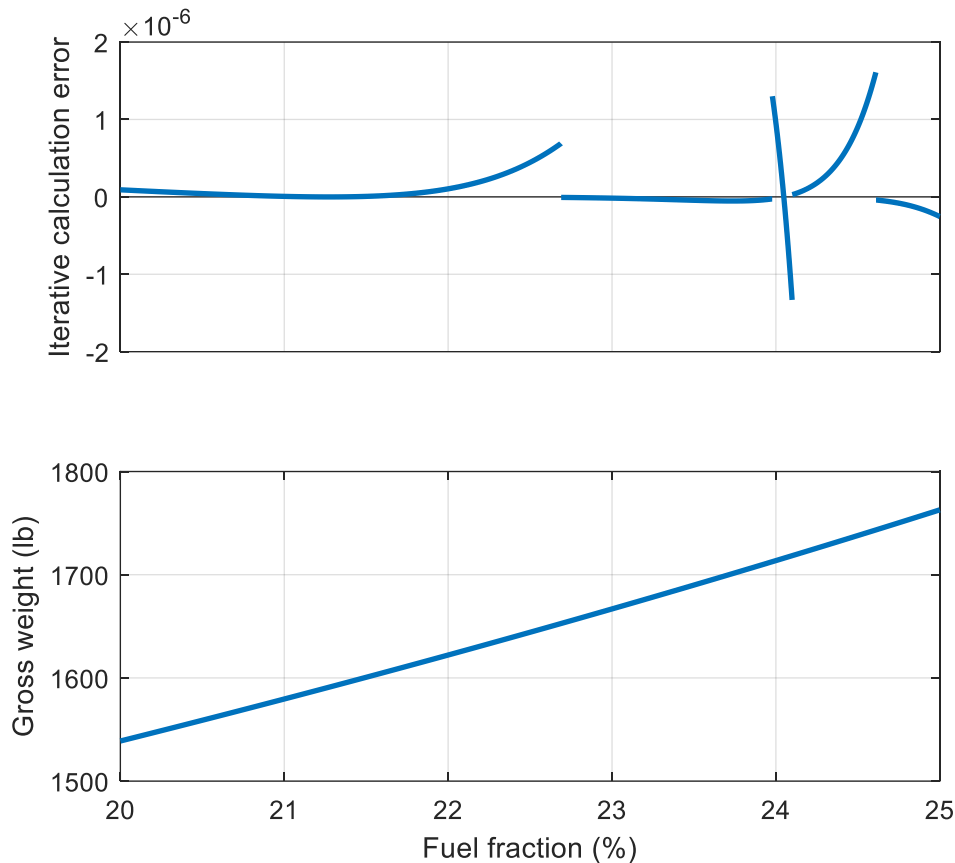


Figure 3-1. Example of 'micro' non-convexity typical of iterative methods used in early aircraft design.

3.1.5 Tightly coupled parameters

A key element of the method is capturing of stakeholder knowledge. One aspect that the approach does not *explicitly* account for is tightly coupled parameters. While it is generally the case, especially in aircraft design, that very few parameters are truly independent, the WISDOM approach operates under the assumption that most parameters are independent, with mechanisms in place to compensate for the inevitable but assumed small coupling of parameters. Therefore, it is primarily left to the designer to recognize tightly coupled parameters and compensate for these themselves. A simple example is that the designer should be cognizant that the empty weight of an aircraft and its gross weight are tightly correlated and that assigning similar strong preferences to both is effectively double bookkeeping of the true underlying preference. A more advanced case may be where two technical assumptions are treated independently but are in fact correlated, for example aerodynamic efficiency and structural efficiency. In this situation, a designer would be better off transforming this to be an assumption applying only to one of the technology levels and roughly modeling the other as a dependent function of the first, possibly introducing an alternate assumption regarding the strength of the correlation. In summary, the applicability is limited to cases where the coupling of the parameters is understood well enough by the designer to be able to sufficiently compensate for it manually in the setup of the problem.

3.2 Issues with optimization in early design

Besides the tendency in early design to have non-convexity, discontinuities, etc., there are some additional characteristics of early design studies which pose a challenge to leveraging traditional optimization in the process.

3.2.1 Imperfectly defined requirements and objectives

The first of these characteristics is that the requirements and objectives for the project are unlikely to be precisely defined and well understood, and they may in fact be in a state of flux as the early design efforts are under way. This is especially true for product design as opposed to, for example, a traditional defense program where the objectives and metric(s) to maximize or minimize may be explicitly stated. Even then, the explicit objectives are only a best effort by the requirements' author(s), and a faithful, literal adherence to those objectives may not result in fulfilling the true underlying needs or satisfying all the ultimate decision-makers.

3.2.2 Low-fidelity analytical models

Conventional wisdom regarding the analytical models used in optimization is captured well by Vanderplaats (2007): "The underlying analysis must properly model the true physics or optimization will generate unrealistic designs." In addition to uncertain requirements, another common trait of early design studies is a lack of maturity or fidelity of the analytical models employed. Models appropriate for conceptual aircraft design, by definition, do not capture the effects of all possible design parameters simply because these parameters are unknown at the conceptual design stage. One positive side benefit is that these models used in early design tend to be relatively simple and computationally inexpensive, easing what can be one of the pain points in implementing optimization search algorithms: long computation times.

3.2.3 Immature system models

Models may also lack maturity because it is possible in early design that the models are being built in parallel to the design effort itself. When unique design problems or solutions are involved (or problems or solutions novel to the particular organization in question), it may be a necessity to develop novel models at the same time as the design definition is maturing. Compared to established models that have been used before, under-construction models are less tested and validated, and the users have less experience with them, so the models have a higher chance of leading the searches astray through unaccounted for responses to certain combinations of inputs.

3.2.4 Rigidity of optimization approach

Even if the deficiencies of modeling are known and acknowledged, system models are still unsuitable for use with automated search due to the inability to integrate more subtle types of information into optimization in the early design stages. Subtle preference information and knowledge is usually very active in early design as an efficient shortcut to maturing the design, the requirements, or both. Implementing optimization, where the design decisions are being made

behind the obscuring curtain of the algorithm, strips away some of the ability to use that information for making quicker or better decisions or, in some cases, challenge design requirements based on gained insights from more manually exploring the design space using more classical design space exploration and decision techniques.

3.2.5 Algorithms focused on final solution

A final aspect of early design studies posing a challenge to using optimization is that the fundamental aims of early design projects are not well aligned with the goals of more typical optimization. Mathematical optimization is focused on finding optima, and, in most cases, this is an optimal design solution according to a single all-encompassing objective function that attempts to capture all facets of value and desires for all stakeholders.

In early design studies, in contrast to focusing on finding the single best design point, the designer is more likely concerned with identifying 'interesting' regions of the design space for investigation in future iterations and informing human-in-the-loop decision making. One major reason for this is that any single design point chosen in early design is certain to change. Especially when working on a novel design problem or solution, the designer may also simply be more focused during this stage on building and maturing the analytical models. Another reason that early design is not as focused on selecting a single design point is that the designer is often more interested in increasing understanding of the design problem as opposed to allowing a search algorithm explore the design space as a black box operating behind a veil of abstraction and obscurity.

3.3 Types of information captured

The method here is designed to facilitate capturing types of stakeholder knowledge, wisdom, and preferences that normally are un- or under-captured, especially *a priori*. The three main categories of information type are assumption uncertainty, preferences on requirements and other parameters, and known modeling idiosyncrasies and deficiencies.

3.3.1 Example regional airliner design for illustration

To illustrate all three of these categories, a textbook early aircraft fuel fraction sizing example is used. For this example, a hypothetical company that already has a product portfolio of business jets and small regional jets is interested in developing a larger aircraft: a small regional airliner. The example is based on the initial sizing of such a vehicle to the requirements in Table 3-2 below using the fuel fraction sizing method, with the gross weight of the vehicle being used, as is common practice in aircraft design, as an unrefined though meaningful surrogate for higher-level figures of merit such as operating costs.

Table 3-2. Example design case range and payload requirements.

Requirement	Value
Range	2500 nmi (4630 km)
Payload	100 passengers + flight deck and cabin crew

The fuel fraction sizing method is named for how a given mission profile is analyzed to find the weight fraction for each mission segment flown, i.e., what portion of the starting weight for that segment is expended as fuel. Typically, in conceptual design, several segments are assigned a fixed, assumed weight fraction such as for taxi, takeoff, and even climb, while for calculating weight fractions of cruising or loitering mission segments, various forms of the Breguet range equation are used, for example

$$\frac{W_{end}}{W_{start}} = e^{\frac{-R \cdot SFC}{V \cdot L/D}} \text{ or } \frac{W_{end}}{W_{start}} = e^{\frac{-E \cdot SFC}{L/D}}, \quad (10)$$

where endurance, E , range, R , specific fuel consumption, SFC , speed, V , and aerodynamic efficiency, L/D , are estimated or assumed with a level of fidelity appropriate for early conceptual design.

The product of all the weight fractions for all the mission segments reveals how much fuel is consumed through the mission as a portion of the starting gross weight of the aircraft. This fuel fraction (usually with some margin added), combined with the empty weight fraction and the known payload weight, yields the gross weight of the aircraft.

If an estimated fixed constant value is used for the empty weight fraction, then this is where the fuel fraction sizing method ends. However, the empty weight fraction is commonly expressed as a function of gross weight based on a historical regression, usually of the form

$$\frac{W_{empty}}{W_0} = A \cdot W_0^C, \quad (11)$$

where the constants A and C are taken based on a fit for a given category or class of aircraft. For the example here, constants (for W_0 in units of pounds) $A = 0.902$ and $C = -0.0385$ are used to be representative for the small jet transport category, resulting in the relationship shown in Figure 3-2.

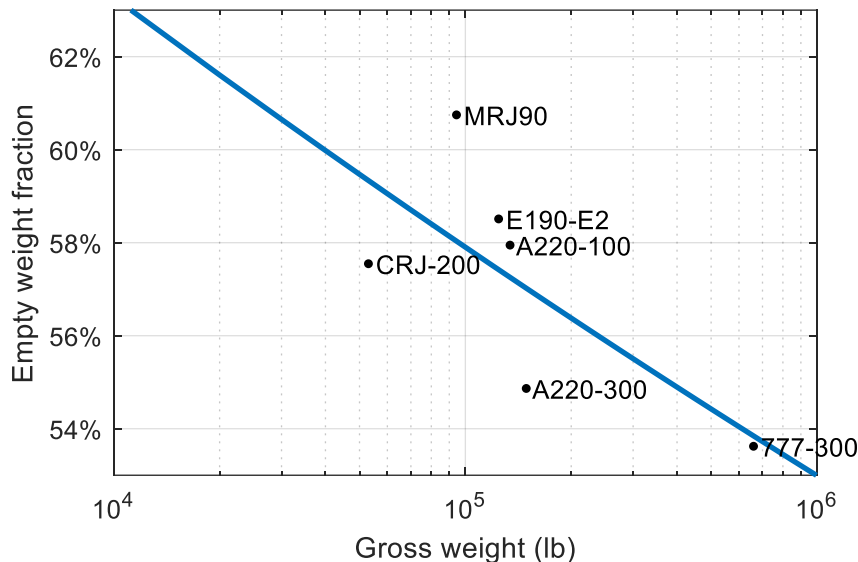


Figure 3-2. Regional jet transport empty weight fraction trend.

3.3.2 Assumption uncertainty

Using this simple example case of the regional airliner, we can illustrate how information regarding technical assumptions is often uncaptured in early design. Note how the weight fractions calculated using the Breguet range equation, Equation (10), are highly affected by the assumptions used in early conceptual design for specific fuel consumption and aerodynamic efficiency. It would be the normal situation for these assumption parameters to be set to singular, fixed values at the beginning of some analysis script (Figure 3-3), which for the regional airliner example here are set as $L/D = 17$ and $SFC = 0.5 \text{ lb/hr/lb}$.

```
maxLiftToDrag = 17; % L/D
sfc = 0.5; % Thrust specific fuel consumption (lb/hr/lb)
```

Figure 3-3. Assumptions set to fixed values early in analysis.

These assumptions could very well be set spontaneously by an experienced designer or could be somewhat informed by some preliminary analysis. In either case, there is significant information that is not being captured here.

With the baseline requirements and assumptions, the resulting gross weight of the sized aircraft is 124,000 pounds (56.1 tonnes). The designer knows at this point that this is only an estimate based on the best assumptions available at the time, and that there is in fact significant possible variation in this result. Whenever this result is presented, it must be accompanied by a “depending on the assumptions” disclaimer.

If the designer can come up with a best estimate baseline value for a given technical assumption, he or she should also, at a minimum, have a reasonable guess for what a worst-case and best-case value for the parameter might be. However, what if instead of just a singular value for an assumption, a probability distribution is captured? Creating such a distribution can be as straightforward as a simple triangular distribution (Figure 3-4), which only requires the three pieces of information that are likely to be readily at hand:

- Best and most likely estimate
- Most optimistic possible estimate
- Most pessimistic possible estimate

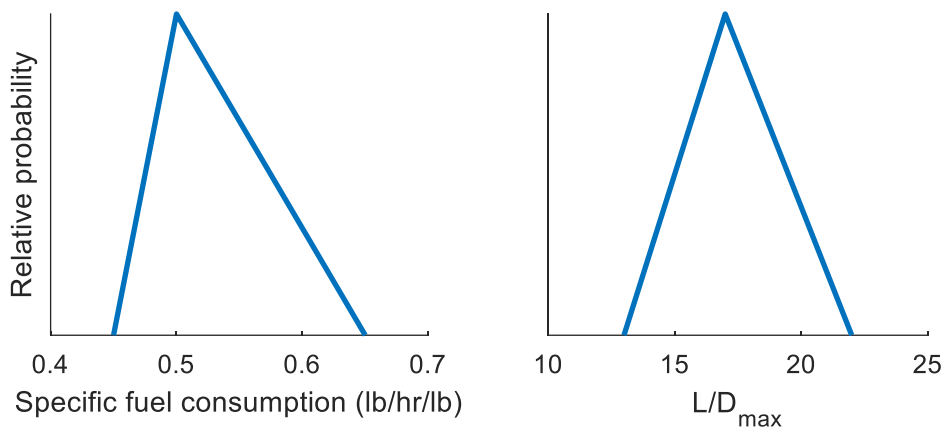


Figure 3-4. Triangular distributions for uncertain technical assumptions.

Now, significantly more information that was already available for the designer is formally captured. With this range of possible values for the technical assumptions, there is now a range of possibilities for the resulting gross weight, with a most-optimistic gross weight of 95,700 pounds (43.4 tonnes) and a most-pessimistic gross weight of 284,000 pounds (129 tonnes). In fact, a distribution of probabilities exists across that range, which is easily obtained by a simple Monte Carlo simulation approach, as shown by the histogram of the results in Figure 3-5 below (result with baseline assumptions shown in red).

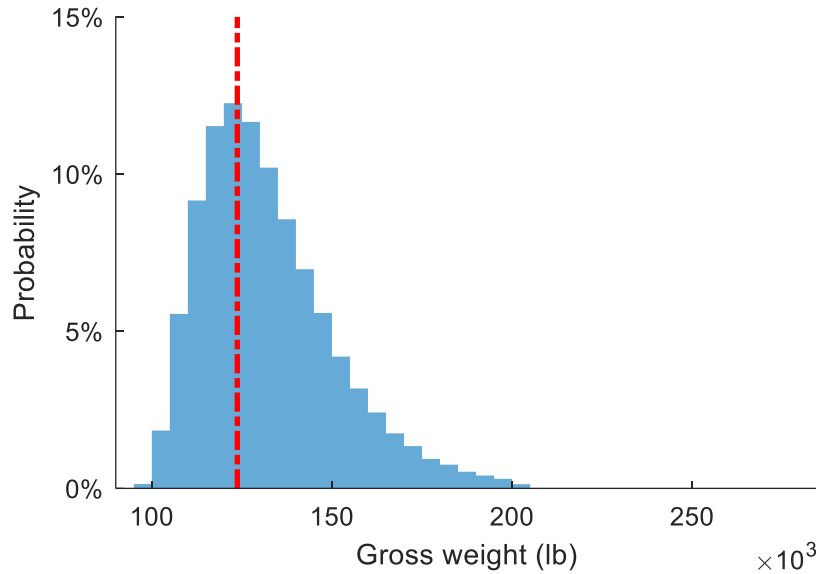


Figure 3-5. Resulting gross weight estimate distribution, propagating uncertainty.

Visualizing the distribution (with a histogram or with some other method such as a whisker plot) is crucial to gaining a better understanding of the effects of the assumptions beyond the basic understanding that comes from simply examining the optimistic and pessimistic cases. In this situation, the pessimistic case is over double the gross weight of the baseline, but when examining the histogram, it is apparent that the pessimistic extreme is *very* extreme and highly unlikely. This contrasts with results in the region of the optimistic case, which are much more realistic to consider.

A triangular distribution makes sense when the only information on hand about a parameter's assumption is the minimum realistic value, the maximum realistic value, and a best guess. However, with that same set of information, slightly more realistic distributions can be used without significant added effort. Taken to the extreme, a technical assumption's uncertainty can be captured in the form of any arbitrary probability density function (PDF) that the designer or stakeholder might choose or may even be informed by experimentation involving many samples.

When picking a probability density function for capturing technical assumption uncertainty in early design, it is important in some cases when generating random values of the parameter for analysis that hard bounds on the parameter's value are possible so that values that simply do not make physical sense can be precluded. A normal (Gaussian) distribution does not fulfill this requirement, but one flexible distribution that is versatile for many situations (while also being free of the discontinuity of the triangular distribution) is the beta distribution, which has the PDF on the x interval from zero to one of

$$f(x) = K \cdot x^{\alpha-1}(1-x)^{\beta-1}, \quad (12)$$

where the two shape parameters, α and β , determine the shape of the distribution. The normalization constant K is equal to the reciprocal of the beta

function of the shape parameters α and β , which ensures that the total probability is one.

The beta distribution is most useful for capturing assumptions when modified to a PERT (Program Evaluation and Review Technique) beta distribution (London, 2013; *The Beta-PERT Distribution*, n.d.) such that the function is again characterized by a minimum value, maximum value, and best estimate (the mode of the distribution). While the beta distribution needs two parameters, α and β , to determine its shape, the PERT distribution only requires a single parameter for the best estimate. That is because the PERT distribution also allows one to capture an additional shape parameter defining how ‘peaky’ the distribution is and reflecting how confident one is in the best guess estimate. With the PERT distribution, the shape parameters of Equation (12) are defined in terms of the mode, m , and this more meaningful shape parameter, λ , as

$$\beta = \frac{\alpha(1 - \mu)}{\mu}, \quad (13)$$

and

$$\alpha = \frac{\mu(2m - 1)}{(m - \mu)}, \quad (14)$$

where

$$\mu = \frac{(\lambda m + 1)}{(\lambda + 2)}. \quad (15)$$

For comparison to the above example, Figure 3-6 and Figure 3-7 represent assumptions and results, respectively, when using a PERT distribution in lieu of a triangular distribution. The result is qualitatively the same, though with slightly more results toward a lower gross weight and somewhat more tightly clustered.

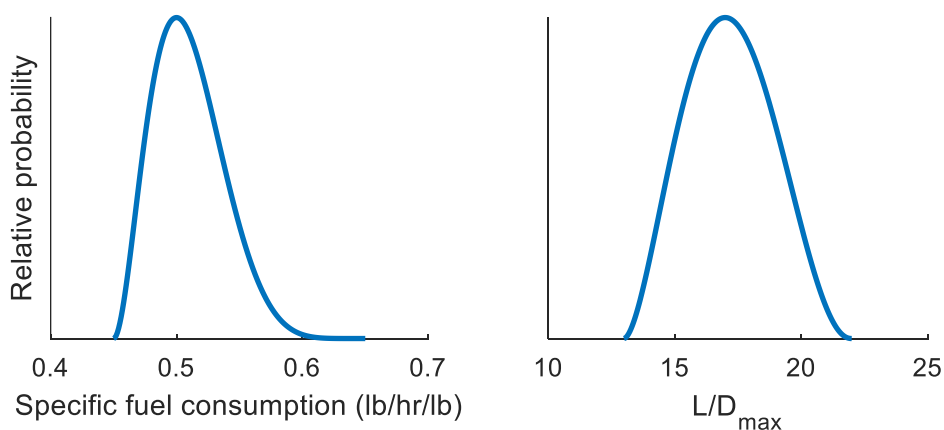


Figure 3-6. PERT distributions for uncertain technical assumptions.

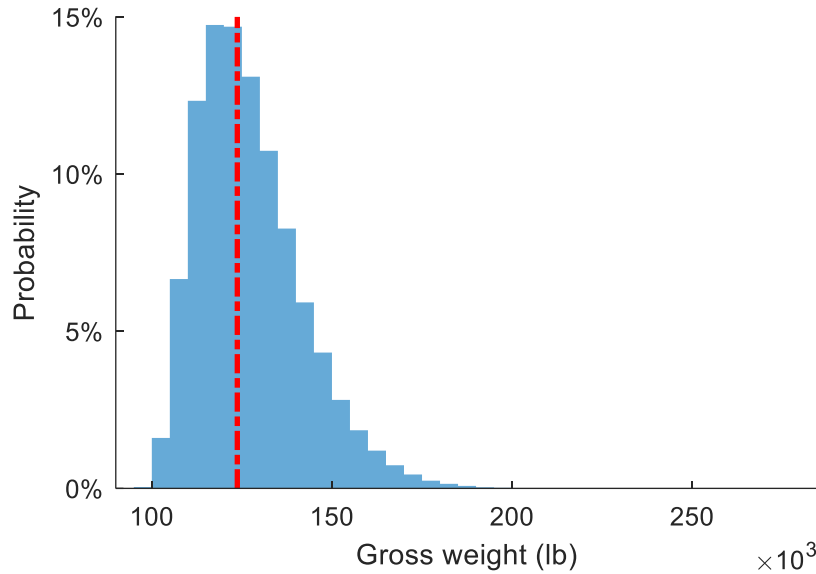


Figure 3-7. Resulting distribution when using PERT-distributed assumptions instead of triangular.

The ability to capture technical assumptions not just as a singular value, but as a probability density function, is an important feature of the method. With that additional information in hand, the uncertainty about assumptions can be propagated to final analysis results, for example through examining the distribution of results of many Monte Carlo trials as shown in the example here. This yields significant additional insight into the design possibilities early in design and without requiring significant effort beyond inputting simple information that is already on hand for the designer.

3.3.3 Preferences on figures of merit and design parameters

Another category of information captured by the technique is preferences on figures of merit, design requirements and specifications, and other design parameters. Different projects in different aerospace sectors may have varying degrees of solidity of definition of certain metrics, as experienced by this author. A small general aviation aircraft may be an example of one end of this spectrum, as these may be purchased by pilots or passengers for whom the financial costs of aviation are of secondary importance to aesthetic or other emotional factors. At the other end of this spectrum may be an airliner product, where there is well-understood existing business model along with a small set of clear, overarching value functions to minimize or maximize (costs and profits). In addition, there is often a connection with customers to support market research with airlines, as well as commitments from customers to further enhance confidence in the requirements for a new design. Across this entire spectrum, however, there can be un- and under-captured stakeholder wisdom in the form of preferences on various requirements, design parameters, and figures of merit.

For the regional airliner example, consider the passenger capacity requirement from Table 3-2 in Section 3.3.1. Like the earlier motivational example (Section 1.3.2 Figure 1-4), a fixed value for the payload requirement does not fully

capture the entirety of the knowledge available regarding the desired passenger capacity of the to-be-developed new regional airliner. Even if an extremely refined cost and profits model exists, blindly minimizing or maximizing to these parameters, respectively, still may omit some key information from the decision-making process.

The one hundred passenger capacity is the specified requirement, which could have been dictated by careful market research or just a rough initial guess. In either case, there are some pieces of information that a designer would still use when examining the whole spectrum of possible passenger capacities to design to. Some examples of these extra pieces of information take the form of qualitative preferences, such as:

- A. Other things being equal, it is good to comply with the specified requirement of a one hundred passenger capacity, avoiding challenging requirements and conflicting with what was specifically requested for the new product. Given the roughness of any results early in design, any result that is close to what is asked (within, say, five passengers) might as well simply default to the specified requirement and avoid the hassle of challenging a requirement.
- B. It is slightly preferable to develop a smaller aircraft if possible, given that it would require a smaller overall investment of time and resources to achieve the goal of bringing a new airplane to market. It is also closer to the experience base of the company, with increased chances of recycling design aspects and expertise.
- C. In this hypothetical scenario, the manufacturer already has existing products with a smaller capacity of fifty passengers, and there is therefore a strong preference not to develop a small aircraft that will impinge on sales of existing aircraft, even if the new product could be potentially very profitable by itself.
- D. In line with preference C above, there is a hard lower limit on passenger capacity at around fifty passengers, where there would essentially be direct duplication of an existing product in the portfolio.
- E. Like the internal competition on the small side, there is also significant competition on the larger side of the range of possibilities, though with other companies and not within the hypothetical firm, in the form of the prolific existing narrow-body airliners in the ca. 150-seat capacity range. Therefore, there is a preference for making sure that whatever new aircraft is developed will be in a capacity category far away from those competitors.

If the passenger capacity is plotted against a meaningful metric or figure of merit (such as in Figure 3-8), and then some decision-makers decide on a specification based on this plot, then they can integrate the preference information that they carry in their minds at that time. However, it is an objective of this work to automate such decisions while keeping them just as informed by that preference information in order to yield similar results.

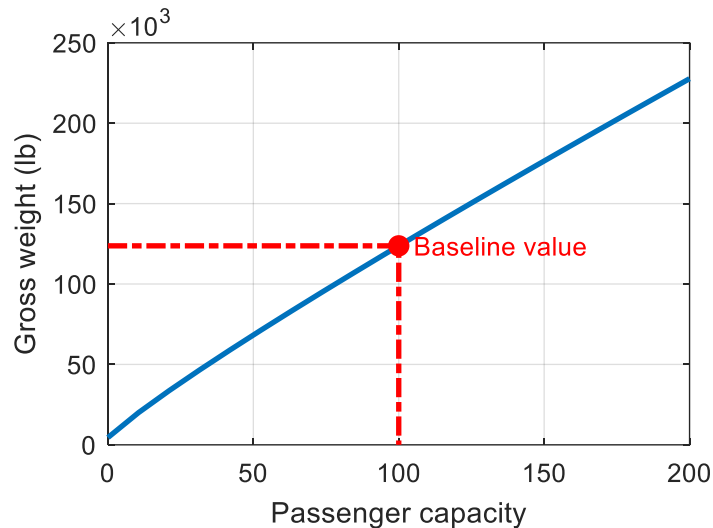


Figure 3-8. Regional jet gross weight versus passenger capacity.

In the WISDOM approach that is the subject of this work, the tool used for capturing preference information is called a preference map. The preference map is simply a map of some penalty or value as a function of a given parameter to which the preference is attached. An example of a basic preference map is shown below in Figure 3-9 for preference penalty as a function of passenger capacity, with regions reflecting the qualitative preferences A through E listed above called out on the map.

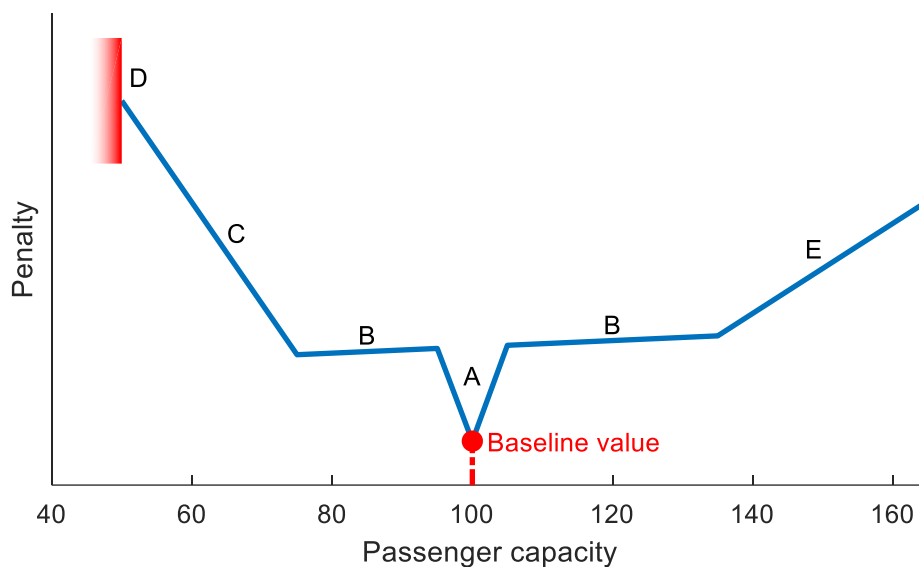


Figure 3-9. Basic preference map of passenger capacity.

In this case, the penalty on the ordinate axis of Figure 3-9 is arbitrary. What is more important is the qualitative shape of the preference map:

- Preference A, a desire to comply with the original requirements, is reflected in the steep trough surrounding the baseline value, such that in

this region, designs will be strongly driven to minimize penalty and gravitate toward the baseline value.

- Preference B, a slight drive toward a smaller and less resource-intensive development program, is captured by the gently increasing penalty in the region surrounding the baseline specification.
- Preference C, not competing with existing smaller products of the company, is captured by the increase in penalty on the left, with a type of 'soft' lower bound on passenger capacity created by the sudden change in preference map slope near ca. 75 passengers.
- Preference D, a hard limit on duplicating an existing product of the company, is implemented with a hard lower bound on passenger capacity.
- Preference E, not competing with the existing narrow-body market, like preference C, is reflected by the increased slope at the right and its associated 'soft' upper bound at the kink in the curve at 135 passenger capacity.

To get design results that integrate this preference information, preference maps from all parameters and figures of merit that have associated preferences are combined into a single objective function. Local minima are then found while searching through this design space using optimization algorithms. Further information on this can be found in Section 3.4, with information on the implementation details in Chapter 4.

With other preferences on weight, cost, productivity, revenue, and other figures of merit also influencing the direction the design takes, one would expect, based on the preference map in Figure 3-9, that the interesting designs would likely be clustered around 75, 100, and 135 passenger capacity vehicles. Some combinations of preferences on the other figures of merit (and with variations in their relative importance) may also drive some designs to be compromises that come between or go beyond these points. For the sake of this illustrative example, simple linear preferences are also captured for two other competing preferences:

- The desire to minimize development and operating cost, using gross weight as a surrogate for both (see also the preference map for gross weight shown in Figure 3-11 in Section 3.3.4 below).
- The desire to maximize revenue, both in sale price for the manufacturer and lucrative fares for the airline, using the product of passenger capacity and design range as a surrogate utility metric.

All penalties from the preference maps are summed to yield a single objective function to be used by search algorithms to find a varied set of design candidates. A histogram of the results when using this type of automated search of the design space integrating the information captured in preference maps is shown in Figure 3-10 (for ca. one thousand total design trials), with the preference map from Figure 3-9 in the background for reference.

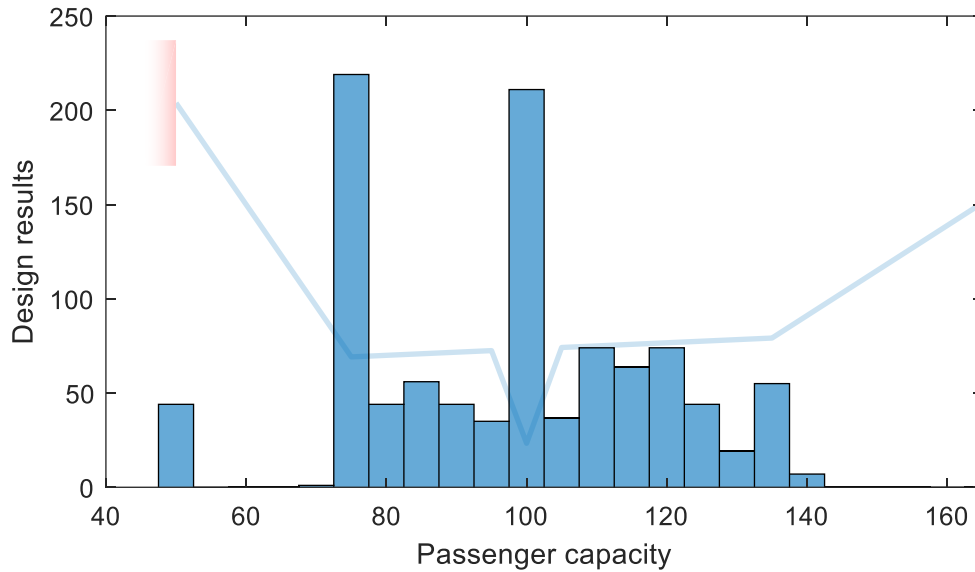


Figure 3-10. Clustering of design results driven by preferences captured in a preference map.

As expected, the results are clustered at the baseline specification, the ‘soft’ lower bound, the ‘hard’ lower bound, and a somewhat at the ‘soft’ upper bound. Due to variations in relative weighting of preferences, there are also some designs that do not close and/or are driven off the chart to the right with runaway weight.

Where the most frequent designs lie is an indicator of the interesting regions of the design space for further investigation. A qualitative interpretation of these results could be:

- A design that complies with the original specification of 100 passenger capacity is probably quite acceptable.
- Alternatively, an appealing new product development option could instead focus on a design philosophy of pursuing the smallest, least expensive, fastest development that would still be in a larger size class and give the company access to some new markets. This would likely be around a 75 passenger capacity aircraft.
- Or, given other drivers, quantitative or qualitative, that may be present, any passenger capacity in between the 75 and 100 points is justifiable,
- Capacities above 100 passengers would also be justifiable with a potential design philosophy of pursuing the largest, highest-revenue vehicle that is not in competition with existing narrow-body airliners.

Of course, the same conclusions could be reached in this example by inspection without the need to implement this analytical process. The difference is that the technique of capturing preferences to preference maps can scale up to higher dimensions with multiple parameters with complex and competing preferences in play. In this way, this type of exploration and decision making can be automated in the design loop such that the design decisions made using the algorithm are aligned with those that would be made by a human designer

examining the same area of the design space with deep understanding of all the factors and issues present for a given case.

There is no limit to the total number of preference maps that can be combined. For this example, a natural additional dimension to add when exploring which class of aircraft to develop would be a preference map for the aircraft range, with a resulting set of design candidates clustering at the most interesting potential combinations of passenger capacity and range.

3.3.4 Known uncaptured system model behaviors

A third type of information that the technique can capture and account for is the known deficiencies, idiosyncrasies, and other behaviors in the system model and its constituent analytical methods. It is often the case, especially in early design, that the modeling of the system under investigation is being built in parallel with the design progression. As the designer(s) and other stakeholders build up these models, they are aware of the analytical methods being employed and how the models are built. They therefore have a reasonable idea of where the models fall short, potentially having sets of inputs where accuracy is questionable or where a specific phenomenon is known to not be captured by the methods.

Going back to the illustrative example of using the fuel fraction method to size a small regional airliner, the fuel fraction sizing method is a very simple approach that is appealing for its usefulness very early in design studies when not many inputs for higher fidelity methods are known. However, just because it is early does not mean that some of these shortcomings should not be taken into account. The deficiencies are known, and that knowledge can be captured, at least roughly, and be integrated into the design iteration cycle.

Let us examine for an example the weight modeling used in the method. At its core, it relies on a model of the empty weight fraction of the vehicle as a simple function of the gross weight of the aircraft (Figure 3-2). Obviously, however, many other factors besides gross weight affect the ultimate empty weight fraction of the final design.

One common trade-off in aircraft development is aerodynamic efficiency versus weight and cost. Higher aspect ratio wings, for example, have less structural depth and require more material to sustain the same loads (not to mention aeroelastic constraints). If using gross weight as the top-level surrogate metric to minimize as a design objective, some of the qualitative information that can be captured is:

- A. The empty weight fraction trend is based on typical existing aircraft in this category. If the aerodynamic efficiency (characterized by lift to drag ratio) is higher than is typical for the category, the weight should be higher than the trend.
- B. Conversely, if aerodynamic efficiency is significantly lower than typical, there may be a slight weight decrease below the trend.
- C. The magnitude of the effects from items A and B above on the resulting gross weight is not precisely understood or quantified.

- D. Designs that incorporate an aerodynamic efficiency that is *significantly* higher than is typical for the category should be viewed with skepticism, possibly prone to aeroelastic or other unmodeled effects (for example fuel volume constraints created by a high aspect ratio wing) that could invalidate the design.

The classic fuel fraction sizing method does not take these factors into account, but with this method, a preference map, the same as discussed in Section 3.3.3, can be used to capture some of this knowledge. Because the gross weight of the aircraft is used as a surrogate primary metric to minimize for this example, the preference associated with gross weight is a simple linear function, as shown in the preference map in Figure 3-11 (which also features a hard lower bound to preclude what would be non-sensical negative values).

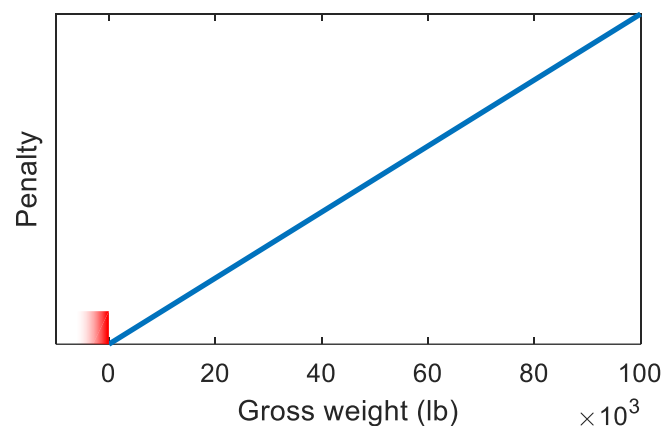


Figure 3-11. Linear preference map for gross weight, the surrogate objective to minimize.

Combined with this is the preference map associated with aerodynamic efficiency, shown below in Figure 3-12, which captures the qualitative information listed above.

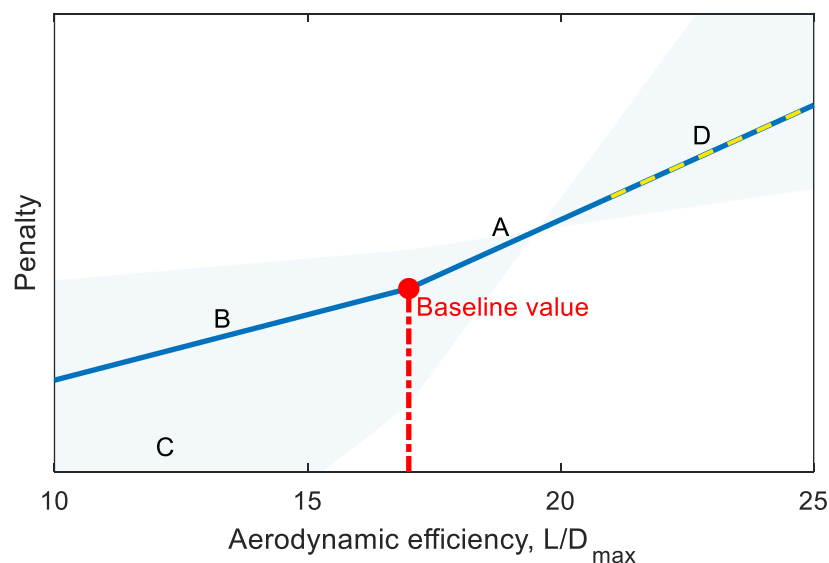


Figure 3-12. Preference map for capturing known system model characteristics.

Captured information items A and B are reflected by the varying slopes to the left and right of the baseline. Item C, the uncertainty in defining these slopes, is represented graphically by the shaded region showing the extents of the possible variation in scaling of the preference map. Finally, the information in D, the skepticism that should be attributed to designs with very high aerodynamic efficiency, is captured not by the shape of the preference map, but rather by explicitly defining the region where, for designs that fall in this domain for this parameter, a flag will be raised to make sure the designer is aware of the potential issue.

Introducing this very coarse preference map on aerodynamic efficiency affects the design results by effectively ‘thumbing the scales’ using the captured knowledge to modify the behavior of the system modeling as a function of aerodynamic efficiency, particularly the objective metric to minimize (in this case gross weight). Consider the gross weight as a function of aerodynamic efficiency, L/D_{max} , in Figure 3-13. Since the preference map for gross weight (Figure 3-11) is linear, the abstracted gross weight penalty as a function of aerodynamic efficiency is the identical shape.

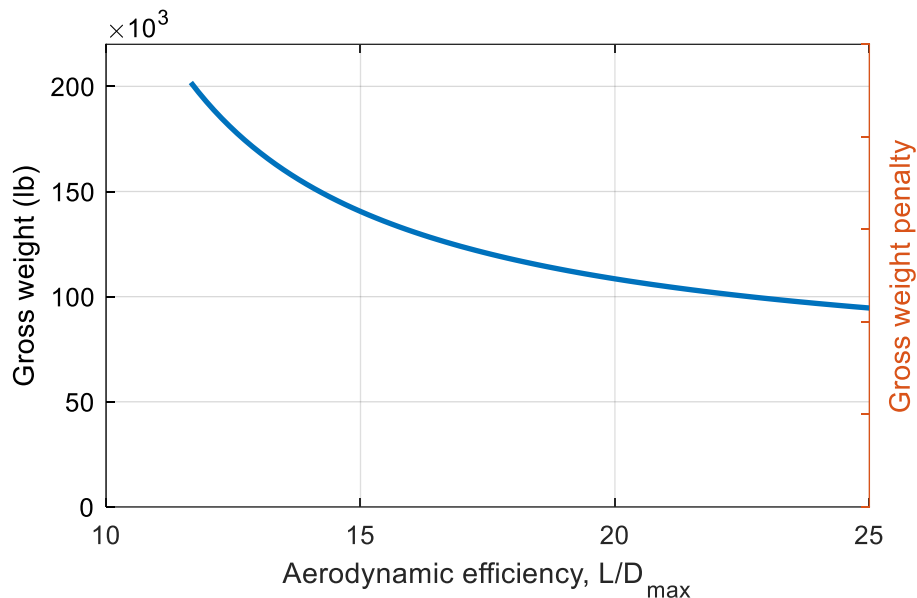


Figure 3-13. Gross weight as a function of aerodynamic efficiency.

With the addition of the preference map for aerodynamic efficiency on top of the preference map for gross weight, however, it is possible to have a qualitative shift in the trend. As shown in Figure 3-14, the extents of the possible relative weighting of the aerodynamic efficiency preference map (captured information item C) is captured in weightings relative to the gross weight penalty.

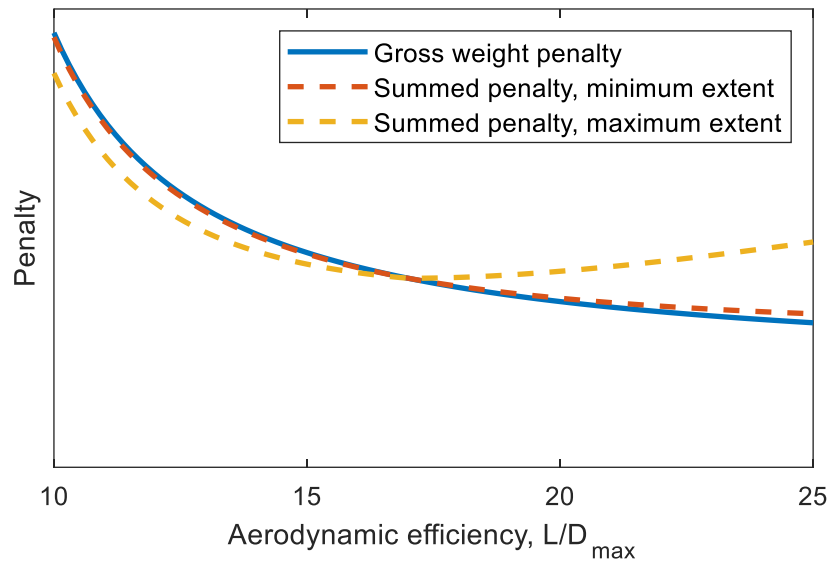


Figure 3-14. Extents of the effect of L/D_{max} preference map on qualitative nature of system modeling behavior.

Without this added information, designs that feature very high aerodynamic efficiency are artificially appealing. Normally, when conducting initial sizing of an aircraft using the fuel fraction sizing method, if aerodynamic efficiency is treated as a design variable, there is nothing in the mathematical system modeling that creates any driver at all for moderate and reasonable values for L/D . With the information added, however, the fact that there are some unmodeled consequences to very high L/D_{max} is captured and integrated into the information that an optimization solver would use to explore the design space.

If the aerodynamic efficiency is treated as an uncertain assumption parameter instead of a design variable, the integration of the preference map still plays an important role. In this case, whichever designs that benefit from an unusually high assumption for aerodynamic efficiency will not be unduly elevated to be top candidates. Conversely, designs with somewhat lower aerodynamic efficiency assumptions will not be inappropriately discarded due to the potential incorrect conclusion that the adverse weight impact is too great.

As discussed previously, the greatest benefits from this approach come when all the preference maps are combined such that the automatic search algorithms can make informed decisions that are as aligned as possible with the decisions an experienced human designer would make. The other great benefit is the significant difference in effort required to capture certain phenomena. To capture the effects of high aerodynamic efficiency (primarily achieved via high aspect ratio wings) on weight, significant research and modeling of structures, aerodynamics, dynamics, etc. could be spent, but instead a simple preference map is drawn using information and designer wisdom on hand, allowing the design process to progress quickly.

3.4 Application of optimization

As mentioned, the goal of the approach here is for the automated search, i.e., optimization algorithms, to make decisions that are as close as possible to the decisions that a human designer would make. This requires mapping preference maps onto an optimization problem in a certain way to integrate all captured information into the search algorithm. There are a multitude of existing off-the-shelf optimization algorithms, and in order to be able to leverage them, design variable preference information can be transformed into a standard-format for a single-objective optimization problem, which typically takes the form

$$x^* = \min_x f(x) \text{ subject to } \begin{cases} LB_i \leq x_i \leq UB_i, & i = 1, 2, 3, \dots, n \\ g_j(x) \leq 0, & j = 1, 2, 3, \dots, m \\ h_k(x) = 0, & k = 1, 2, 3, \dots, p \end{cases} . \quad (16)$$

3.4.1 Objective function

The objective function transformation that includes the preference maps is given by the sum of design variable preferences:

$$f(x) = \sum_{i=1}^k p_i(S(x)), \quad (17)$$

where S is the system model function that analyzes a design defined by design variables contained in design variable vector x and returns all figures of merit and parameters, including design variables themselves, to which preferences may be attached. Each of k preference functions, p , acts on its respective output parameter of S and yields the preference penalty value mapped to that parameter value based on the i^{th} preference map corresponding to the i^{th} parameter returned by the system model function.

3.4.2 Introducing variation

A primary result of the method presented here is generating interesting and useful sets of multiple design points from which insights can be gained and decisions can be made. There are three main elements that by themselves or combined lead to creating diverse sets of results.

3.4.2.1 Multi-start multimodal search

This method seeks to find many unique local minima by searching from many different starting points, x_0 . This is a simple but effective approach to multimodal optimization and finding multiple local optima, and each resulting j^{th} local optimization solution,

$$x_j^* = \min_x f(x) \forall j \in \{1, 2, \dots, n\}, \quad (18)$$

is associated with its j^{th} search starting point, $x_{0,j} \forall j \in \{1, 2, \dots, n\}$.

3.4.2.2 Preference map uncertainty scaling

The other mechanism for adding variation to the resulting set of design optimization solutions leverages the parameter preference map property that is a measure of uncertainty in the preference map magnitude. This property is depicted visually as the shaded region C of Figure 3-12 in Section 3.3.4 above. This property is integrated into the objective function mathematically by adding a factor such that Equation (17) becomes:

$$f_j(\mathbf{x}) = \sum_{i=1}^k \rho_i^{X_{i,j} \sim U(-1,1)} \cdot p_i(\mathbf{S}(\mathbf{x})), \quad (19)$$

where in addition to using a different starting point, $\mathbf{x}_{0,j}$, for each search, the uncertainty factor, ρ , for each preference map is used to scale the ordinate of the preference map stochastically. This random scaling is based on uniformly distributed random number X between minus one and one. An uncertainty factor value of 4 assigned to a given preference map, for example, means that the preference map may be scaled by between $\frac{1}{4}$ and 4, for a total ratio of maximum possible scaling to minimum possible scaling of 16.

3.4.2.3 Stochastic assumption parameters

Recall from the illustration in Section 3.3.2 that significant additional information is captured by using probability distributions in lieu of singular parameter values for uncertain technical assumptions. To additionally capture the variation due to uncertain assumptions, these parameters are randomly assigned values according to their respective specified probability density functions. These parameters are used in the system model objective function separately from the design variables, \mathbf{x} , that are allowed to vary during each search. Each j^{th} solution, \mathbf{x}_j^* , in addition to being affected by its starting point, $\mathbf{x}_{0,j}$, is influenced by the randomly generated uncertain assumption parameters, $\boldsymbol{\omega}_j$:

$$\mathbf{x}_j^* = \min_{\mathbf{x}} \sum_{i=1}^k \rho_i^{X_{i,j} \sim U(-1,1)} \cdot p_i(\mathbf{S}(\mathbf{x}, \boldsymbol{\omega}_j)). \quad (20)$$

A flowchart representation of how the objective function is constructed is shown in Figure 3-15.

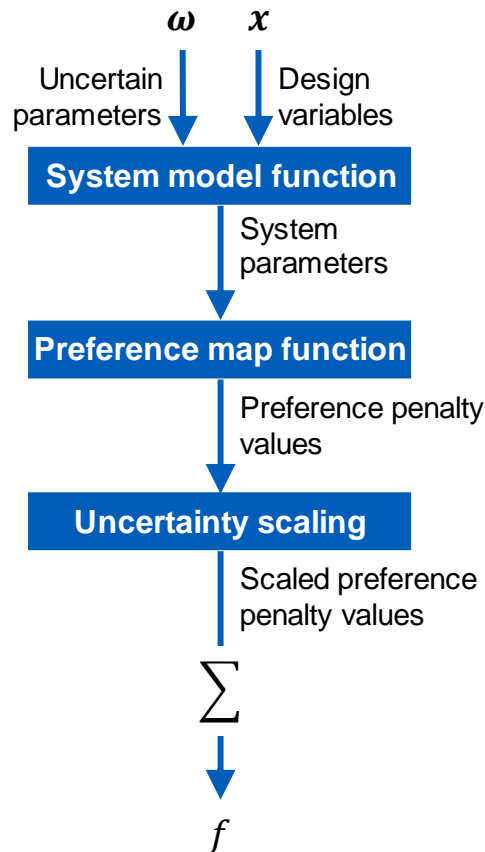


Figure 3-15. Flowchart of objective function integrating preference maps.

Observe that if there are no uncertain assumptions at play and all preference maps are linear or affine without uncertainty scaling factors, the technique simply regresses to a weighted sum approach to multi-objective optimization. In the case that the system modeling is also smooth and convex, i.e., convergence to the one existing local and global optimum is ensured, only the uncertainty factors, not the varying start points, form variation in the resulting design point set. If all uncertainty factors, ρ , are equal to one, then there will be no variation in results and each j^{th} optimization will converge to the same solution, reducing to a conventional implementation of optimization.

3.4.3 Constraints

Preference information attached to design parameters often contains hard bounds on the parameter. The treatment of these hard bounds depends on if the parameter is an input to the system model function, i.e., it is a design variable, or if it is an output. Upper or lower bounds are implemented as simple side constraints for input design variables and as nonlinear inequality constraint functions for bounds on output design variables.

There are many cases in design iteration where it is also desirable to set an output of the system model function to a fixed value. In this case it is appropriate to enforce this as nonlinear equality constraint functions. It is also sometimes the case that parameters are both inputs and outputs and that they must be consistent for the design to make sense or close. An example would be an estimated gross weight input and a calculated gross weight output. These

would also be enforced as nonlinear equality constraints. This provides the side benefit of removing the need for the implementation of the system model to inefficiently converge internally.

Figure 3-16 shows a version of Figure 3-15 that integrates a flowchart of how some of the constraints are implemented.

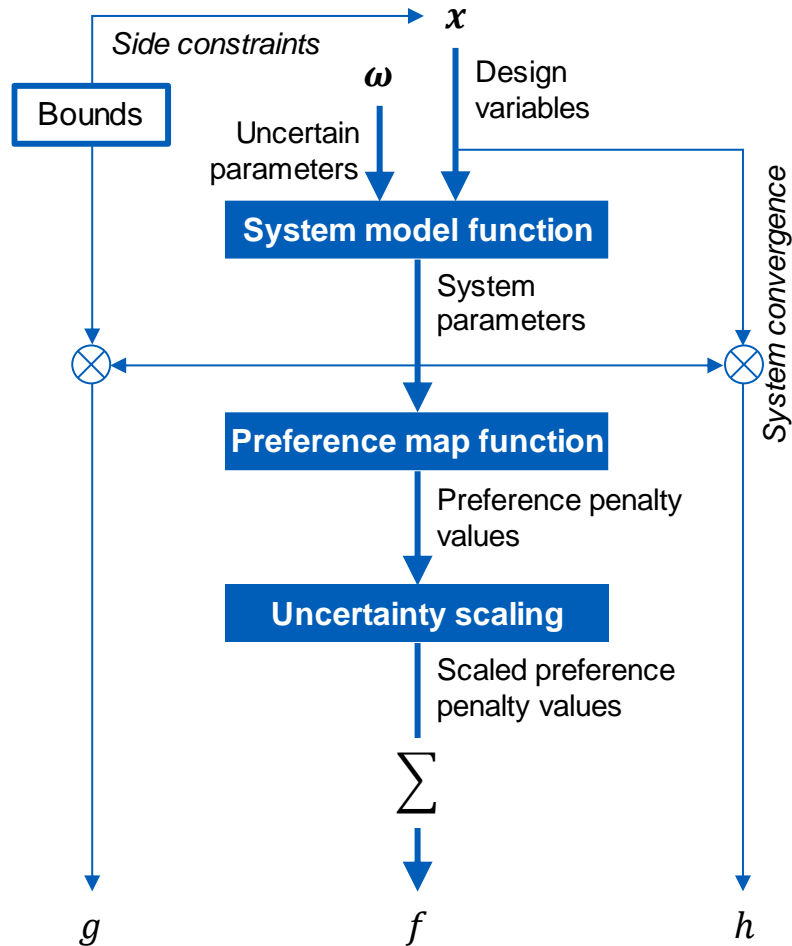


Figure 3-16. Flowchart of objective function with constraints.

3.5 Understanding and decision-making

The application of optimization is not the final step in the process for this method. The variety of results yielded by that step must be reduced to manageable and understandable alternatives from which insights can be gained, decisions can be made, and the next iteration can proceed with further changes and refinements.

3.5.1 Nature of optimization results

The optimization process in this method involves running multiple optimization studies, and the results of each optimization will vary from one to the next due to the factors discussed in Section 3.4.2 above that introduce variation. The total number of optimization runs is determined by the designer as a compromise between computing time, thoroughness of the search of the design space, and statistical significance of the stochastically derived insights that can come from the results. This could mean that there are dozens, hundreds, or thousands of possible design alternatives. Assuming optimization convergence, all of these results are closed designs that are locally optimized for the given assumptions and preferences, so any selection of a single alternative from these will be a valid and justifiable design option.

3.5.2 Processing of optimization results

Given that in many cases the total number of design results will be far too many for a designer to individually scrutinize, some tools are necessary to reduce the set of results down to a comprehensible form for designers and other decision makers. This form may be a condensed set of candidates that is finite enough for comprehension, but it may also take the form of some other kind of analysis or visualization of the results as a whole.

A simple histogram is one of the tools available for gaining insight into results. Not only, as per the examples shown in Figure 3-5 and Figure 3-7 in Section 3.3.2, can it provide information on the relative likelihood of outcomes, but it can also show where results are clustered, indicating potentially interesting regions of the design space for further exploration, as in Figure 3-10 in Section 3.3.3. This type of clustering visualization can be extended to other types of visualizations, such as scatter matrices similar to Figure 2-11 or corner plots (a related multidimensional visualization technique presented in Chapter 4), revealing where, despite variations in search starting points, relative preferences, or even uncertain assumptions, similar or identical designs still arise.

Outliers, the converse to clustered and high-frequency regions of the design space shown by the results, can also serve a function in providing further understanding of the design space. Besides being valid and potentially interesting design alternatives themselves, experience has shown that the existence and location of outliers can provide information that may call into question certain assumptions or the way in which the system is modeled.

It can also be useful, instead of visualizing the entire set of results, to focus on reducing a large set of results down to a more manageable set of distinct design

alternatives that are representative or indicative of the larger set. Since in most cases most or all resulting design points are Pareto dominant, reducing to only a Pareto set does not usually prove a significant enough reduction to be useful. More often, other methods of classification and clustering are required. Again, methods for doing so are discussed further in Chapter 4. The result of clustering is a handful of representative or interesting design candidates that are few enough that they can all be laid out in front of stakeholders and act as artifacts to facilitate understanding and informed discussion.

3.5.3 Iteration actions

Whatever form the processed results take, usually the aim is not particularly focused on explicit ranking, prioritization, or selection of designs. Rather, the primary purpose is to direct actions that will influence the approach and outcome of the next iteration.

In the design process of any moderately complex system, there are a significant number of areas where efforts can be spent or changes can be made from one design iteration to the next. A very common activity in iterating in early design is continual improvement and refinement of the system modeling, and the understanding provided in this method can help direct those efforts. The other major activity between iterations is locking in or narrowing down on requirements, design decisions, and other design parameters. This is the design space 'pruning' that was discussed Section 1.3.3. The specific implementation of the method, discussed in the next Chapter, provides several features and tools designed to facilitate this design refinement and pruning process in an efficient and flexible manner.

4 Implementation

The method as described in the previous chapter is in principle sufficient for prototyping and demonstrating the technique and assessing how it fulfills the two primary objectives of this work. However, the implementation of the technique in a way that is practical for real use is not entirely straightforward and contains some interesting challenges that must be addressed. This chapter lays out the specific workflow, the architecture of the software implementation, and the details of some of the key components in their current state at the time of this writing, along with some of the reasoning behind implementation decisions.

4.1 Overview

When building a set of software tools for a one-off design task, it is often not a terribly significant additional step to put those tools together into a single design tool or design suite to be used for a later project. However, the moment that any new design challenge deviates qualitatively from the original design task, the all-encompassing design tool requires significant modification to be applicable and useful. Recognizing this, instead of building a software suite with the intention of quickly and easily re-using it for each new design undertaking, this author simply maintains a library of tools that are useful for various design-related tasks and analyses, with the expectation that each new project, which will likely be fundamentally different from the last, will require putting these tools together in a new and unique way. The library is called the Design Understanding and Exploration Library, or DUEL, so named also for the nature of engineering design and development as always being characterized by conflict and competition between various drivers and aspects. The Well-Informed Search Design Optimization Method (WISDOM) approach that is the subject of this work was implemented as a tool within DUEL as a framework called the Well-Informed Search Environment (WISE).

4.1.1 Implementation priorities and requirements

Just like with the design of a new air vehicle product, it can be helpful when developing a new piece of software to keep in mind some explicit priorities that help inform design decisions and requirements. These priorities can also be thought of as themes or, per Lempia & Miller (2009), goals. In any case, for this software development project, the general themes and priorities are usefulness, usability, flexibility and reusability, and speed, expounded below:

4.1.1.1 Early and broad utility

The tool is explicitly meant to be used in early design studies, as discussed in Section 3.1. However, it is also important to try to make the tool useful in activities that both precede and follow early design. In this way, the user is more likely to begin to use the tool or at least think about using the tool from day one and, for example, build system models and analyses with usage of the tool in mind. Conversely, the tool should be built in such a way that minimal requirements are inflicted on the system model so that the model can be useful on its own before integration and use with the framework. If the tool is not useful from extremely early on, the likely alternative scenario is that the designer becomes too committed down one path, and the friction involved in pivoting to use of the tool is too great to make its use appealing.

This usefulness goal is also a driver for making the framework useful for additional activities within and beyond early design studies. For moving beyond early design, it is desirable that the framework be useful in preliminary design or at least keeping the transition to preliminary design in mind for projects that get that far. For activities within early design studies, what this means is that there is a high desirability for the framework to be built in such a way that it makes sense to implement some design tasks within the framework, unrelated to the WISDOM searching, that would otherwise be done externally. A good example of this is creating plots, parametric sweeps, or tables of design data. These

types of artifacts are often needed for the design reviews, proposals, or reports that are common deliverables in design projects.

4.1.1.2 Usability

The WISE tool should not only be useful, but highly usable. What this means is that the tool should be low effort to use and should not introduce significant unnecessary burdens or barriers to entry in the form of superfluous activities or added cognitive load. A derived goal of this usability priority is that, at least for some activities, using the framework should not only result in a long-term reduction in overall effort or improvement in outcomes, but also provide a near-term reduction in effort. In other words, a designer may have many alternatives in the toolbox for accomplishing a given task, and the tools that are easy to use early may be selected over those that are high effort early on but save effort in the long term. The design of the framework therefore should seek to represent a similar or decreased level of effort as comparable alternatives.

An example of this is implementing the common task of closing a design. It can be very quick to build a simple analysis script or spreadsheet that, for a given set of design parameters describing a system, provides a computed result. In some cases, writing the loops or implementing the functions to make the design converge can take more time than implementing the analysis itself. This is an opportunity for the WISE framework to be an attractive option. For applications such as setting up and running an optimization problem, the framework has even more potential to not just match the status quo level of effort as a conventional optimization implementation, but to offer significantly reduced effort.

4.1.1.3 Flexibility and reuse

The architecture of the software should be built keeping in mind the wide variety of design tasks for which it might be used. This stems from lessons learned in developing the Conceptual Optimization of Rotorcraft Environment (CORE) (Sartorius, 2011b) for this author's Master's thesis. In the case of CORE, a useful framework was built for exploring the design space and optimizing helicopters or lift- and/or thrust-compounding helicopters. However, the parts of the tools that were specific to helicopter design and analysis were intimately embedded in the parts of the tool that facilitated design, optimization, understanding, and visualization of the design space – so much so that neither aspect was at all useful alone or reusable for other tasks. With this in mind, the WISE framework should have the flexibility to be applied to any type of early design task. This means not just independent of aircraft type, but also not restricted to just aeronautical applications. Along these same lines, there should also be the flexibility with the framework to build and use a wide variety of domain-specific tools and resources, including those not written in the same language as the framework itself.

The reusability priority refers not just to the framework, but also the artifacts that are created on a project-by-project basis. There may be very little or significant overlap from one project to the next, and in the latter case it can represent not only significant effort savings but also valuable preservation of expertise if some project-specific artifacts can be reused. This carries implications on the format

with which information is captured as well as drives for features that facilitate, for example, good documentation. Both flexibility and reusability drive towards a level of modularity in the architecture, especially with an emphasis on making sure it is possible to cleanly keep separate anything project-specific from the framework tool.

4.1.1.4 Speed

One of the pain points of using optimization and MDO in design can be long computation times. The WISDOM approach relies not only on optimization, but on running each optimization tens, hundreds, or even thousands of times just for one iteration of the method. Because the approach is known to be fundamentally computationally expensive, thought should be given from the very beginning to profiling the software and making sure that any possible bottlenecks and any possible additional overhead resulting from the implementation are kept to a minimum.

The speed theme applies not just to use, but also to development of the framework. Speed of development creates yet another incentive for some modularity in the architecture. Modularity facilitates tests as well as experiments to try out new approaches to determine if better results can be achieved. An example of this is the use of a standard interface to optimization algorithms, which not only allows for easily experimenting with alternatives (and even making it easy to offer alternatives to users), but also makes it much easier to integrate some new and possibly superior and faster off-the-shelf option if and when it becomes available.

4.1.2 Nominal workflow

A nominal workflow is envisioned for the use of the framework. The workflow should support the typical iterative process of aircraft design, as discussed in Section 1.2.1 and illustrated in Figure 1-3. In addition, as shown in Figure 4-1 below, it is important for the initial setup to be possible without significant burden imposed by system model syntax restrictions. What is unavoidable, however, is the sometimes-tedious step of formally capturing, in the form of preference maps, all relevant information on hand about parameters of interest. This exercise in itself can sometimes require some back and forth with adjustments to the specific input/output scheme of the system model, but as will be shown in Section 4.2 below, it is possible to segregate these activities and be able to build a system model independent of the initial creation of design variables.

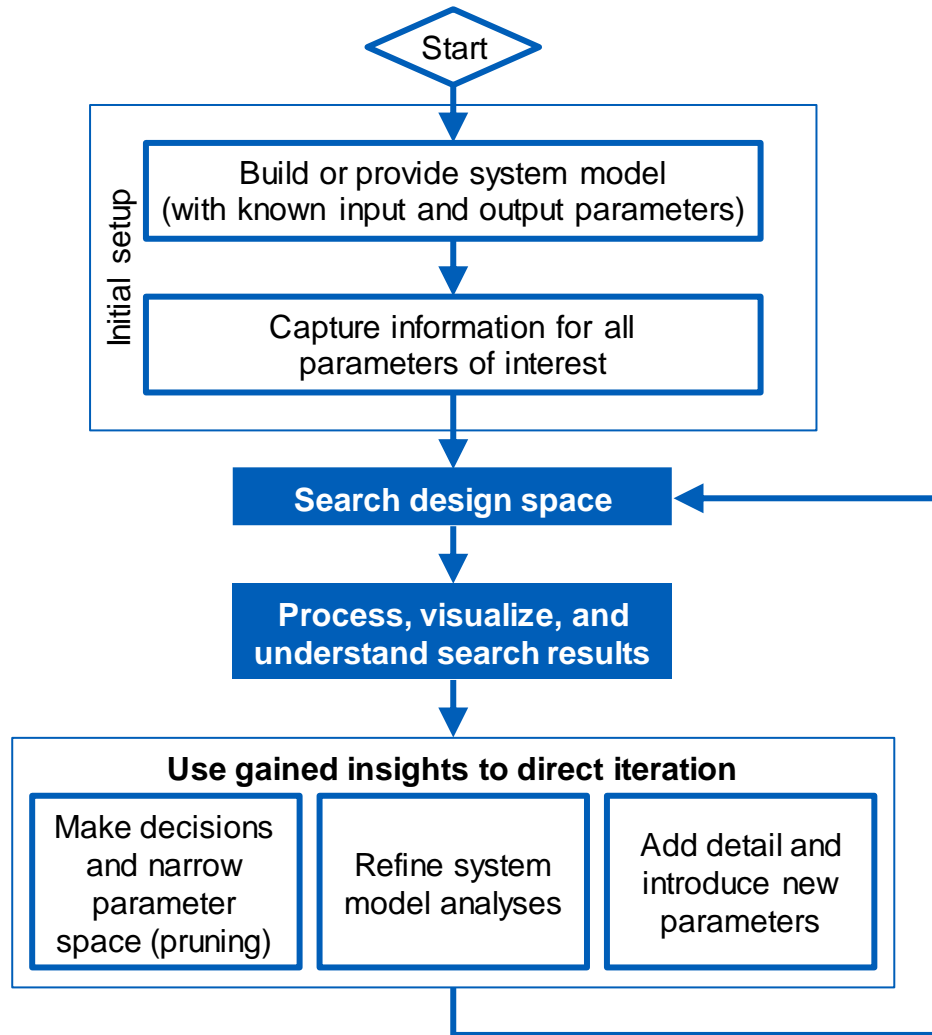


Figure 4-1. Nominal workflow for WISE framework.

Once initial setup is complete, the iteration workflow is designed to be flexible enough to allow the framework to continue to be useful as design continues to mature beyond just the early studies, with seamless transition to more advanced design stages. The key activities at each iteration involve continuing down the iterative design spiral, continually increasing the definition of the design, understanding of the performance, and increased confidence in the analytical results.

4.1.3 MATLAB as selected programming language

One of the first decisions to make in creating the WISE framework is the language to write it in. Because the rest of DUEL is already almost entirely written in MATLAB, it is a natural choice for WISE implementation. The tool is built with MATLAB up to version R2018b (*MATLAB (Version 9.5)*, 2018) and relies on the Optimization Toolbox (*Optimization Toolbox (Version 8.2)*, 2018). Still, other languages were considered. Python 3 is the most attractive alternative since it is free and therefore easy to distribute. Python also has a wide variety of freely available packages built by a strong developer community. However, MATLAB was selected for implementation because it provides some other key advantages:

4.1.3.1 Common environment

MATLAB is a programming language, but it also refers to the integrated development environment (IDE) that is used for editing and running MATLAB code. Having a single common IDE that is used by every single user of a language provides many advantages for development. The WISE framework is built for exploring a design, interacting with the models, and making adjustments. It is therefore fundamentally interactive, meaning that there are necessarily some graphical user interface (GUI) elements. Using MATLAB allows for piggybacking on the common IDE for creating some of the desired GUI capabilities.

4.1.3.2 Command line use

Along the same lines as having an environment that is fundamentally interactive, it is helpful that MATLAB is built with command line usage as a primary use case interface. This means that, unlike other languages that may necessitate importing packages or defining variables at the top of every source file, MATLAB allows for relatively seamless on-the-fly interaction with the defined variables and objects in the workspace via the command line. This is a very useful feature when exploring a new design space, as the direction the designer may want to go is not always known ahead of time.

4.1.3.3 Designed for technical computing

At its core, MATLAB is targeted almost exclusively at scientists and engineers. This means that there are tools that make it useful for engineering-related tasks such as analysis and optimization.

4.1.3.4 Popularity and familiarity

Being an excellent tool for scientists and engineers has led MATLAB to become a very popular programming language and IDE with engineers. Since it is popular with the type of users to whom the WISE framework is targeted also means that many potential users will already have a level of familiarity with MATLAB. The popularity of MATLAB has also led to a strong user community, including an active exchange of freely available tools and modules.

4.1.3.5 Interaction with other languages

One central element of the WISOM approach is the system model. One highly desirable feature of the implementation is that whatever language is chosen does not necessarily dictate the language in which the system model must be written. MATLAB accomplishes this by having relatively straightforward means available to call functions written in other languages such as C or Python.

4.1.3.6 Vectorization

The word 'MATLAB' comes from 'matrix laboratory,' so at its core it is very fast and efficient at dealing with vectors, arrays, and even multi-dimensional arrays of data. This characteristic can be quite useful for tasks that involve repeating similar operations that in other languages would normally be done using some sort of loop. In MATLAB, loops can be avoided, and a workflow is feasible whereby initial development is done using scalar data and little to no modification is required to repeat the analysis with large arrays of inputs.

4.1.3.7 Object-oriented

MATLAB supports object-oriented programming. Though much typical usage of MATLAB takes on a functional programming paradigm, the object-oriented capability and the ability to define new custom classes is an invaluable tool in keeping the WISE framework organized and usable. As will be shown in the discussion of the framework organization in Section 4.1.4 below, encapsulating data relevant to design variables and the design space in defined objects is an essential feature for keeping the design organized and reducing the burden involved in implementing key design space search and exploration functions.

4.1.4 Framework components and organization

The two major components of the WISE framework (whose readme file is included in Appendix A1) are the `DesignSpace` class and the `DesignVariable` class. In addition to these two main classes, a central part of any project is the system model function that analyzes the expected performance of a given system. A `DesignSpace` object defines a given project by containing a vector of `DesignVariable` objects and a pointer to the system model function (in the form of a function handle), as shown in Figure 4-2. This architecture lends to a certain amount of modularity inasmuch as the system model can stand and be useful on its own. Each design variable can also be handled and manipulated in isolation.

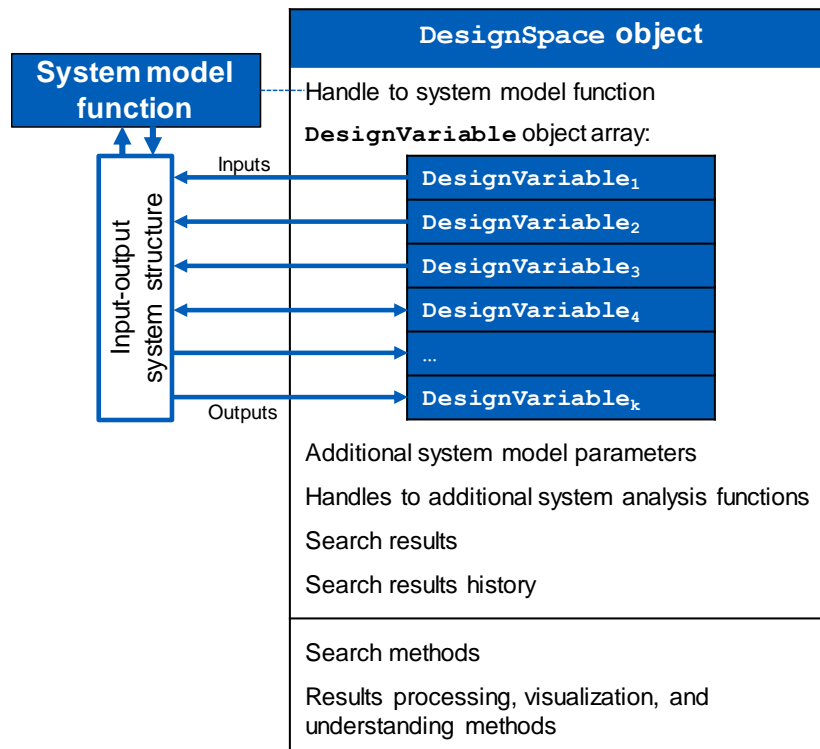


Figure 4-2. Top-level organization of the WISE framework setup of a design.

The primary interface between the design variables and the system model function is the input-output system structure. This structure defines the system and is a simple MATLAB standard data type (a `struct`) that provides significant flexibility in working with the system model, which is discussed in Section 4.2.

Much of the information contained in a `DesignSpace` setup, such as defining exactly which design variables are inputs, outputs, or both, lies in the `DesignVariable` objects. The design variable class, its attributes, and its methods for defining and iterating design variable attributes are discussed in Section 4.3.

Further `DesignSpace` class attributes and methods facilitate conducting WISDOM searching, including processing and storing results, details of which are discussed in Section 4.4.

4.2 System model function

The system model function has already been mentioned in Section 3.4.1. It is, abstractly, an arbitrary function that takes as inputs all design variables and returns all parameters of interest related to the system. However, consideration has been given to the syntax and other aspects to increase flexibility and accommodate an early design workflow.

4.2.1 Syntax

The system model function syntax is an important contributor to the usability and flexibility of the framework, especially compared to more traditional MDO setup requirements. A standard-format optimization problem in MATLAB (as is also typical of optimization formulations in other languages) requires an objective function to be defined as a function of a single input variable that is a vector of floating-point numbers (example in Figure 4-3). Note how this means that a necessary task for the designer is keeping track of which parameters lie at which index of the design vector, x . Not only is this bookkeeping an extra burden and error-prone, the list also requires revision whenever a new parameter is added, say to increase the fidelity of the analysis, or whenever a design parameter is removed by locking it into a set value.

```
function objectiveValue = objective_function(x)

input_1 = x(1);
input_2 = x(2);
input_3 = x(3);
...
input_n = x(n);
...
Analysis
...
objectiveValue = ...;
```

Figure 4-3. Traditional MDO typical required syntax for objective function.

In addition, the constraints for a typical MDO setup must often be defined in an entirely separate function (example in Figure 4-4). Again, the same burden of tracking parameter indices is present (and it must be kept synchronized with the list in the objective function). It is also highly likely in early aircraft design that the same or similar analysis will inform both the objective function and the constraints. The analysis that determines the performance of the vehicle over a mission profile, for example, would likely play a role in informing some objective value to minimize while also being a key part of informing key constraints on minimum performance.

```

function [inequalities, equalities] = constraint_function(x)

input_1 = x(1);
input_2 = x(2);
input_3 = x(3);
...
input_n = x(n);
...
Analysis
...
inequalities(1) = ...;
inequalities(2) = ...;
...
inequalities(m) = ...;

equalities(1) = ...;
equalities(2) = ...;
...
equalities(p) = ...;

```

Figure 4-4. Typical MDO required syntax for defining nonlinear constraints.

The syntax for the system model function in the WISE framework breaks away from the traditional MDO pattern by a) putting all threads of analysis in a single function and b) leveraging MATLAB's `struct` data type for inputs and outputs. The `struct` data type stores data in an unlimited number of named containers called fields. This significantly eases the burden on the designer by using a meaningful name instead of numeric indices and allowing for an arbitrary number of additional fields. An example syntax for a WISE system model function is shown in Figure 4-5.

```

function S = system_model_function(S)

intermediateParameter = foo(S.input_1, S.input_2);
...
Analysis
...
S.output_1 = bar(S.intermediateParameter, S.input_n);
S.output_2 = ...;
...
Analysis
...
S.output_k = ...;

```

Figure 4-5. System model function syntax.

The variable `S` is the input-output system structure as depicted earlier in Figure 4-2, and its fields contain rich descriptive system data. It is important to note that the implementation here puts no restrictions at all on what fields are or are not required in the system model, and it is entirely up to the designer for each new project to decide on the naming and organization of the relevant input-output system structure fields. This lack of enforced names and organization is a valuable feature for facilitating early design workflows but can also present a hazard as a project may grow, particularly if it grows to the point where many individuals are authoring submodules of the system analyses. At that point, it may begin to be prudent to implement an organized mapping of input-output system structure fields onto pre-standardized aeronautical system parameterizations such as the Common Parametric Aircraft Configuration

Scheme, CPACS (Nagel et al., 2012), or ADDAM, the Aircraft Design Data Model (Herbst & Hornung, 2015). In this way, the set of design parameters within the WISE framework is still limited and manageable while the system model analysis can be done collaboratively or easily implemented efficiently with existing tools.

4.2.2 Input and output variable types

Another advantage of the `struct` data type is that a field can contain data of any type (even another sub-structure), unrestricted to the floating-point numbers in a typical MDO function input vector. Therefore, the most meaningful and appropriate data type can be used on a per-parameter basis, and there is also added flexibility for a parameter to take on non-scalar values ad hoc, which can be leveraged for parameter sweeps, Monte Carlo analyses, plotting, etc.

There are two types of continuous variables that are typically useful for aircraft design and are supported as system model inputs and outputs. The first is the classic MATLAB double-precision floating-point `double` (the default numeric data storage type in MATLAB). The second is the `DimVar` (“dimensioned variable”) data type, which is similar in functionality to `double` but carries with each variable meaningful physical units. This data type and the associated Physical Units Toolbox (Sartorius, 2019a) enables the use of the most meaningful unit for a given parameter without the need to implement any unit conversions in software, along with the added benefit of catching unit-related coding errors by enforcing unit consistency (adding a mass to a weight, for example, results in an error). An additional benefit of the `struct` data type is that it displays by default in the MATLAB command window in a compact and meaningful way. Dimensioned variables also display along with their units, including when displayed as part of the input-output system structure.

Discrete system model inputs and outputs have even more flexibility regarding data type and can be nearly any MATLAB type or class. Some aircraft-related examples of leveraging discrete data types include using the `logical` (true or false) data type to indicate the presence or absence of a feature, system, or characteristic, e.g. `S.is_pressurized`, or using the `string` data type to distinguish between several named alternatives, e.g. `S.spar_material = "2024-T6 Aluminum"`.

4.2.3 Typical evolution of system model function

One path to creating a system model function is the simple use of an existing analysis suite, in which case the only additional step is creating a MATLAB function wrapper. Another typical case for early aircraft design involves not only creating a system model function from scratch but doing so in a way that is a natural progression and evolution from the very first lines of code up through a complex multi-module analysis that works with the framework.

The first lines of code are usually in a simple script with inputs (key design variables and assumptions) declared near the beginning, followed by some analysis that adds variables of interest to workspace. A simple script such as this is quite useful for setting up the analytical approach, debugging, and for

some very initial understanding of the system and its modeling. For use with the WISE framework, the key variables of interest, both inputs and outputs, should be made fields of the system structure instead of standalone variables by simply prepending e.g. "S." to the variable names. This script can then be easily made into a function by adding a function header with the simple syntax shown in Figure 4-5 above.

Instead of removing all lines that declare the key input variables, the created system model function can remain a useful stand-alone script by adding a switch for different behavior based on if it is being used as a function or a script. Also, additional parameters can be passed to the function when the function is being used for one-off analysis instead of inside WISDOM search loops. Pseudocode for these features is shown in Figure 4-6.

```
function S = system_model_function(S_in, flag_1, flag_2, ...)

% Set static inputs, parameters, and assumptions:
S.input_1 = ...;
S.input_2 = ...;
...
S.input_n = ...;

if input provided
    overwrite fields of S with fields of S_in
end

...
Analysis and setting additional fields of S
...

if flags set or running as script
    run additional expensive analyses
    display parameters of interest
    produce plots and reports
end
```

Figure 4-6. System model function syntax and pseudocode with additional typical features.

The additional inputs to the system model function are the ‘additional system model parameters’ shown previously in Figure 4-2, and they enable code that normally should not run during optimization to nevertheless remain in the system model code. Examples of this are more expensive analyses, code that displays parameters in the command window, code that creates plots, or code that does not run properly with the array inputs sometimes used in the framework. In lieu of or in addition to building this directly into the system model function, the “handles to additional system analysis functions” from Figure 4-2 also enable extra functionality for the case of performing richer depth of analysis on a one-off design. These additional system analysis functions use the same simple structure-in, structure-out syntax as the primary system model function, and they can be chained together such that each subsequent function can utilize information generated by the last, as discussed further in Section 4.5.3.4. An example of an in-depth additional system analysis function would be a module that creates a 3D model of a design point for visualization and generating a drawing.

4.3 Design variable class

One of the primary objectives of this work is to formally capture stakeholder knowledge and wisdom, and the design variable class is what facilitates this. As such, the `DesignVariable` class has many attributes (called properties in MATLAB terminology). The main user-facing attributes fall into a few categories, which are described in this section along with descriptions of some of the user-facing design variable class methods. In addition, many dependent properties, set/get methods, and other utility methods, not discussed here, are present in the class to further enable the capabilities and bolster ease of use of the class. The top-level help block documentation of the class is in Appendix A2.

4.3.1 Descriptive and system model interfacing attributes

Each parameter described by a design variable object should be easy to work with for the system model function, the WISE framework, and for the humans interacting with the parameter. To this end, there are a few basic user-facing attributes that the designer sets to make a given parameter easy to understand for human and machine.

4.3.1.1 Name

The `name` attribute is used as the field name for the input-output system structure. There are therefore some hard restrictions regarding making sure it is a valid string of characters to use for a field name, and there are also some softer practical restrictions. Even though this name acts as the primary shorthand reference to the parameter, it may still be referenced frequently in the system model function, so it should be both descriptive and succinct.

4.3.1.2 Description

To capture more information about the parameter than what can just be captured in the name, the optional `description` attribute is available. Normally this is simply used with a descriptive string. However, as an arbitrary container, it can also hold more complete information that may be useful to have co-located with the other parameter attributes, including structured information that could be, for example, links or references to external documentation.

4.3.1.3 Label

The framework uses plotting in many places, and the `name` attribute, with its restrictions, does not always make for the clearest of most attractive label for things like plot axes. The optional `label` attribute fills this role. The label is a simple TeX-interpretable string, but it makes the plots generated within the framework clearer, with a shorter path with less labor required from generation to inclusion in a report, memo, design review, or other deliverable.

4.3.1.4 Units

It is greatly beneficial when discussing requirements, design parameters, and preferences to be able to have that quantitative discussion using the physical units that are most natural and that the stakeholders are most used to thinking

in for that parameter. Length dimensions of an air vehicle may best be thought of in inches or millimeters, for example, while altitudes are best discussed in terms of feet and long distances in terms of kilometers or nautical miles. That is why it is so important to be able to attach physical units to each design variable using the `units` attribute.

If specified for a given design variable, the appropriate dimensioned variable will be passed to, and expected from, the system model function. Through the use of the Physical Units Toolbox, not only are unit conversions unnecessary within the system model function or anywhere else, but the preferred unit for working with a given parameter can be easily changed, without affecting the preferences or analyses, simply by changing the `units` attribute. The units are also automatically included wherever the design variable is displayed, including when labeling plot axes.

4.3.1.5 Input-output type

Features of the design space search methods (see Section 4.4 below) mean that when searching the design space, for the most part, the designer does not have to think about parameters that are inputs to the system model differently than outputs from the system model. These features make it so that when building the system model function, the most straightforward and fastest to develop analyses can be used without the need to try to invert complex equations or iteratively converge to solutions.

Still, the tool requires some indication of what the system model expects, so the `ioType` attribute allows the user to distinguish between parameters that are inputs to the system model function, outputs, or both. A parameter can be both if its value is used as an input to some analysis but ultimately a better estimate is calculated as part of the system model function. A common example is aircraft gross weight, which is used for almost all performance calculations but is also used itself as an input to calculate the weights of various components for the sake of a gross weight buildup. If a parameter is both an input and an output, this is signaled by specifying a required 'greater than' relationship (input must be greater than output), 'less than' relationship, or 'equal to' relationship to be enforced as a constraint as described in Section 3.4.3.

4.3.1.6 Closing flag

The `closing` attribute is only used for parameters that are both inputs and outputs. If the closing flag is set (i.e., its logical value is set to `true`), this is an indication that if the input does not equal the output for a given parameter, the design is not considered a 'closed' design. A design that is not closed is a design that does not make physical sense and is considered invalid.

4.3.1.7 Smallest meaningful step

It is typical for optimization algorithms to use very small finite differences to determine gradient-based search directions. For some types of analyses, however, the precision is much more granular than is compatible with these minute differences used by the algorithms. The optional `smallestMeaningfulStep` attribute captures the smallest change in a

design variable that will produce meaningful changes with the analytical methods being used.

4.3.2 Attributes capturing preferences and other information

Beyond the descriptive and interfacing design variable attributes discussed above, the most critical attributes for using the WISE framework for exploring the WISDOM approach are discussed below:

4.3.2.1 Assumption type parameter flag

The `parameter` attribute, like the closing flag, is another simple logical true/false switch to indicate, for input variables, if the variable is an input that the designer has control over or is more of an assumption. This distinction comes into play during search optimization, where the uncertainty of an assumption is captured but it is not made available as a design variable to an optimization algorithm. This may be because the designer truly has no control over the parameter, but the distinction may be driven simply by what phenomena are or are not captured by the analyses employed (and the assumption may become a design variable in later iterations as the system model evolves).

4.3.2.2 Lower and upper bounds

Every design variable can have optional upper and/or lower bounds (simply set to +/- infinity when not in force). These are hard bounds, defined by the `lowerBound` and `upperBound` attributes. As discussed in Section 3.4.3, the bounds are internally treated differently for inputs (enforced as side constraints) as for outputs (enforced as inequality constraints), but there is no difference from the user perspective.

4.3.2.3 Starting lower and upper bounds

While hard lower and upper bounds are dictated by physics or requirements, the domain that is selected as the starting point for a search may be subject to different bounds. This is especially true if the parameter has no hard bounds, when it must still have a defined reasonable and finite domain for the starting point. The `startLowerBound` and `startUpperBound` attributes are what set these bounds for stochastic starting points as described in Section 3.4.2.1.

For variables where the assumption type parameter flag is set, the variable is not part of the optimization search space. Instead, the `startLowerBound` and `startUpperBound` attributes are what set the bounds on the randomly generated uncertain assumption parameters, ω_j , as described in Equation (20).

4.3.2.4 Value

The `value` attribute of a design variable indicates the baseline, default value for that parameter, which is usually the designer's best reasonable guess. As iteration proceeds, design decisions are made, and the parameter that was a free design variable may be set to this fixed value by setting the `fixed` attribute to `true` (or the `free` attribute to `false`). When variables that are outputs of the system model function are set to fixed values, equality constraints are used to enforce their values. With fixed input variables, the tool automatically removes

them from the problem presented to optimization algorithms, reducing the dimensionality of the optimization and greatly increasing the efficiency of future searches.

4.3.2.5 Starting distribution

The `value` attribute is also used as the mode of the probability distribution used for randomly generated input starting points or for randomly generated uncertain assumption parameters. The specific PDF to use is determined by the `distribution` attribute. The starting distribution can be used to direct searching to focus on certain areas of the design space by having relatively more searches start in those regions. Often, however, especially for convex search spaces, the results may be relatively independent of the starting point for certain parameters, in which case a simple uniform distribution or single starting point is sufficient (and the designer does not have to be burdened with thinking about defining the ideal distribution).

For assumption type parameters, however, the starting distribution is the primary tool for capturing the designer and expert knowledge about the assumption, so in these cases a more nuanced PDF can be indicated using the `distribution` attribute. The triangular and PERT distributions, whose merits for this task were described in Section 3.3.2, are implemented using simple functions built with engineering estimates in mind (Sartorius, 2019b). The control over the starting PDF afforded by modifying the `value` and starting upper and lower bound attributes is illustrated in Figure 4-7, where numeric values correspond to the shape parameter, λ , of the PERT distribution (with $\lambda = 4$ for default "pert").

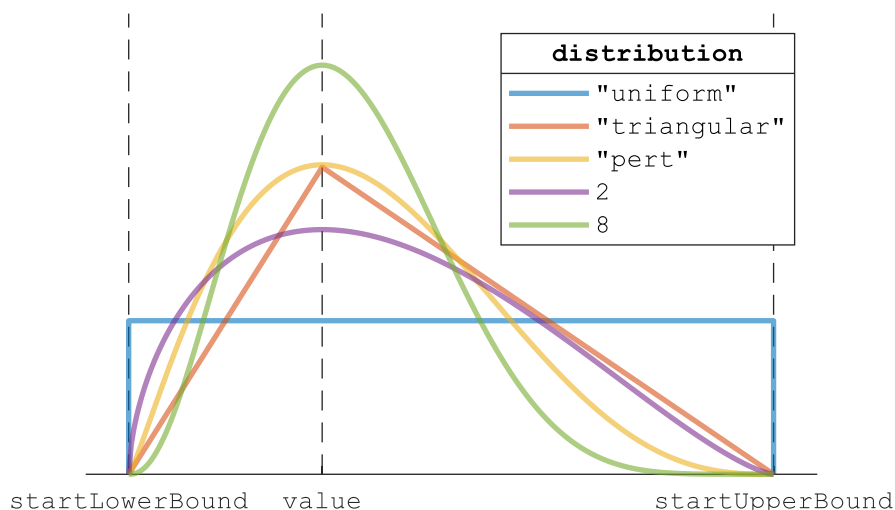


Figure 4-7. Illustration of various possible PDFs based on distribution attribute.

4.3.2.6 Preference map abscissa and ordinate points

As introduced in Figure 3-9, the preference map is one of the core tools for capturing preferences and other types of information. The preference map is a simple polyline / table lookup defined by a set of abscissa and ordinate points

(stored as floating-point vectors in the `prefAbscissa` and `prefOrdinate` attributes) that together define some preference as a function of that design variable's value. In the case of discrete parameters, the `prefAbscissa` attribute is not a floating-point array but rather a cell or string array data type containing the arbitrary discrete values. A `prefMap` attribute allows the option for straightforward and clear setting of both the abscissa and ordinate points simultaneously, optionally along with the preference map node information described in Section 4.3.2.11 below.

The preference ordinate can be expressed as a penalty or, as determined by the `prefType` attribute, as a value, depending on in which sense it is more natural to think about a given parameter. The penalty or value associated with the preference ordinate of a preference map is arbitrary and can be an abstracted point system or something more concrete, such as a dollar cost associated with values of the parameter. The important thing is that whatever scale is used is kept consistent across all design variable objects' preference maps used in a design space.

4.3.2.7 Active or inactive preference map

The `preferenceOn` flag is a simple tool that can be used to easily turn off the preference map of a parameter for a given search. This is equivalent to setting all preference map ordinate points to zero while maintaining the other aspects such as the upper and lower bounds and flagged regions. This can be a useful tool to, for example, answer what-if questions related to customer preferences and desires.

4.3.2.8 Preference map uncertainty

Despite attempts to keep the scale consistent between preference maps used in the same design space, the designer will never be able to achieve perfect consistency. This is not only acknowledged in the WISDOM approach but is embraced and utilized to yield a richer set of useful search results. The `prefUncertainty` attribute captures the level of confidence in the ordinate scale for a given preference map, as discussed in Section 3.4.2.2.

4.3.2.9 Preference map slope

In some cases, it can be useful when defining preference maps to think of the preference ordinate penalties or values in absolute terms. It is sometime more natural to think in terms of the slope or relative slopes of the preference map, as this occasionally better captures designer intent or results in more expected behavior. The `prefSlope` attribute can be used either as a read-only reference, for modifying the slopes of individual preference map segments, or to adjust the slope of the entire preference map.

Slopes of preference map segments tend to be more aligned with what drives real-world designer decision-making behavior when presented with similar regions of the design space. Using finite slopes and ramps instead of hard steps in preference maps also introduces fewer numerical issues and tends to yield more useful WISDOM search results in terms of creating an environment

for the search algorithms to make design ‘decisions’ similar to those that a human designer might make.

4.3.2.10 Preference map smoothing

Most of the time, the designer or other stakeholders do not have highly detailed preferences with any level of fidelity or confidence to warrant anything more granular than a simple polyline function. Still, the discontinuities inherent in a polyline function may in some applications be undesirable, particularly if the discontinuities cause numerical issues with an optimization algorithm, so the optional `prefSmoothing` attribute is available to round off the junctions of the polyline function. It does this by using nested blending functions based on the hyperbolic tangent function (Sartorius, 2016).

4.3.2.11 Preference map node information

Each node in a preference map can be tagged with unstructured additional information using the optional `prefInfo` attribute. This additional information can be as simple as notes to the designer. Another use case can be linking to requirements, which can be quite informal or employing a formal requirements tool such as IBM’s Rational DOORS or other requirements management schemes such as that presented by Glas and Sartorius (2012). This helps make the defined design variables useful sources of documentation as well as reusable artifacts for future projects in cases where similar design drivers are present.

4.3.2.12 Preference map segment warning flags

The optional `prefFlags` attribute allows for attaching warnings to certain segments of the preference map, as introduced in Figure 3-12. The flags can contain rich information, for example notes on the reasons why a given segment is flagged. These flags are automatically raised when examining an individual design point that has a parameter that falls in one of these marked segments. This becomes a valuable tool as the complexity of the design tasks increases, when it can become easier for a design to seem appealing despite some unnoticed detrimental characteristic.

4.3.3 Design variable class methods

The main user-facing design variable class methods are oriented toward helping the user capture information. These methods therefore focus on creating, viewing, and editing `DesignVariable` objects.

4.3.3.1 Plotting and visualization

Most of the key information about a design variable can be visualized using the `plot` method. This method creates a plot of the preference map that is overlaid with other information. Additional plot elements such as the axis labels are used to convey some additional information contained in the object’s attributes. An annotated example screenshot of the plotted information (based off the passenger capacity preference map of Figure 3-9) is shown below in Figure 4-8. The annotations on the plot are created using a customization of the built-in MATLAB data tip capability, allowing users to interactively and graphically

access node information, segment warning flags, or other additional information about the design variable object without overly cluttering the usable graphics area. Implied soft bounds signified by orange triangle markers are discussed in Section 4.5.2.

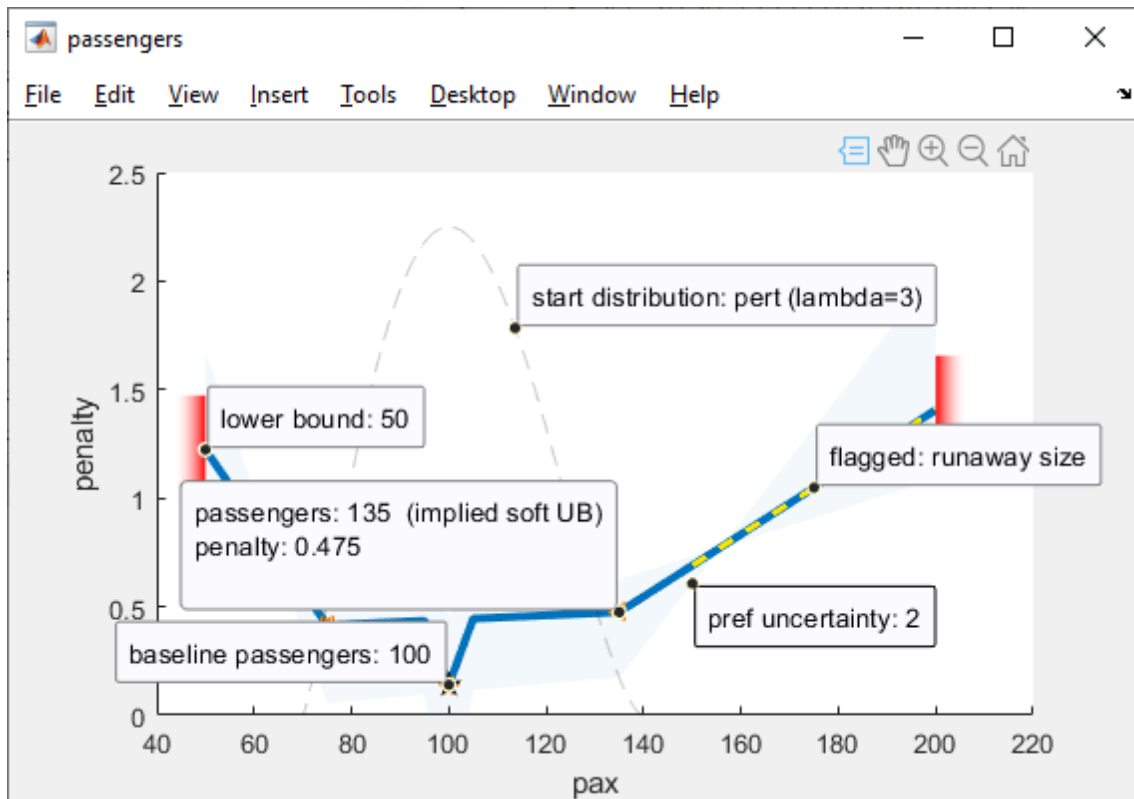


Figure 4-8. Screenshot of design variable class `plot` method.

4.3.3.2 Real-time editing

The typical way to define a `DesignVariable` object is by setting the attributes in a script. The plotting and visualization method discussed above is then an excellent tool for verifying those attributes. However, especially with the preference map abscissa and ordinate points, the cycle of editing a script, plotting the results, further revising, and re-plotting is inefficient and can become tedious. To address this, the `editmap` method creates a basic graphical interface for interactively editing design variable attributes.

There are two elements to the editing interface. The first is the GUI window (Figure 4-9), whose main element is the same visualization as created by the `plot` method. Added to this are areas where certain attributes can be modified, with the visualization updating immediately when any valid change is made.

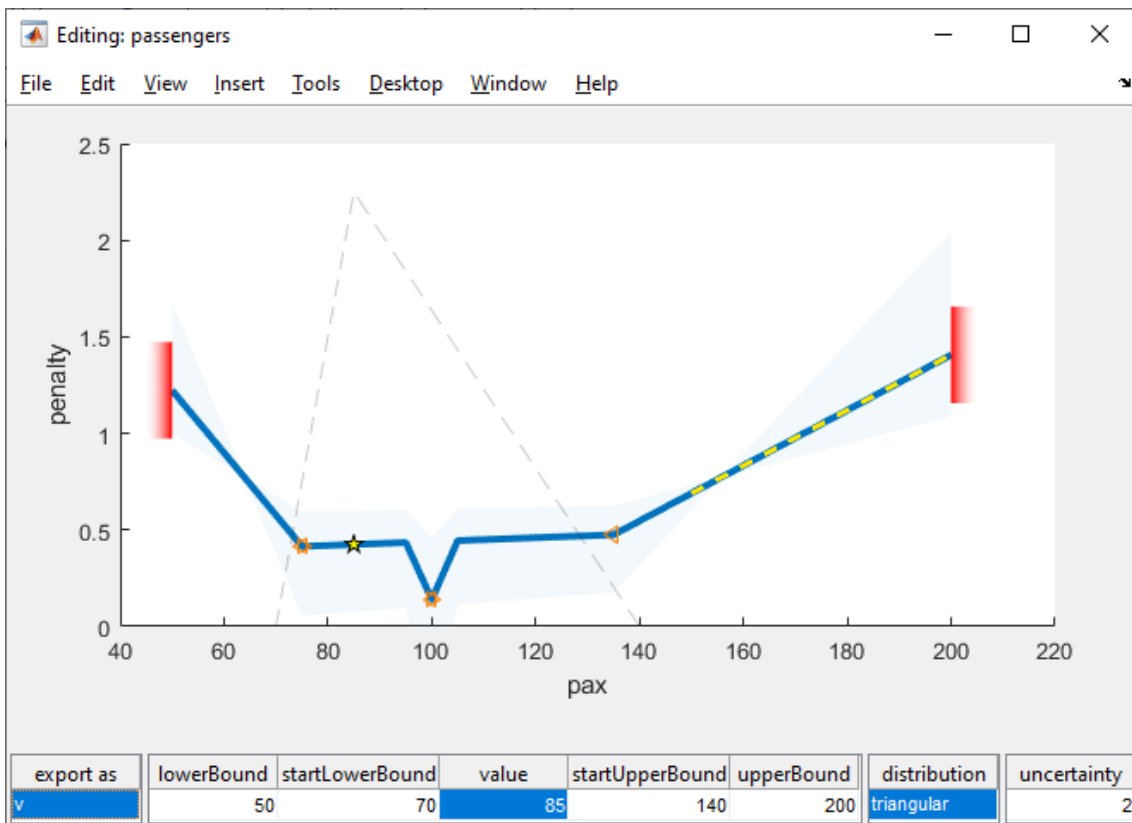


Figure 4-9. GUI window for the design variable `editmap` method.

It is possible to add to the main GUI window a table that would also allow users to make real-time edits to the other attributes related to the preference map such as the abscissa and ordinate points, the additional information attached to nodes, and information for flagged map segments. However, these attributes are not scalar and could have an unwieldy size that is difficult to edit in a relatively static GUI window. Instead, the `editmap` method brings up an additional window of MATLAB's built-in variable editor loaded with a special array (Figure 4-10) that can be used for simultaneously editing some of these non-scalar design variable attributes. The most important capability offered by this approach, though, is the intuitive context menu and keyboard shortcuts that can be used to add or remove columns, creating a very quick and usable approach to growing or shrinking the number of nodes in a preference map.

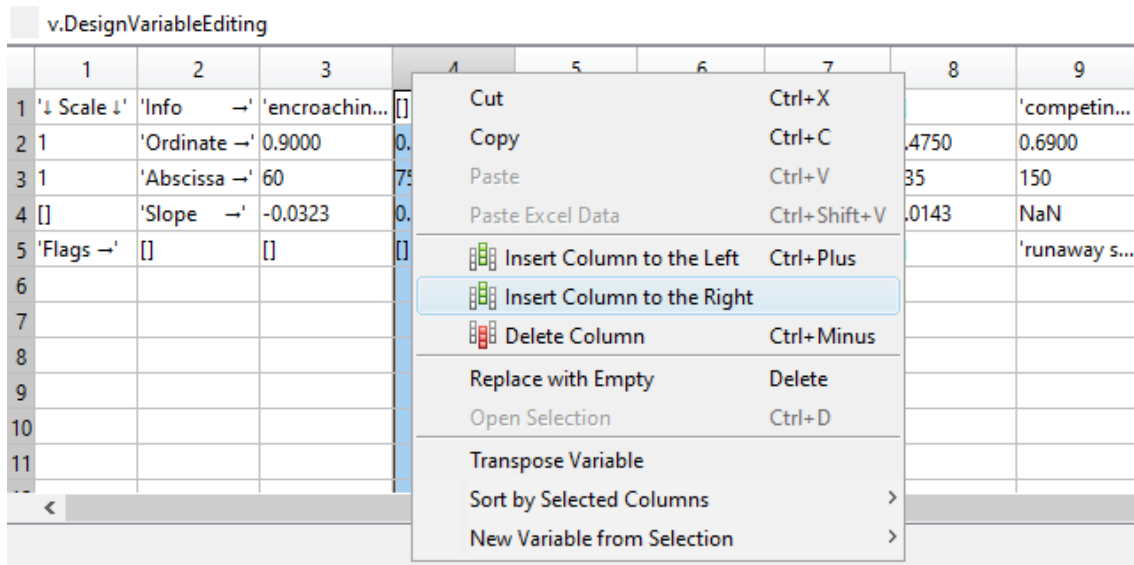


Figure 4-10. Design variable editing using the built-in variable editor.

4.3.3.3 Export

Design variable attributes can be modified by making changes to attributes on the command line, for example, or by using the `editmap` method discussed above. In any case, the changes often need to be recorded, and this is facilitated by the design variable `export` method. Instead of exporting the design variable information as a file to be saved to the hard drive, a simpler approach is taken whereby the export function creates a string that, when executed, recreates the design variable and its attributes and assigns the `DesignVariable` object to a specified variable name.

If no output is specified from the method, it simply copies the string to the system clipboard for easy pasting into a setup script. The export capability is also built into the design variable editing GUI (note the “export as” cell in the lower left of Figure 4-9 above). An example of the resulting exported string is shown in Figure 4-11.

```
% passengers
v = DesignVariable();
v.name = "passengers";
v.label = 'pax';
v.distribution = "triangular";
v.prefUncertainty = 2;
v.prefAbscissa = [60 75 95 100 105 135 150];
v.prefOrdinate = [0.9 0.415 0.435 0.14 0.445 0.475 0.69];
v.prefInfo = {"encroaching on firm's existing portfolio" [] [] ...
             "original requirement specification" [] [] ...
             "competing with established narrow-bodies"};
v.prefFlags = {[] [] [] [] [] [] [] [] "runaway size"};
v.lbvub = [50 70 85 140 200];
```

Figure 4-11. Example code output from design variable `export` method.

4.4 Design space class

While the design variable class creates a framework for capturing and refining information, the function of putting that information to good use in searching the design space falls to the `DesignSpace` class. Because of this segregation of purposes, there are some contrasts between the nature of the two classes, summarized in Table 4-1.

Table 4-1. Comparison of `DesignVariable` and `DesignSpace` classes.

Characteristic	<code>DesignVariable</code>	<code>DesignSpace</code>
User-facing attributes	Many	Few
User-facing methods	Few	Many
Objects in a project	Many	One

As with the design variable class, there are many additional dependent properties, set/get methods, and other background utilities present in the class but not discussed here. Select documentation of the `DesignSpace` class and methods are included in Appendix A3.

4.4.1 User-facing design space attributes

The most important user-facing attributes for the design space class have already been mentioned in Section 4.1.4 and shown in Figure 4-2. Namely, the handle to the system model function is held in the `systemModel` attribute, and the `DesignVariable` object array is held in the `variables` attribute.

One of the issues with storing all design variables in an array is that referencing them must be done using indices in the array. When dealing with dozens of design variables, it can be quite cumbersome to remember which design variable occupies which position in the array, made more complicated by any changes in the design space or setup. To address this, an alternative reference to design variables can be made using the design variables' names and dot referencing of the `v` attribute of the `DesignSpace` class. In this way, there are two alternative ways to access or edit a design variable in a design space, as shown in Figure 4-12, which also illustrates the added usability added by using suggestions and tab completion with this scheme.

```
>> o.variables(5).fixed = true;
>> o.v.range.value
ans =
    2500
>> o.v.
```

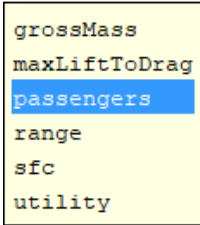


Figure 4-12. Name-based command-line design variable referencing in `DesignSpace` object `o`.

4.4.2 Design space utility methods and attributes

There are also design space class methods that allow for more guided inspection and modification of design variables. Most of these methods, though belonging to the design space class, act simply as wrappers to the corresponding method of the design variable class. Some of these methods have different behavior, however, when operating on an array of design variable objects instead of a scalar object, so they are discussed here instead of in Section 4.3.3.

A significant portion of the other utility attributes and methods of the design space class are primarily used in adjusting the design space, design variables, and search behavior affecting design iteration outcomes.

4.4.2.1 Display

When working with a conventional MATLAB structure or object, the display of the object in the MATLAB command window is usually a simple list of object attributes and their values. However, a couple of factors make this default display behavior undesirable for display of a design space object. The first factor is that most of the information in a design space is in the contained array of design variables, not the design space object itself. So, display of a design space object should bring forward significant information from the design variables. The second factor is that, if displaying an array of objects such as the array of design variable objects, the default display behavior lists attribute names only and not attribute values.

These issues prompted creating custom display methods. Non-scalar design variable objects, instead of the common list display, are displayed as a table with only the most relevant attributes included as columns. The display method for the design space primarily calls the display method for the design variable array. It also adds some additional information such as the system model function name, as shown in Figure 4-13 for a design space for the regional airliner illustration example from Section 3.3.

systemModel: duel_wise_demos.regional_jet.system_model											
	io	frdm	unc	traits	unt	LB	sLB	val	sUB	UB	dist...
range	in	free	2		nmi	400	500	2500	3500	4000	unif...
passengers	in	free	2			50	70	100	140	180	unif...
grossMass	=	free	1	closing	lb	2204.6	4409.2	4409.2	4409.2		unif...
utility	out	free	4		lb-nmi						...
sfc	in	free	1	param	lb/hr/lb		0.45	0.5	0.65		pert...
maxLiftToDrag	in	free	3	param			13	17	22		pert...

Figure 4-13. Tabular display of DesignSpace objects or arrays of DesignVariable objects.

4.4.2.2 Plotting and editing design variables

The `plot` method for the design space class is a straightforward wrapper for the `plot` method for the design variable class. The design space class also implements a method that calls the `editmap` method of a single design variable in the design space as identified by its name.

When plotting the multiple design variables in a vector of `DesignVariable` objects, such as would be contained in a design space, the `plot` method

automatically plots all design variables in a single figure window and by default will also make sure that all of the ordinate axes of the preference maps are using the same scale (examples of which are shown in Chapter 5 Figure 5-2 and Figure 5-7). This is immensely valuable in sanity checking the defined preferences and making sure that their `prefOrdinate` values are defined on an appropriate range. The background color of the plot is also grayed out for variables currently fixed at their value and therefore not active for the current iteration.

4.4.2.3 Processing and utility attribute setting

One approach to keeping a class organized is to implement set and get methods for attributes. Set methods or validators instantly check the validity of what the user is trying to set as a value for an attribute. Get methods are useful when some useful parameter is a function of one or more other attributes and the get method is called whenever that parameter is needed.

However, the process of building up a design space and its design variables can be messy, so strictly checking for errors for every setting of an attribute and enforcing consistency can get in the way of the process. Secondly, and most importantly, many of the useful parameters that would normally be made available using get methods are called many times during a given run of the system model function. These tend to relate to the trimming away of inactive variables, converting a conventional numeric design vector into an input-output system structure, calculating preference values based on system parameter values, etc.

Even for operations that would not necessarily need their own get method to get the needed parameter(s), even simple steps, for example concatenation of all design variable default values, may be run so many times during a search that meaningful speed gains can be made by pre-setting utility parameters. For this purpose, the design variable class implements a `processvariables` method, which sets many useful utility attributes of the design space class for use in other exploration and understanding methods. Part of this method makes sure to run the `runchecks` method on the design variable array, which checks the design variable definitions for completeness and consistency, offering useful warnings and error messages to help the user correct any errors.

4.4.2.4 Toggling active and fixed variables

One of the most basic adjustments that is made from one iteration to the next is adding or removing design freedoms from the design space. Keeping certain design parameters fixed makes for quick exploration and easier understanding, while allowing more design variables to vary facilitates more design refinement. Because it is so common to want to adjust the set of design parameters under current investigation, the `toggle` method provides a quick, command-prompt-based interface for changing the fixed or free state of design variables, as shown in Figure 4-14.

```

>> o.toggle

```

	n	io	frdm	val	unt	distro
range	1	in	free	2500	nmi	uniform
passengers	2	in	free	100		uniform
grossMass	3	=	free	4409.2	lb	uniform
sfc	4	in	free	0.5	lb/hr/lb	pert,L8
maxLiftToDrag	5	in	fixed	17		pert,L3

```

Toggle fixed/free (return to exit): 4

```

Figure 4-14. The design space class `toggle` method.

4.4.2.5 Memoization

Some design space attributes are simply settings for controlling the behavior of other methods, particularly the search and optimization methods. Some of the settings are regarding the use of memoization. Memoization is the technique of storing the inputs and outputs of a process and using the stored outputs instead of re-running the computationally expensive process when identical inputs are provided. Typical implementations of memoization store all unique sets of inputs and outputs, making a trade of storage and memory against potentially very expensive processes. However, in the WISE framework, by far the most common repeated calls to a process are back to back, so the memoization implementations can be simplified to have a short ‘memory’ that can be used not just for expensive processes, but even for only moderately repetitive processes to make meaningful gains in speed.

The processes that are optionally memoized based on their `use{Process}Memoization` settings are calculating preference map values, converting a traditional design vector to an input-output system structure, closing a design, and running the system model function. The latter provides by far the most benefit due to it being the mechanism by which only one system model function call is necessary for analyzing a given optimization search point. Traditional search algorithms run separate functions to analyze objectives and constraints, while the WISE framework uses only one function and, with memoization, one system model function call.

4.4.2.6 Evaluating a design point

At the core of the WISE framework and an essential part of several other methods, including the implementation of the WISDOM approach, are the methods that allow for easily running the system model to evaluate or close a single design point.

The `run` method is the compilation of routines, shown in Figure 4-15, that allows for easy running of the system model function for a given partial design vector of the active (i.e., not fixed) input variables. It automatically populates fixed parameters with default values so that, when the `run` method is called by search optimizations, those algorithms must only deal with the fewest possible number of dimensions in the design space. It also handles turning the design vector into an input-output system structure. The `run` method is also useful as a standalone utility for evaluating the system model at a given design point as defined by the baseline `value` parameters of the design variables.

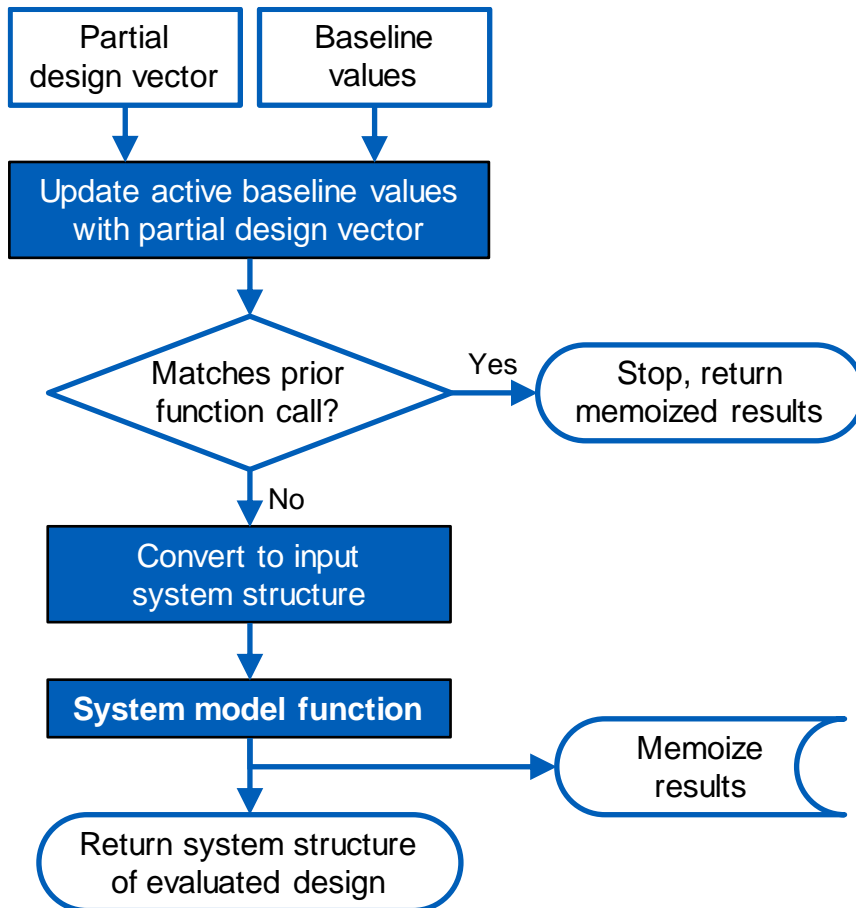


Figure 4-15. Routine used by `run` method for evaluating design points.

When a user is examining only a single design point, however, is it significantly more meaningful if the result makes sense, or in other words, is a closed design where. For example, the estimated gross weight input should equal the calculated gross weight output. For this purpose, for designs that have these types of closing variables, the `closedesign` method is useful for analyzing a given design point (reverting to the `run` method if there are no closing type variables).

4.5 WISDOM approach implementation

The bulk of the WISDOM approach methodology presented in Chapter 3 is brought to practice as methods of the design space class as described in this section.

4.5.1 Batch analysis for sweeps and Monte Carlo analysis

Instead of evaluating just a single design point at a time, there are use cases for evaluating many design points at once. This is one of the instances where significant advantage can be taken of MATLAB's vectorization discussed in Section 4.1.3.6. The first use case is for sweeps of parameters, usually to make a simple plot of sensitivities to one design variable, though vectors of multiple variables can be provided to the `batchrun` method for this purpose. The same method can also be used for running Monte Carlo simulations to understand the impacts of the assumption type parameters that are active in the design space, as discussed in Section 3.3.2. There is also a mechanism to run all combinations of the extrema of assumptions to ensure that the full spectrum of possible results is captured.

In all cases, many design points are run using vectorized inputs to the system model function. This can still be done even if there is a closing type of design variable present via the automatic implementation of a vectorized root finding method, in this case a vectorized bisection method (Sartorius, 2015). If the system model function is not built in a vectorized way, or if there is more than one closing type of variable, then other approaches must be used. For parameter sweeps, the `closedesign` method can be used in a loop, and for Monte Carlo analysis, the `searching` method implementation of WISDOM optimization discussed below can be used with all parameters fixed except for the uncertain stochastic assumptions introduced in 3.4.2.3. This is a special simplified case of WISDOM searching that results in a Monte Carlo simulation.

4.5.2 WISDOM searching and optimization

The primary impetus for creating the WISE framework was to implement the WISDOM approach that is the primary subject of this work, and the design space `searching` method is the routine that executes it. Once a design space has been set up, with design variables and system model function set, the `searching` method implements search optimization that minimizes the objective function subject to constraints as described in Section 3.4. If running the method with only a single starting point, the simplified unvaried objective function of Equation (17) is used. With multiple starting points, the routine selects randomized inputs according to the specified starting distributions, generates randomized preference map uncertainty scaling factors, then minimizes the objective function defined in Equation (20) for each case one by one in a loop, as shown in Figure 4-16.

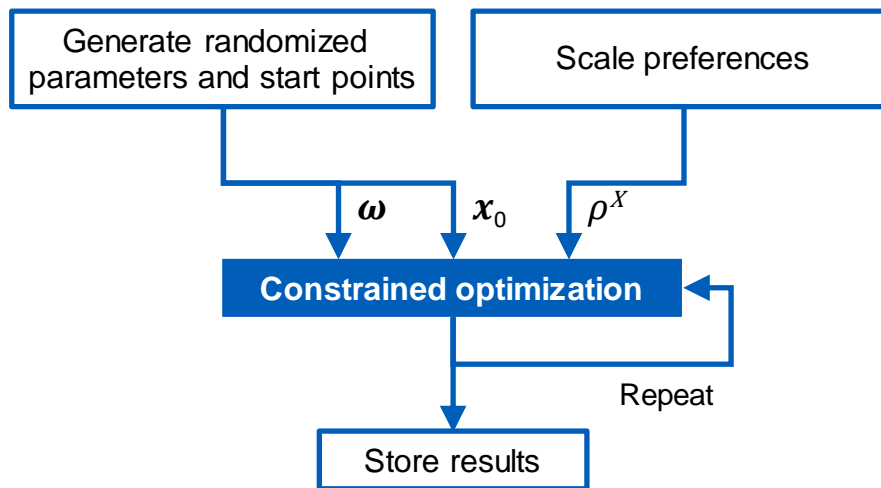


Figure 4-16. Operations used by searching method to generate a variety of optimized designs.

The number of WISDOM constrained optimization search cases, and therefore the extensiveness of variation introduced by using different combinations of starting points, uncertainty scaling factors, and stochastic assumption parameters, is specified by the designer and is a compromise between computation time and breadth of exploration. Since this can be a time-consuming method, especially with an expensive system model function or with many starts to run, features are in place to back up, interrupt, and restore large runs so that computational work is not lost.

Because the WISDOM approach is relatively agnostic to the optimization algorithm used (though it should be a deterministic local search algorithm), the searching method can easily use different algorithms. Algorithms implemented include those supported by the Optimization Toolbox (*Optimization Toolbox (Version 8.2)*, 2018) `fmincon` function, such as interior point (default), sequential quadratic programming, or active set algorithms. A constrained Nelder-Mead simplex search algorithm (D’Errico, 2012) is also implemented as an option (with `fmincon` used to quickly find a feasible starting point for algorithms that require it).

The behavior regarding closing the design can also be changed as an option for the method, which can affect speed or results. Sometimes stability can be increased by making sure that search start points are closed design points. There may also be cases where it is desirable and worth the significant added computation time to close every single design point analyzed at every internal iteration of a search algorithm, so this option is also present.

Another setting that can affect both the speed of searching and the results of the search itself is the so-called “ratcheting” functionality. This is an additional alternative approach to WISDOM searching whereby ‘soft’ lower and upper bounds are automatically inferred based on changes in slope of preference maps. Though normally unused and not required to yield useful results, if the design space `ratchetBounds` attribute is set, then the WISDOM method will be modified such that if the starting point for a parameter is on the valid side of an inferred bound, then that inferred bound will be enforced for the search with

that starting point. This is generally controlled at the design space level, but the behavior can be turned on or off individually for design variables as well.

4.5.3 Methods for understanding WISDOM results

The output of the `searching` method WISDOM implementation is a large set of designs optimized using slightly different starting points, objective functions, and assumptions. The design space class has some methods designed to help process these results to help the stakeholders harvest useful information in order to facilitate design decisions and the next iteration.

4.5.3.1 Results structuring and handling

One important aspect of design iteration is documentation, and part of that documentation is keeping track of past iterations, their results, and the methods and assumptions that led to those results. To support this, the design space class has some features related to storing not only the results of the current iteration, but also for storing results for reference.

A structure of results as generated by the WISDOM `searching` method is automatically stored in the design space `results` attribute. This is a structure that, in addition to containing some information related to the run that generated the results (such as a timestamp, search settings, and run duration), contains several sets of design points and related data (such as active preference map segment warning flags) filtered by different criteria. The standard fields for results are:

- `all`: all design points.
- `valid`: design points from searches that converged successfully to closed designs.
- `dominant`: valid design points that are not Pareto dominated based on unscaled preference map values.

Being a structure, additional fields can be added as more post-processing routines filter out other interesting sets of design points.

To help track and document iterations, the `sendresultstocache` method sends the current results to the stack stored in the `cachedResults` attribute, along with date, time stamp, and optional iteration notes. The `results` attribute is then automatically cleared and made ready to store the results of the next iteration `searching` run. Alternatively, the `appendresults` method can be used to append a set of results to another to make a single larger set of results.

Finally, the `buildresulttable` method turns a set of results into a data table that lists all design points and all design parameters. This table is not only useful for displaying design points in MATLAB, but an additional formatted table is created that easily pastes into a spreadsheet, including with useful headers. It is often the case that a spreadsheet, especially with, for example, cell coloring based on parameter values, can be an excellent approach to understanding and filtering a finite set of different design options.

4.5.3.2 Visualization

Visualizing a multi-dimensional design space can be quite useful for enhancing the understanding of the system for stakeholders. The built-in routine for doing this in the WISE framework is the `scattermatrix` method. This method implements two separate categories of visualization, but both are based on a matrix of two-dimensional plots such that every parameter of interest is plotted against every other. An additional useful element is plotting histograms of each parameter on the diagonal of the matrix.

Sometimes the density of resulting design points can be valuable, in which case a corner plot, as implemented by Adler (2018), is available for visualizing the areas of the design space where search results may tend to cluster. The primary option, however, is a scatter matrix, whereby design points are plotted as icons whose color and size can be used to indicate any additional attributes of the design points, for example to indicate designs with preference map segment warning flags, designs with better preference values, etc. Significant additional value comes from this type of plot via interactively brushing the data, which lets stakeholders get an extra intuitive understanding for the higher dimensions of the design space. Scatter matrix examples are shown in Chapter 5 in Figure 5-3, Figure 5-4, and Figure 5-12.

4.5.3.3 Clustering

From a large set of results, often a key step in iteration and decision making is reducing that set to only a handful of representative design points from which a selection or selections can be made. In rare cases, a useful first step in paring down a set of design points to a manageable few can be to simply eliminate design candidates that are Pareto dominated by other designs, as is done automatically as part of the `searching` method. This is rarely highly fruitful in significantly reducing the set, however, as the implementation of the `searching` is normally quite efficient in returning primarily Pareto-efficient sets. In other cases, just looking at points in the design space where many searches converged to, i.e., the ‘popular’ areas of the design space, can be useful.

Other approaches to yielding a small and succinct set of design points hinge on identifying clusters of design points. There are many existing clustering algorithms, and several were evaluated for use with the types of data sets that are produced by this method. In this case, effectiveness of an algorithm is based on clustering design points into groups that are qualitatively different from each other, what might be called different ‘subspecies’ of designs. The most useful algorithms are also those that preserve outliers, since outliers are frequently interesting design points.

The off-the-shelf algorithm examined yielding the best results regarding effectively identifying clusters of design points is Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) (Sorokin, 2018). However, this algorithm still requires some tuning for each use case, and the field of clustering in general is geared more toward training models that can identify future points rather than focusing solely on the data set at hand. So, some other clustering ideas were explored.

5 Example Cases

To round out the explanation of the WISDOM technique and its implementation in the WISE framework, two example cases are presented to illustrate the usage, the workflow, and most importantly, the artifacts produced and results. The regional airliner introduced in Chapter 3 to illustrate the types of information captured by the WISDOM technique is taken through an iteration as the first example case. The second example case, a more rigorous test of the approach with more parameters and complexity, is the early investigation of a possible redesign of the wing tip of an existing narrow-body airliner, including the possibility of adding a winglet.

5.1 Regional airliner

Recall the regional airliner market analysis and initial sizing scenario introduced in Section 3.3. In this fictional case, a manufacturer of small business jets is seeking to enter the regional airliner market with an all-new aircraft design. For this illustration, a simple system model function is built to analyze a given defined aircraft, and a design space is set up with design variables and preference maps. The results of an iteration show various artifacts useful for gaining insight and driving decisions, particularly when compared to a more traditional optimization method.

5.1.1 System model

The Breguet range equation and the fuel fraction sizing analysis method used have been described in Section 3.3.1, including the empty weight fraction trend parameters, A and C . The payload mass is assumed to be 95 kg per passenger, including baggage, cargo, cabin furnishings, etc. The crew mass is assumed to be 80 kg per crewmember, which includes two flight deck crew plus a number of cabin crew that is an affine function of the number of passengers based on the flight attendant requirements of 14 CFR § 121.391.

The system model implements a calculation of the fuel fraction based on the Breguet range equations of Equation (10) and an IFR (instrument flight rules) mission profile, shown in Figure 5-1, that includes a diversion of 200 nautical miles plus a 45-minute reserve.

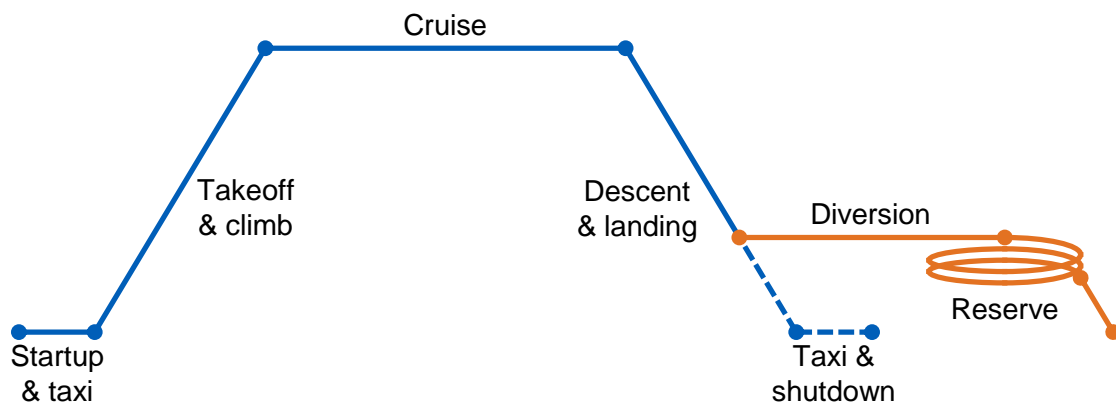


Figure 5-1. Simple mission profile for regional airliner system model.

The mission profile model uses certain fixed assumptions for flight speed, fuel margin, weight fractions for non-cruising, non-loitering flight phases, and other parameters as listed in Table 5-1. There is no range credit for climb or descent.

Table 5-1. Mission profile parameters and assumptions.

Parameter	Value
Cruise speed	450 KTAS
Cruise & diversion condition	$\frac{\sqrt{3}}{2} L/D_{max}$
Reserve condition	L/D_{max}
Startup & taxi mass fraction	0.98
Takeoff & climb mass fraction	0.985
Descent & landing mass fraction	0.99
Additional fuel margin	6%

Other important elements of the regional airliner system model are input handling to allow usage as a script as discussed in Section 4.2.3 and the setting of additional simple fields that represent useful information for easy access later, for example useful load or empty weight. The full source code for the system model and mission profile are in Appendix B1.

5.1.2 Design variable definitions

Only six `DesignVariable` objects must be defined for this simple example case. A tabular display of these variables was shown in Figure 4-13. Full source code for the design space setup and design variable definitions is in appendix B2, and a graphical display of the design variables in the design space, as produced by the `plot` method, is shown in Figure 5-2. Note that the plots with grayed-out background signify design variables that are fixed at their nominal value for the iteration. In this case, the parameter assumption type variables are kept fixed for the initial searching runs. Also important to the visualization of all the design variables together is that all of the preference maps use the same scale for all the ordinate axes.

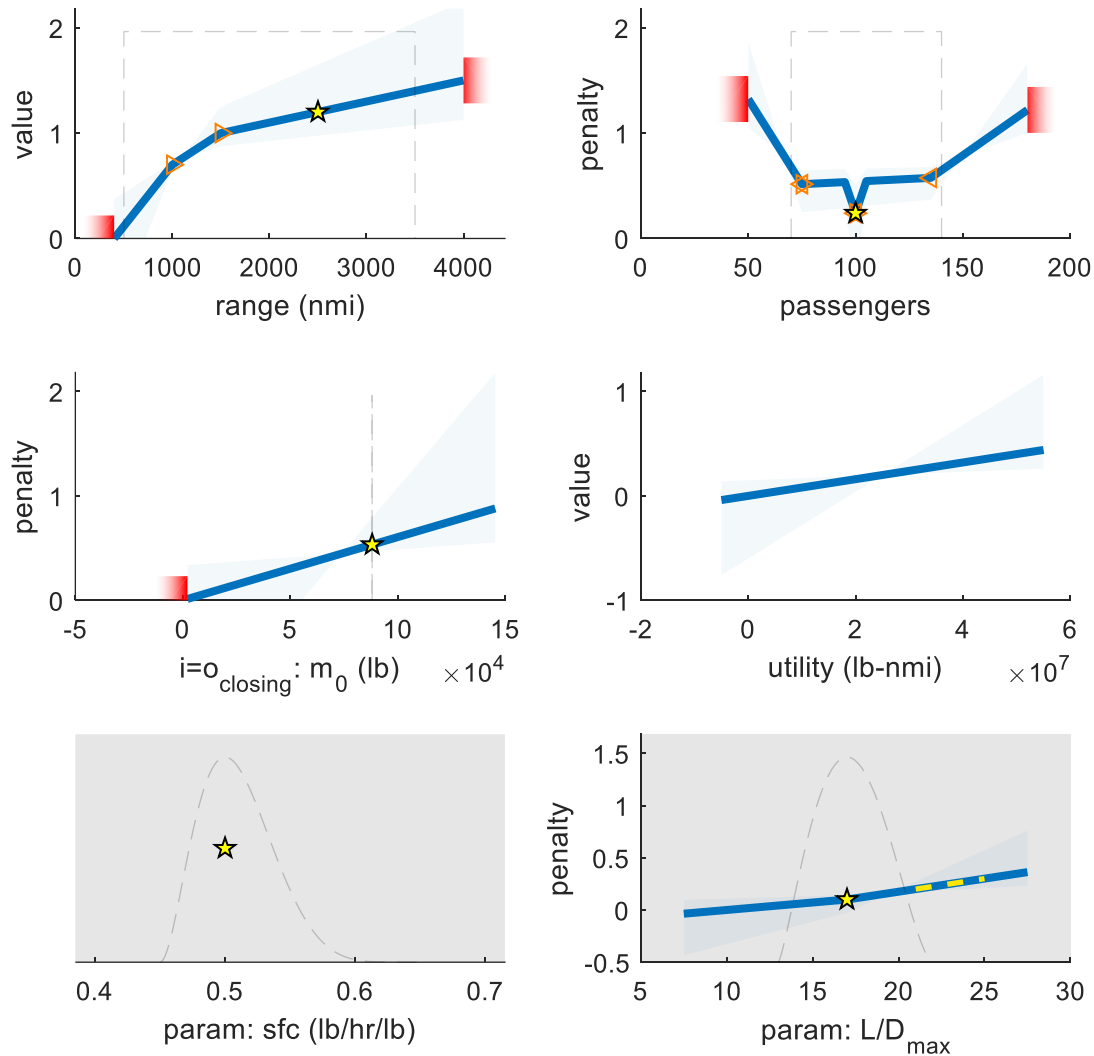


Figure 5-2. Regional airliner design variable plots.

5.1.3 Searching with WISDOM approach

After the setup of the design space, searching the design space using the WISDOM approach is a straightforward single call to the `searching` method, e.g. `o.searching('nStarts', 500)`. For these five hundred searches and the default searching settings, 235 system model function evaluations were required on average for each search, for a total run time of a couple of minutes.

The design space in this example is particularly small, so many traditional options for visualization are possible, for example a three-dimensional scatter plot. The `scattermatrix` method, which easily scales up to higher dimensions, yields the array of plots shown in Figure 5-3 below. A handy step can be to use the custom clustering algorithm or HDBSCAN to sort resulting design points into their various ‘subspecies.’ In the case of Figure 5-3, color is used to signify membership in different clusters as determined by the custom clustering algorithm described in Section 4.5.3.3 (outliers are light gray).

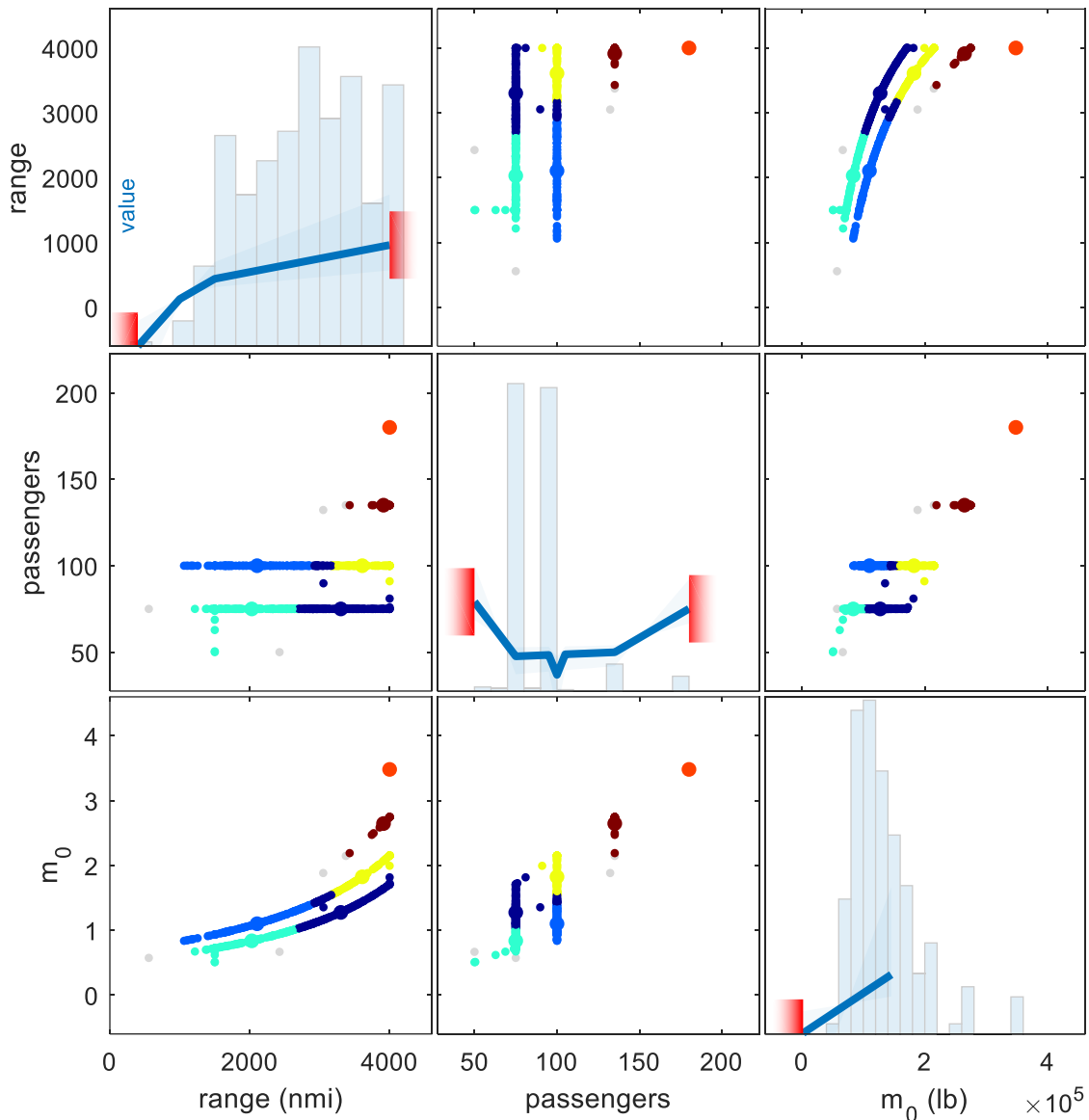


Figure 5-3. Regional airliner results visualization with clustering. *The main diagonal shows the preference maps for reference and histograms to indicate design point distribution and density.*

While this type of visualization can be useful for gaining a greater understanding of the nuances of the design space, sometimes the best tool for facilitating a conversation between stakeholders is a fair and quantitative comparison between a small number of design alternatives. In the scatter matrix in Figure 5-3 above, marker size is used to indicate the design points that are most representative of their respective clusters as measured by being closest to the mean position of the cluster.

Utility methods make it straightforward to turn a set of design points into a data table, including in a format that can be easily exported outside of MATLAB, for example to spreadsheets. The type of tabulated data of the cluster-representative design points such as in Table 5-2 below can be a useful artifact to facilitate discussions and decision-making between stakeholders. Additional columns in exported data (not shown) include preference map segment flags,

information on whether the design point is closed and valid, and if the design point is Pareto dominated by another design point.

Table 5-2. Table of design points representative of regional airliner clusters.

$\sum p(S(x))$	Range (nmi)	Passengers	Gross weight (lb) $\times 10^3$	Utility (lb-nmi) $\times 10^6$
-0.58	3609	100	182.4	76
-0.47	2105	100	110.0	44
-0.39	3302	75	127.5	52
-0.24	2029	75	83.6	32
-0.09	3912	135	264.6	111
0.72	4000	180	348.1	151

Note that even when the assumption type parameters are allowed to vary in a search, this type of clustering analysis and the representative design points are still useful and can provide insight. With large searching sets and PDFs of the assumptions that have a small standard deviation, the cluster-representative design points will be close to a fair comparison using similar assumptions. Or, sometimes different assumptions will lead to a multimodal result, whereby the assumptions are a determining factor in the optimal subspecies (i.e. cluster membership) for a search.

Since the assumption type parameters were fixed above, variations in the found optimum results came only from variation in starting points and relative uncertainty scaling of the preferences. When the uncertain assumption type parameters for specific fuel consumption and maximum lift-to-drag ratio are free to be randomized for each search, significantly more variation is introduced. The resulting design points for 2000 searches is shown in Figure 5-4. For this visualization case, color is used to visualize the sum of preference penalty values (unscaled by uncertainty factors).

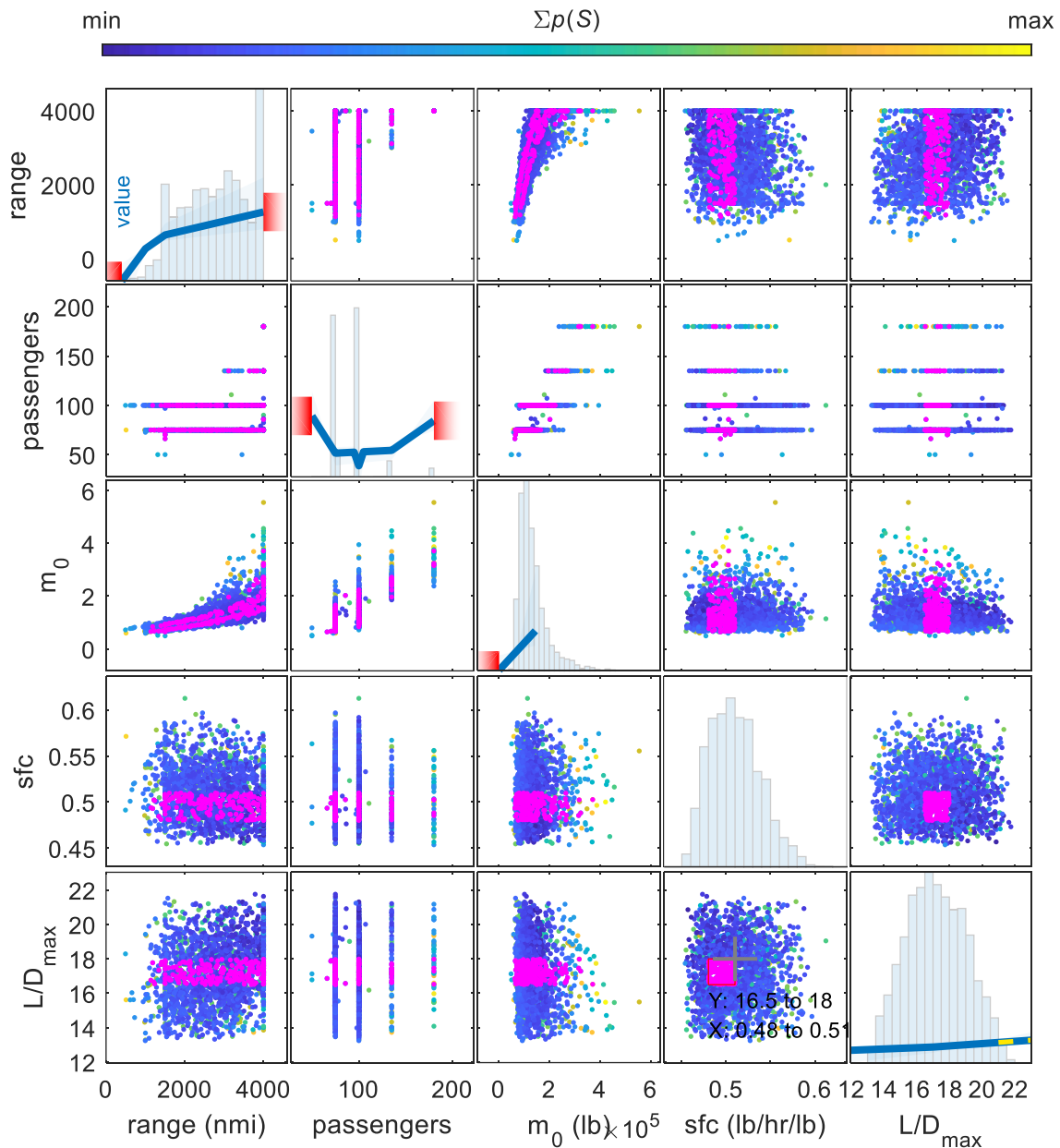


Figure 5-4. Regional airliner results visualization with data brushing.

Both because of the significant additional variation and because it is now a higher-dimensional design space to visualize, data brushing becomes a powerful tool for interactively exploring the design space and gaining a deeper understanding. As an example, the brushed data (magenta markers) in Figure 5-4 highlight design points resulting from a narrower range of possibilities for technical assumptions for specific fuel consumption and maximum lift-to-drag ratio.

5.1.4 Comparison to value function methods

As discussed above, one of the key results of applying the WISDOM approach to this problem is a small set of alternative design points that are all valid designs that could each be a viable result, depending on preferences and requirements. Each of these possible designs, such as those in Table 5-2

above, is optimized for the given circumstance. The same regional airliner example case can be used in a much more conventional optimization approach for an illustrative comparison to using the WISDOM approach for early design space exploration and optimization.

One of the most common value function methods for multi-objective optimization is the weighted sum method (as discussed in Section 2.4.2). The WISE framework can be easily used for a weighted sum approach by simply ensuring that all preference maps are linear and that all assumption type parameters are fixed. Using a least squares fit to linearize the preference maps, the optimum result (no matter the search starting point) is an aircraft with a capacity of 50 passengers and a 4,000 nautical mile range. Preference map uncertainty scaling can be used to mimic the full range of possible 'tunings' of the parameter weights for the weighted sum approach. Even when doing so, all resulting designs, as is a common pitfall with the weighted sum method, are still pinned against the hard constraints for passenger capacity (50 or 180 passengers) or range (400 or 4,000 nautical miles). The cluster-representative designs are shown in Table 5-3 below (short-range, high-capacity designs were infrequent and considered outliers by the clustering algorithms). To drive the optimum to any sort of compromise solution, either significant effort must be put into careful fine-tuning the objective weights, or the hard constraints must be changed.

Table 5-3. Regional airliner design points from weighted sum method.

Range (nmi)	Passengers	Gross weight (lb) × 10³	Utility (lb-nmi) × 10⁶
4000	50	124.4	42
4000	180	348.1	151
400	50	38.5	4.2

5.2 Wing redesign with winglet

The regional airliner example case provides a simple illustration of the mechanics of the approach and the framework, and the case is simple enough to allow straightforward inspection of the results to verify that method yields sensible designs. The next example case, originally presented by Sartorius and Hornung (2018), goes further towards testing and demonstrating some of the advantages of capturing more information early and putting it to use with optimization. The scenario in this example case is that a wing redesign is being considered for an existing, in-production narrow-body airliner.

The objective of the study is to determine, with minimal expended effort, what a redesigned wing might look like, what gains may be possible in performance, and what compromises are likely necessary to achieve those gains. This scenario is similar to the Boeing 737-700 airliner, and wing geometry for the example case were measured from 737-700 drawings (Jackson et al., 2004) for both the baseline wing and for the version of the 737-700 that features a winglet (Figure 5-5).



Figure 5-5. Wing of Boeing 737-700 narrow-body airliner with winglet.

5.2.1 System model

The modeling used for this example case is simple and does not capture the full physics of the system, as is typical in early design studies. While simple conceptual design methods and models are used to estimate other parameters of interest such as weight or parasite drag, the employed vortex lattice method is a relatively fast and simple analysis method that can still effectively capture the effects on induced drag and aerodynamic loads due to variations in three-dimensional wing geometry and twist. The primary analysis for lift, induced drag, and loads is therefore based on the vortex lattice method as implemented in Athena Vortex Lattice (AVL) (Drela & Youngren, 2017). For this study, the modifications to the wing are limited to the tip section that is entirely outboard of the aileron and slat. Figure 5-6 shows the half-wing AVL model of the baseline

wing along with arrows marking the wing station outboard of which modifications are considered.

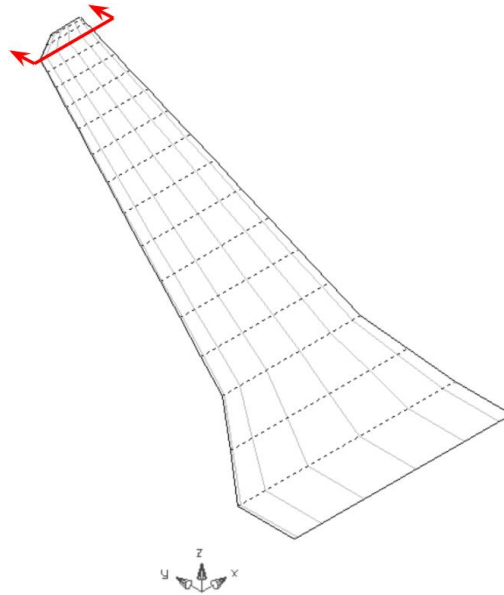


Figure 5-6. Narrow-body airliner baseline half-wing modeled in AVL. *The tip region subject to redesign is marked.* (Sartorius & Hornung, 2018, p. 6)

Since it is a very early design study, additional simplifications are made for the modeling. The key results of the modeling are the changes in performance rather than absolute performance, so modeling that is somewhat inaccurate is acceptable if it appropriately captures the correct trends. One simplification is that the vehicle is analyzed at a single flight condition, flying at Mach 0.78 at 38,000 ft at a weight of gross weight minus five tonnes (about one third of fuel capacity). Two conditions are analyzed with AVL at this flight condition: one at level cruise lift coefficient to determine induced drag and another at a high load factor to determine bending loads. Other simplifications are that airfoils with zero camber are used for the vortex lattice model and the main wing is untwisted from root to tip. For numerical stability, winglets with span less than five centimeters are not included in the vortex lattice model.

The main design freedoms available to adjust are the wing tip panel trapezoidal geometry, the winglet area, and the winglet rigging (twist and incidence). The weight change of the wing is based on an assumed weight per unit planform area, and parasite drag change is based on an assumed average section drag coefficient. Table 5-4 shows a summary of the modeling methods, with the source code in Appendix B3.

Table 5-4. Wing redesign study modeling summary.

Calculation	Method
Lift & aerodynamic loads	Vortex lattice (AVL)
Induced drag	Vortex lattice (AVL)
Parasite drag, fuselage and tail	Constant
Parasite drag, wing & winglet	Fixed drag coefficient \times reference area
Wing weight	Parameter \times reference area
Winglet weight	Parameter \times reference area

Other key modeling outputs of interest are the change in drag at the cruise condition and the change in the root bending moment at the high load factor condition. At the limit load factor condition, some bending relief by the masses of the wing tip and winglet is accounted for, but no alleviation due to deflections is included.

5.2.2 Design variable definitions

A total of sixteen design variables were defined for this example case, shown in Figure 5-7 below.

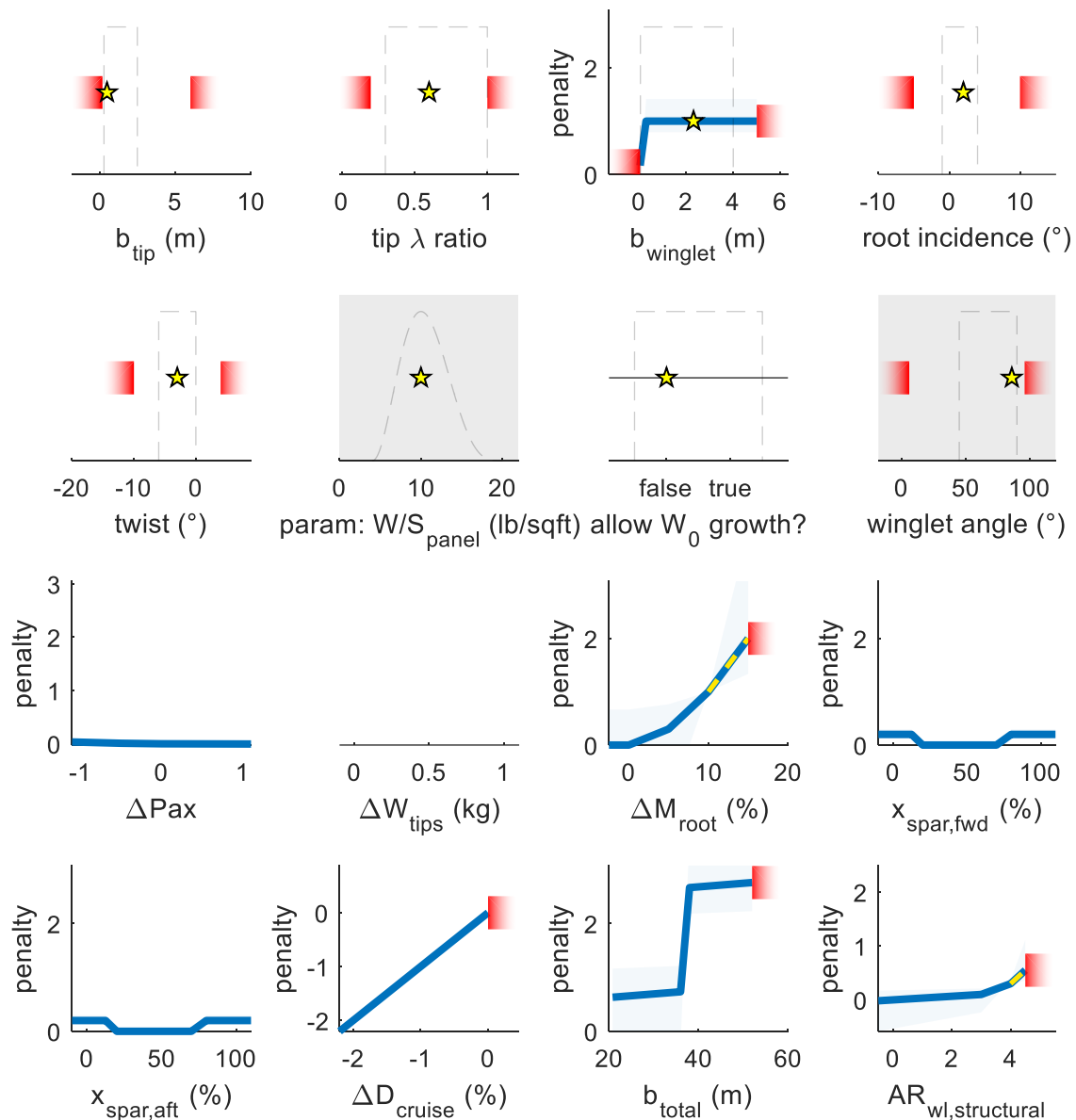


Figure 5-7. Wing redesign design variables.

This case exemplifies the versatility of the structure of the design variable class, as it contains a mix of design variables that are inputs and outputs, with and without preferences attached. Many of the inputs with no preferences attached, such as those for the winglet rigging, are simply to allow for any resulting design points presented to stakeholders to be optimal for the given conditions. Many

inputs also feature upper and lower bounds simply for keeping the analysis in a reasonable domain where the analysis will be computationally stable. Other design variables are kept fixed (in addition to some fixed parameters hard coded into the system model). Some parameters are simply included as a likely output variable of interest to bring to the surface for discussions without any specific preference information or bounds attached (hence the blank plot for the change in the weight of the wing tips). Some of the more interesting design variables for this example are discussed in more detail below. Source code for all design variable definitions is in Appendix B4.

5.2.2.1 Change in cruise drag

For this case, all preference maps are drawn keeping in mind changes in cruise drag as an explicit unit of value or penalty for the ordinate. A reduction of cruise drag of one percent is used as one unit of ‘currency’ when recording the preference. That is why the preference map for change in cruise drag is not only linear with a unit slope, it also is the only preference map with an uncertainty factor of unity, indicating no uncertainty, as cruise drag change is the anchor for other preferences.

5.2.2.2 Loads and structures

One of the reasons that the system model can be built very quickly with very little effort is because the system model features no structural analysis at all. Instead, captured preferences for winglet structural aspect ratio (aspect ratio using structural span instead of span normal to the freestream) and wing root bending moment (Figure 5-8) act as surrogates for any structural modeling. Increases in root bending or winglet structural aspect ratio are progressively penalized. Any root bending moment increase of 10% or more is flagged as highly likely to require structural redesign effort that is more major than might be worthwhile in scope of the project.

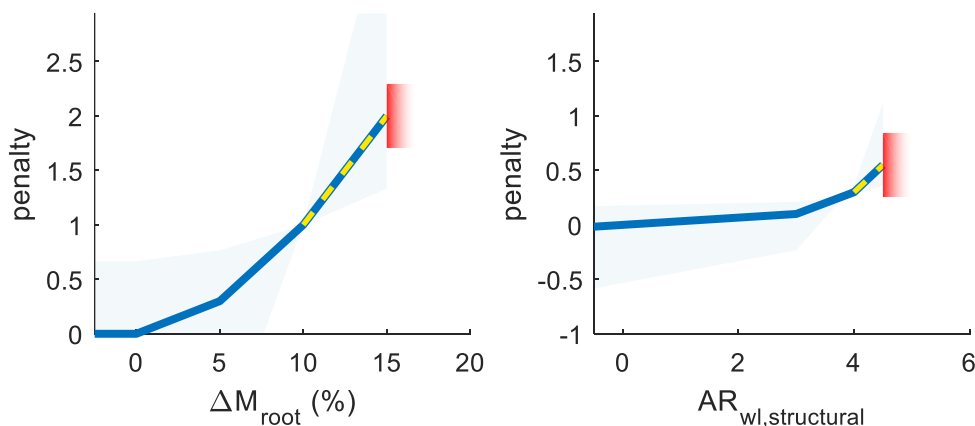


Figure 5-8. Preference maps for root bending moment and winglet structural aspect ratio used as surrogates for structural modeling.

5.2.2.3 Spar alignment with winglet root

It is desirable that the major structural elements in the wing do not have to follow convoluted load paths in the redesign. To capture this desire, preference maps are attached to locations on the root of the winglet where the forward and

aft main wing spars align (Figure 5-9). In the case that a main wing spar is close to being aligned with the thicker areas of the winglet root section, the slopes of the preference maps in this region help drive the search toward designs with better alignment.

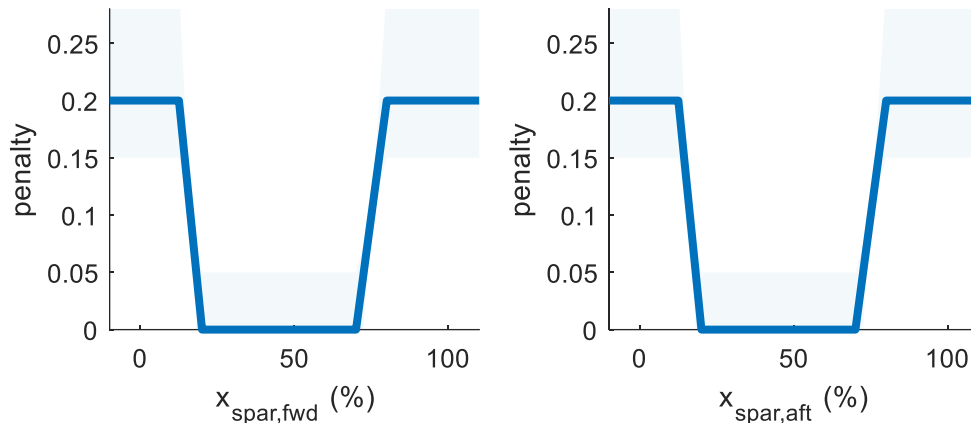


Figure 5-9. Preferences driving toward favorable structural alignment and load paths.

5.2.2.4 Weight changes

At the outset of the study, it is not clear if an increase in gross weight of the aircraft is a possibility in the scope of the project. The technique allows for moving forward with both possibilities given equal consideration. A discrete design variable (Figure 5-10, left) that captures this is a simple input flag determining if any additional wing weight should result in a direct increase in gross weight or, alternatively, a decrease in useful load, as measured in passengers. A gross weight increase is indirectly penalized in the change in root bending discussed above, while the preference against any loss in useful load is captured in its own design variable (Figure 5-10, right).

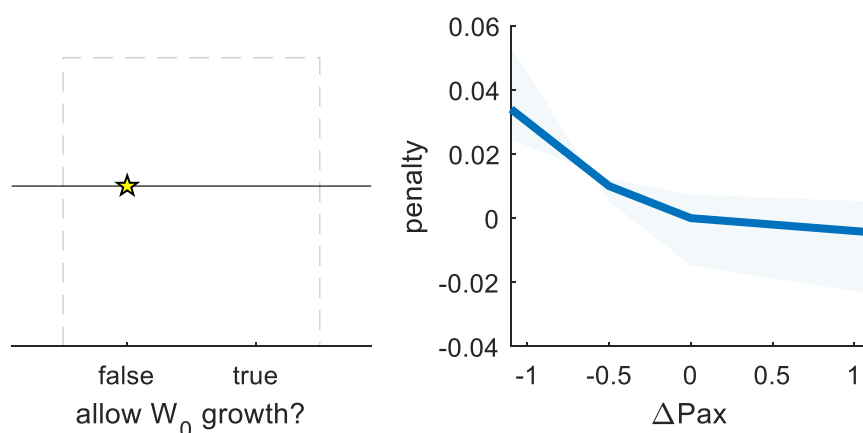


Figure 5-10. Capturing binary possibilities of gross weight increase or useful load decrease.

5.2.2.5 Span

If a winglet is present, the aerodynamics of the wing should be allowed to be optimized to the furthest extent possible, so there are therefore no explicit

preferences on the winglet geometry. The exception that the winglet span preference map (Figure 5-11, left) captures is that if considering designs that have only a very tiny winglet, it is desirable to simplify the design and reduce part count by not having any winglet at all.

For various reasons, ranging from hangar space to roll and yaw inertia to tip strike angle, a shorter overall wingspan is desirable, all other factors being equal. Hence, the preference map on overall wingspan (Figure 5-11, right) has a gentle positive slope to capture this incentive. In addition, the overall aircraft wingspan preference map captures the significant penalty for exceeding the ICAO Code C span limit of 36 meters (and with a hard bound at the Code D limit). What is captured is the designer instinct that an optimized aircraft that features a wingspan only slightly above 36 meters is unlikely to be a sufficiently superior product to one with wings 'clipped' to be within ICAO Code C.

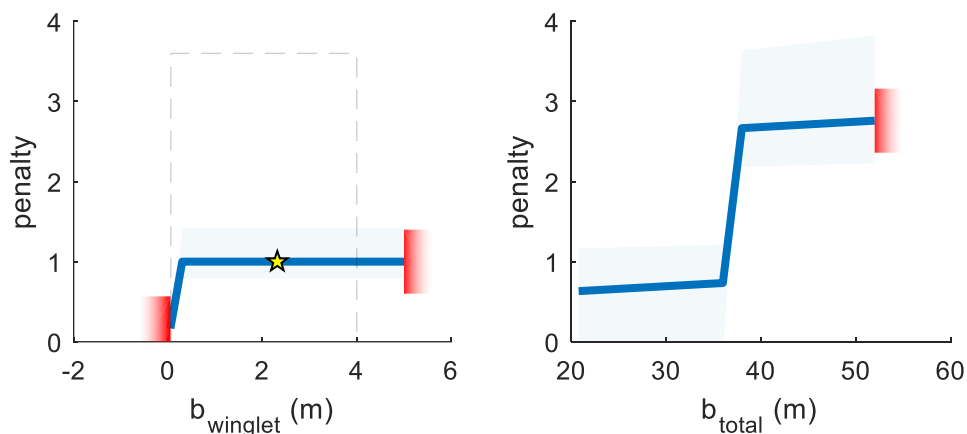


Figure 5-11. Preferences on winglet and total span.

5.2.3 Searching results

This scatter matrix visualization of the results for the wing redesign study is shown in Figure 5-12. Clustering analysis shows some distinct subspecies of interest, with some interesting variation within clusters as well. Note that parameters related to the weight changes discussed above in Section 5.2.2.4 were not included for the clustering analysis, so each cluster contains designs both with and without gross weight increases.

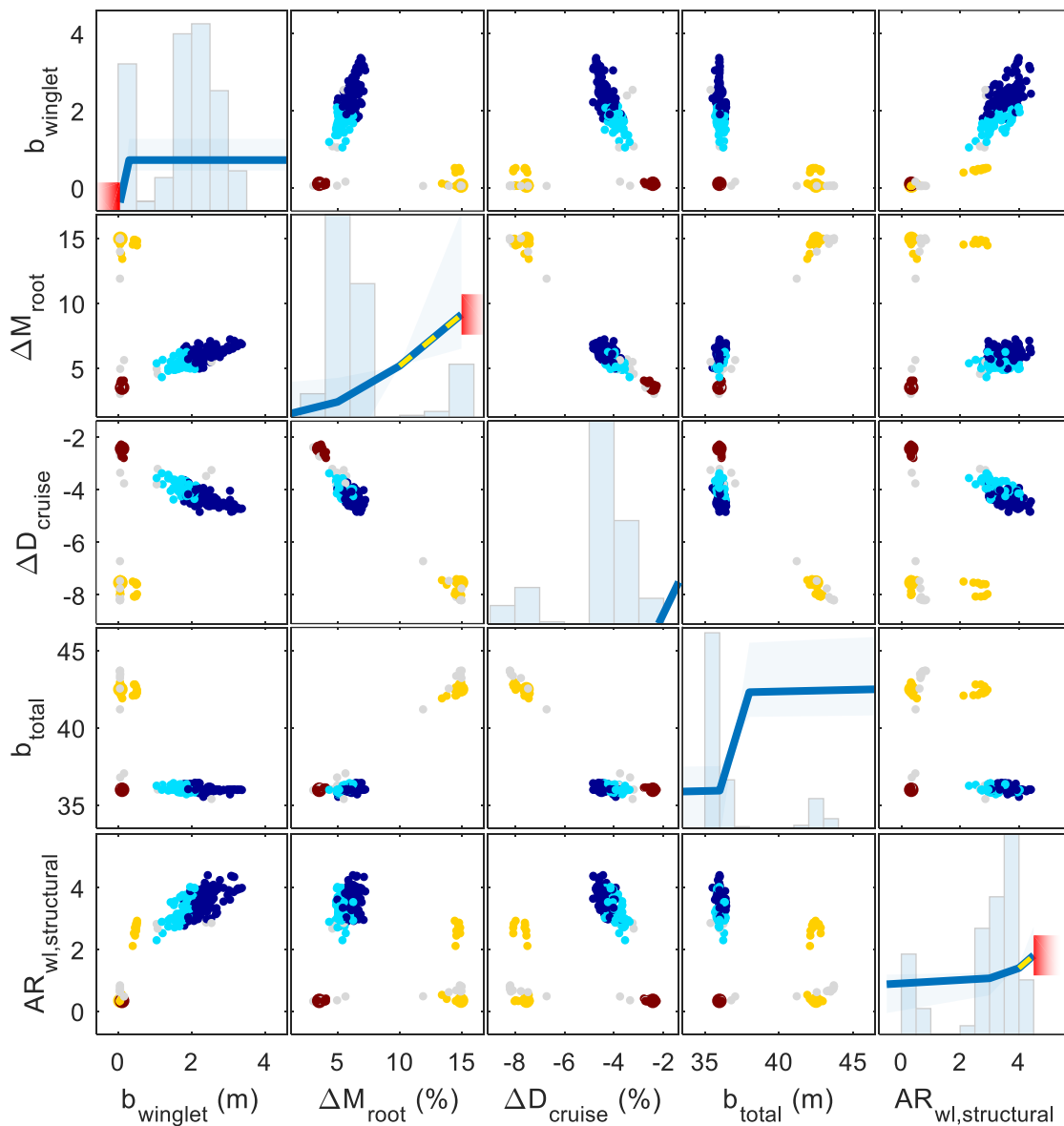


Figure 5-12. Wing redesign results visualization with clustering.

The wing geometries for the cluster-representative designs (indicated with larger markers in Figure 5-12) are shown in Figure 5-13, with data for each representative design in Table 5-5. In this case, the WISDOM search has yielded a variety of interesting potential alternatives. The design in Figure 5-13(a) is representative of design options that, once having violated the ICAO Code C span limit, are very high span, giving a significant drag reduction in exchange for very high root bending moments (flagged as possibly unmanageably high). Figure 5-13(c) shows a design representative of a very simple redesign of the wing tip that does not add a winglet, but rather is simply a small extension up to 36-meter limit for moderate drag reduction. In this case, the tuning of the clustering algorithm resulted in two representative designs that both feature winglets. Both utilize the full 36 m span, but the design in Figure 5-13(b) features a much smaller winglet (and slightly less drag reduction) than the design in Figure 5-13(d).

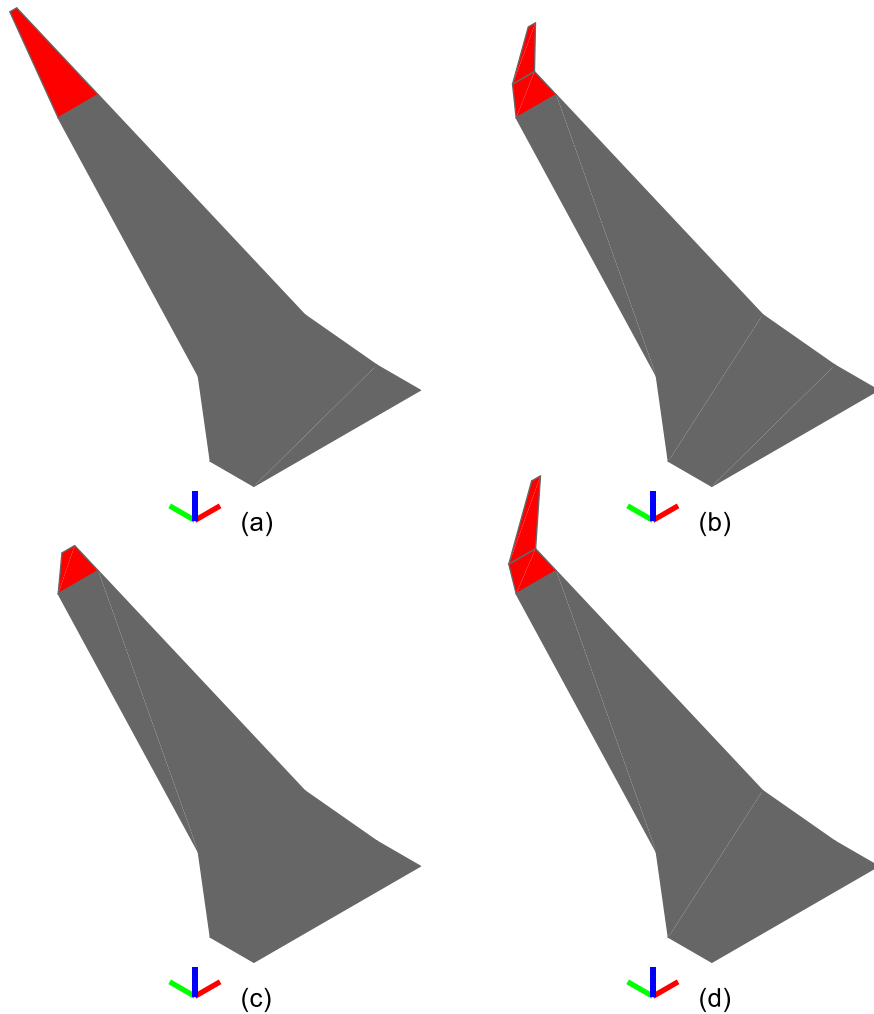


Figure 5-13. Representative alternative wing geometries. *Triads are one meter for scale.*

Table 5-5. Data for representative alternative designs.

Design	ΔW_{empty}	ΔM_{root}	ΔD_{cruise}	Span
a	1.04%	15.0%	-7.5%	42.5 m
b	0.53%	5.3%	-4.0%	36.0 m
c	0.23%	3.5%	-2.4%	36.0 m
d	0.76%	6.1%	-4.5%	36.0 m

These kinds of results represent the end of one iteration of the WISDOM searching process, to be used to inform the next iteration. For example, with the given results, the benefits of a high-span design such as in Figure 5-13(a) are quantified, potentially justifying exploration into more detailed structural analysis to better understand cost and feasibility. Or, it may prompt exploration into clever ways to circumvent span limits such as folding wingtips. The marginal benefits of the design of Figure 5-13(c) may be weighed against doing nothing at all. The alternatives that feature a winglet may be further refined with more design freedoms added to the setup to optimize airfoils, planform shapes, and riggings.

6 Conclusion

By implementing a new approach to achieve the objectives of capturing stakeholder knowledge and automating the integration of that information into the design iteration process, several benefits were observed. In this chapter, some of those benefits of the method and implementation developed for this work are discussed, along with some of the weaknesses. The motivations and objectives are related to the key results based on experience with the technique and tools so far. Experience has also revealed some pain points that will lead to likely future work to address them and/or increase capability.

6.1 Expected benefits

Many of the expected benefits of the WISDOM approach, the primary subject of this work, can be discussed in the context of the early design example cases. These benefits have been observed anecdotally in the test cases described in this document as well as additional cases, for example a personal homebuilt airplane case used for software development and testing. Experience utilizing the approach in the various cases has even indicated additional benefits not originally intended with the project.

6.1.1 Allocation of resources

Part of the motivation for undertaking this body of work was the potential to increase the efficiency of design processes in early design. One of the key benefits of the approach is the relatively low amount of effort needed to build preference maps compared to alternate activities where resources might be placed. Because the approach simply captures information that is already available, it provides an alternative path to putting significant effort, with associated diminishing returns, into market research to lock in requirements, building more in-depth and elaborate system models, or getting better technical assumption estimates.

The wing redesign example case system model described in Section 5.2.1 was written and tested in a matter of hours, benefitting from simple models and existing tools for integration with AVL. The resulting wing tip geometries in the use case presented here are quite reasonable and realistic without requiring any major effort in order to have informed analysis for structures or weights that normally constrain such a design. The preference maps used as surrogates were created using exclusively *a priori* knowledge and therefore did not require any research to develop, and the only effort was recording those surrogate preferences in the setup script. There was no need to edit or tune preference maps after initial searches to get reasonable results, either, as the uncertainty factors thoroughly capture the ‘roughness’ with which preferences are recorded at early stages of design.

6.1.2 Facilitating effective design iteration

It was found that the approach sometimes forces stakeholders to think more critically than they normally might. If there is foreknowledge that the approach will be used, for example, then this will influence how the system models are built, which assumptions are brought to the surface versus buried in the analysis, and which methods and analyses to choose. The activity of building preference maps is also a forcing function for stakeholders to introspect and sometimes gain insight about their own preferences and about the engineering and technical issues associated with a problem.

On the other end of the WISDOM process, the important next step after searching is for the designer and other stakeholders to use the results, along with the available artifacts that help facilitate conversation, to make human-in-the-loop design decisions to narrow down the design space and allow proceeding with the next iteration. The technique enables this by generating qualitatively distinct alternatives and presenting primarily those that are likely to

be of interest for further exploration or that reveal new insights about the design problem. In addition, through tools such as clustering and the scatter matrix visualization with interactive data brushing, the framework enables exploration and understanding of the quantitative tradeoffs between design alternatives and the driving issues. These sets of information yield the insights needed for accelerated design cycles, quick triaging of resources, and informed design decision-making. Also, because part of the information acquired in an iteration is the effects due to uncertain assumptions, the phrase “depending on the assumptions” becomes less frequent and design reviews and conversations become more productive.

Another way that the approach helps facilitate more effective iteration and decision-making is by presenting equally ‘fair’ alternatives in the results. In other words, a wide search through the design, requirements, and assumptions space still presents designs that, given their conditions, are optimized with the same fidelity as the others. Importantly, compared to more ‘hunt and peck’ approaches to design iterations, no time is wasted examining Pareto-dominated designs or designs that are not closed (possibly due to infeasibility).

6.1.3 Early optimization

One realized benefit that was not an explicit objective of the work was an ability to use optimization much earlier in a design process than would normally be appropriate. Optimization is a powerful tool, and one of its key benefits is searching in higher dimensions than humans are capable of similarly comprehending. The nature of the WISDOM approach is that requirements, design parameters, and assumptions are intentionally left unfixed for as long as possible, so not only is optimization essential, by allowing for a ‘messier’ problem setup than traditional optimization approaches, optimization can be used earlier to get closer to optimal designs.

Normally, optimization is a “garbage in, garbage out” process that requires the modeling of the system and the objectives to faithfully capture the physics of reality. With early design studies, the system model and/or requirements are normally too immature to be appropriate for use with optimization. With the technique presented in this work, however, the process can proceed under significant but explicitly captured uncertainty in modeling and requirements. As an example, when the winglet study case was run as a weighted sum multi-objective optimization, the resulting design point was a high span and very high bending moment design similar to that in Figure 5-13(a). With the WISDOM approach, “garbage in” is now allowed if, instead of putting significant effort into refining the input to optimization, a small effort is put into capturing the information on hand.

One of the other obstacles to using optimization earlier in design processes is simply the effort required to set up optimization and the inflexibility of that setup in keeping up with a quickly evolving design. With the WISE framework, setting up an optimization problem, especially one that takes care of the key speed-related pitfalls that can otherwise take some effort to avoid, is simple and straightforward to execute without mistakes in implementation. So, the

framework is quite a useful tool for a traditional optimization problem, even without taking the time to capture granular preferences and other knowledge.

6.2 Pitfalls and drawbacks

The primary objectives of this work were to formally capture additional information on hand and to put that information to good use in early design, and it should be acknowledged that there are potential alternative approaches to accomplishing either of these objectives. In the course of developing the specific approach that is the subject of this work, some alternatives were examined or trialed and not selected for various reasons. While the approach presented here has shown to yield many benefits, there are also some drawbacks, weaknesses, and pitfalls to highlight.

The learning curve can be one issue with both the WISDOM approach to design space exploration and the WISE framework and workflow. Because it presents a slightly different way of thinking about a design problem, it takes some practice to get used to thinking critically and introspecting about each requirement, how the analyses are formulated, and so on. What can also take some getting used to is the extent to which the designer and other stakeholders must acknowledge and accept a lack of certainty and a recognized level of incorrectness that is being carried through the design iteration process. Without this, there is a danger of falling into the trap of spending too much time trying to get a 'correct' optimization setup when it usually is much more efficient to simply use uncertainty factors to avoid this type of paralysis.

Various forms of over-capturing information can also have detrimental effects on the usefulness of the approach. One instance is the danger of double or triple bookkeeping of preferences. Notice for example that in the preferences in the wing redesign example of Section 5.2.2, there is no preference map attached to the magnitude of the span extension of the wing tips because the preference is fully captured in the preference map for the overall aircraft span. Fortunately, the method is somewhat robust to small amounts of double bookkeeping, but it primarily relies on the diligence of the designer to recognize and avoid the issue.

The root of other adverse effects from over-capturing information can often be traced to the nature of the underlying optimization algorithms employed and how their behavior may worsen the further away the problem is from being smooth and convex. In designing a personal airplane for one's own personal use, for example, a preference map on aircraft range may have two or three distinct steps to reflect the distances of specific frequent destinations from a known home airport. There may also be specific discontinuities and steps in the preference map for payload capacity based on precisely known weights of one's self and spouse, for example, such as in Figure 6-1. However, taking this same idea of capturing such granular preferences to the extreme would be, for an airliner example, to have a step change in value every time a new city pair in an airline's network can be serviced or have a step change in value whenever a single additional passenger seat can be added. Instead of yielding richer, more useful, and interesting results, this can lead to useless results due to effectively adding more noise than information to the design problem presented to the optimization algorithm.

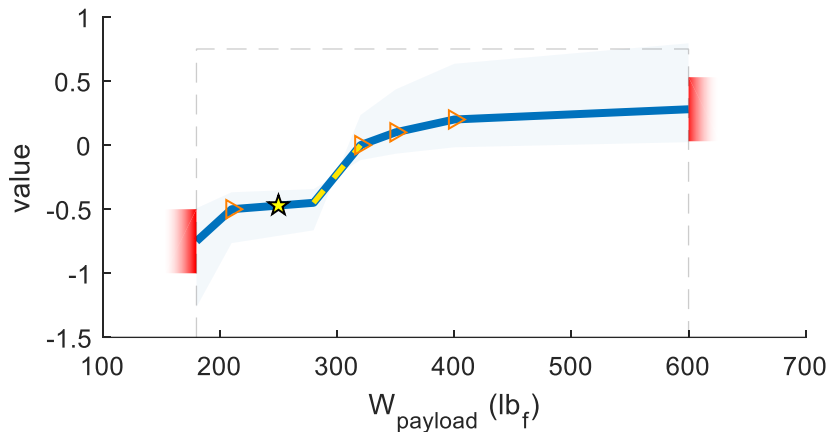


Figure 6-1. Granular preference map for payload capacity of a personal-use homebuilt aircraft.

The handling of discrete variables in the framework requires compromises to make them compatible with search algorithms that are not normally designed to accept them. Therefore, over-capturing information in the form of discrete variables can also lead to less useful results. The resulting values of discrete input variables is driven entirely by their associated starting probability density functions, so these parameters are determined in more of an ‘open loop’ fashion than from an informed search. Discrete output variables can be very useful for presenting information to human stakeholders, but because they present such a sharp step in the objective function that the optimization algorithm must work with, discrete output variables with attached preferences are often detrimental to the convergence performance of the algorithms.

Finally, one of the biggest downsides to the approach is the time required to execute a search. One of the features of the approach is that it allows retaining more design freedom later into the design process, but this also means that the optimizations being solved tend to have some more dimensions than the equivalent traditional design optimization setup might feature. This means that each search may be marginally slower, but the major source of long search times is the requirement to conduct dozens to hundreds of these sub-searches for each WISDOM iteration so that a significant sample of starting points, preferences, and assumptions can be captured.

6.3 Future potential

Initial experience using the WISDOM technique and WISE framework for development test cases, example cases, and other projects in industry has shown enough promise to warrant further development and exploration. The architecture of the framework makes it straightforward to implement changes. Continuing to use the approach and framework for real-world projects will drive efforts for further improvements and new capabilities.

6.3.1 Improvements

Recall from Section 4.1.1 that the requirements and drivers for creating the WISE framework center around utility, ease and flexibility of use, and speed. Most anticipated improvements will focus on increasing applicability to later stages of design, where it is anticipated that the usefulness of the technique will diminish as uncertainty is significantly reduced, requirements are solidified, and system models more closely capture true physics. So, though there may be less of a place for the WISDOM approach in later stages, the WISE framework can work well for carrying on iterating and optimizing the design.

6.3.1.1 Multi-level system model structure

The decision to use a structure data type as the representation of the system led to a workflow and design space that is intuitive to work with, with no need to use or track any positions of design variables in a vector, for example. However, this structure is flat, i.e., the framework is only able to work with the top-level fields of the main input-output system structure, precluding the use of nested structures, which can be a more intuitive way to manage system information as the complexity increases. The background process of conversion to and from the system structure happens every time the system model is run, so the decision to deprecate a `category` design variable attribute and impose a flat structure restriction was to reduce the overhead and increase the speed of design space exploration. This yields measurable speed gains for the simple system model functions used for development. However, it is also one of the few hurdles that can cause friction in continuing to use the framework beyond a very early design. In a more mature project, more parameters of the system are tracked, and a flat structure is a less appropriate format for describing the system.

One solution is to build a wrapper inside the system model function. This keeps the framework simple but in addition to creating a burdensome step of reparametrizing and rebuilding the system model and introducing some bookkeeping tasks, it also reduces the utility of the system structure itself, which has high visibility and usefulness in being viewed, passed to additional analyses, etc. Instead, the speed cost of the additional overhead, which is minor compared to system model evaluation with more realistic system models, can be absorbed to un-deprecate the capability to allow for multi-level system structures.

6.3.1.2 Speed improvements

In addition to the system parameterization becoming more complicated and involved as design progresses, so too do system models and analyses become more elaborate and higher fidelity. This usually comes with a significant increase in computation time needed to evaluate just a single design point. As the project progresses out of early design studies, it also becomes much less likely that the system model function can be easily built to be vectorized such that many design points can be evaluated in a similar amount of time as it takes to evaluate just one.

To address the speed issues, there are two likely strategies to pursue for improvement. The first is to improve coverage of the WISDOM search space by implementing alternative sampling techniques. Currently, only simple random sampling is used to generate starting points, scale preference maps, or choose assumption type parameters. Alternative sampling techniques, for example Latin hypercube sampling, have the potential to provide similar coverage of the search space using significantly fewer total searches of the design space.

The other speed improvement strategy is to enable parallelization. The WISDOM approach is ideally suited for parallelization, as after the randomized parameters are established for a given search, every search is fully independent of each other (assuming a thread-safe system model function). With multi-core processors now common in nearly all devices and especially in workstations, this is low-hanging fruit for a speed improvement, which, especially when combined with alternative sampling techniques, has the potential to yield an order of magnitude speed improvement for a WISDOM search. Unfortunately, both speed improvement strategies highlight one of the downsides of the choice of MATLAB as an environment for developing the framework, as both strategies, especially parallel processing, require additional toolboxes, reducing the accessibility of the framework.

6.3.2 New capabilities

In addition to improvements to the existing capabilities and workflow, there are several possibilities to expand on the approach and/or leverage the approach and the framework to build in new capabilities. In general, new capabilities are built when there is driving need coming from the requirements of a given project.

6.3.2.1 Correlation between technical assumptions

Currently, the simple random sampling used for uncertain assumption type parameters assumes that all these parameters are fully independent from each other. However, in many cases there is a known correlation, at least qualitatively, between parameters. This is something that the house of quality technique, discussed in Section 2.2.2, captures in its design feature correlation matrix. It would be beneficial in some cases to capture this additional available *a priori* information.

This could be useful, for example, in the case of the regional airliner example. Instead of penalizing designs based on aerodynamic efficiency using a preference map (Figure 3-12), an alternative formulation could instead use an

additional structural efficiency assumption type parameter that would act as a correction factor for the empty weight fraction function in Equation (11). The negative correlation between structural efficiency and aerodynamic efficiency could then be captured in the sampling used to stochastically generate these assumption parameters for search. Capturing this correlation may not only be a more realistic way to capture the information, but, crucially, it can sometimes be a more natural way to conceptualize the information for the designer and other stakeholders to capture. Sampling with correlation and arbitrary PDFs is an established and off-the-shelf capability, for example the correlated Latin hypercube sampling provided by Iman (2017).

6.3.2.2 Generation of additional information and artifacts

Setting up a design space in the WISE framework means that significant information is captured and available for use in generating other useful information and artifacts with uses not explicitly supporting optimization. Parameter sensitivities are one of the pieces of information that can be useful and acquired easily from the framework. A searching results set contains several perturbations of the design space, and this information can be processed to automatically estimate sensitivities and create a report or visual depiction.

Other types of visual artifacts can be useful for inclusion in reports, proposals, design reviews, etc. There is potential to leverage the framework to quickly and easily generate attractive constraint diagrams or carpet plots, for example, using the information in design variables to avoid the often significant burden associated with formatting and labeling plots for clarity.

6.3.2.3 Automated surrogate modeling

Because the technique already involves significant exploration of various parts of the design space, the framework lends itself well to the potential of being a platform for building surrogate models and hybrid analyses for computationally expensive system model functions. Initial rounds of searching can yield the data necessary to generate response surfaces that can be used in lieu of the expensive system model for much faster future iterations. A hybrid approach is also possible, whereby the more expensive higher-fidelity analyses are only used to occasionally calibrate correction factors on a much simpler and faster system model function.

6.3.2.4 More suited optimization algorithms

One of the largest risks when beginning to explore this work was that existing optimization algorithms would not be suitable for the WISDOM approach. Fortunately, this risk was not realized, and suitable performance was achieved using existing off-the-shelf optimization algorithms. However, these algorithms are not formulated with this unique type of search problem in mind, so there is potential for improvements in speed or usefulness of searching results if alternative optimization algorithms are explored or if a customized algorithm is created that is specifically suited and tuned for this type of application.

6.3.2.5 Configuration synthesis applications

The WISE framework creates a kind of language that captures a design space. There is therefore a potential to use the framework with configuration synthesis, such as the shape grammars used for aircraft configuration synthesis presented by Oberhauser et al. (2015). In this way, artificially generated unique configurations can be coupled with their respective design variables and a design space for that specific configuration. This setup would automatically create a runnable WISE optimization setup alongside synthesizing the configuration itself, allowing for fair comparisons of optimized versions of entirely different species of design solutions.

References

- Adler, W. (2018). *cornerplot*. GitHub. <https://github.com/wtadler/cornerplot>
- Atanasov, G. (2011). *Preliminary Design of a Two Seat General Aviation Electric Aircraft: EGL-K*. Unpublished semester thesis LS-SA 11/11. Institute of Aircraft Design, Technische Universität München, Garching, Germany.
- Brandt, S. A., Bertin, J. J., Stiles, R. J., & Whitford, R. (2004). *Introduction to Aeronautics: A Design Perspective* (2nd ed.). American Institute of Aeronautics and Astronautics. <https://doi.org/10.2514/4.862007>
- Chen, W., Sahai, A., Messac, A., & Sundararaj, G. J. (2000). Exploration of the Effectiveness of Physical Programming in Robust Design. *ASME Journal of Mechanical Design*, 122(2), 155. <https://doi.org/10.1115/1.533565>
- D'Errico, J. (2012). *fminsearchbnd*, *fminsearchcon* (1.4). MATLAB Central File Exchange. <https://www.mathworks.com/matlabcentral/fileexchange/8277>
- Defense Acquisition University Press. (2001). *System Engineering Fundamentals*. <http://www.dtic.mil/docs/citations/ADA606327>
- Drela, M., & Youngren, H. (2017). *Athena Vortex Lattice (AVL)* (3.36). <https://web.mit.edu/drela/Public/web/avl/>
- ESA. (n.d.). *Technology Readiness Levels (TRL)*. Retrieved January 19, 2019, from https://www.esa.int/Our_Activities/Space_Engineering_Technology/Shaping_the_Future/Technology_Readiness_Levels_TRL
- Felix, A. (2004). Standard Approach to Trade Studies : A Process Improvement Model that Enables Systems Engineers to Provide Information to the Project Manager by Going Beyond the Summary Matrix. *International Council on Systems Engineering (INCOSE) Mid-Atlantic Regional Conference*.
- Glas, M., & Sartorius, S. (2012, March). Towards a Continuous Build-up Process of a Reusable Requirements-based System Model. *2012 IEEE Aerospace Conference*. <https://doi.org/10.1109/AERO.2012.6187338>
- Haugen, E. B. (1980). *Probabilistic Mechanical Design*. Wiley.
- Hauser, J. R., & Clausing, D. (1988). The House of Quality. *Harvard Business Review*. <https://hbr.org/1988/05/the-house-of-quality>
- Herbst, S., & Hornung, M. (2015, June 22). ADDAM: An Object Oriented Data Model for an Aircraft Design Environment in MATLAB. *AIAA Modeling and Simulation Technologies Conference*. <https://doi.org/10.2514/6.2015-3243>
- Hoffpauir, D. (2017). NASA Systems Engineering Handbook. In *NASA/SP-2016-6105* REV 2. <https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20170001761.pdf>

- IBM. (2016). *IBM - Rational DOORS*. Rational DOORS. <https://www.ibm.com/us-en/marketplace/rational-doors>
- Ilgın, M. A. (2019). Aircraft Selection Using Linear Physical Programming. *Journal of Aeronautics and Space Technologies*, 12(2), 121–128.
- Ilgın, M. A., & Gupta, S. M. (2012). Physical Programming: A Review of the State of the Art. *Studies in Informatics and Control*, 21(4), 359–366. <https://doi.org/10.24846/v21i4y201201>
- Iman. (2017). *lhsgeneral* (1.2). MATLAB Central File Exchange. <https://www.mathworks.com/matlabcentral/fileexchange/56384>
- ISO/TC 20/SC 14. (2013). Space systems – Definition of the Technology Readiness Levels (TRLs) and their criteria of assessment. In *ISO/FDIS 16290:2013*.
- Jackson, P., Munson, K., & Peacock, L. (Eds.). (2004). *Jane's All the World's Aircraft, 2004-2005*. Jane's Information Group.
- King, R. (1987). Listening to the Voice of the Customer: Using the Quality Function Deployment System. *National Productivity Review*, 6(3), 277–281. <https://doi.org/10.1002/npr.4040060312>
- Lempia, D. L., & Miller, S. P. (2009). Requirements Engineering Management Handbook. In *DOT/FAA/AR-08/32*. https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/AR-08-32.pdf
- Loehr, F. (2013). *Survey of Multiobjective Optimization*. Unpublished semester thesis LS-SA 12/26. Institute of Aircraft Design, Technische Universität München, Garching, Germany.
- London, F. L. (2013). Analytic Method for Probabilistic Cost and Schedule Risk Analysis. In *National Aeronautics and Space Administration (NASA) Office of Program Analysis and Evaluation (PA&E) Cost Analysis Division (CAD) Contract Number NNH10PR24Z Final Report*. [https://www.nasa.gov/pdf/741989main_Analytic Method for Risk Analysis - Final Report.pdf](https://www.nasa.gov/pdf/741989main_Analytic%20Method%20for%20Risk%20Analysis%20-%20Final%20Report.pdf)
- MATLAB (Version 9.5)* (9.5 (R2018b)). (2018). The MathWorks, Inc. <https://www.mathworks.com/products/matlab.html>
- McQuarrie, E. F. (2016). *The Market Research Toolbox: A Concise Guide for Beginners* (4th ed.). SAGE Publications, Inc.
- Messac, A. (1996). Physical Programming: Effective Optimization for Computational Design. *AIAA Journal*, 34(1), 149–158. <https://doi.org/10.2514/3.13035>
- Messac, A. (2015). Optimization in Practice with MATLAB®: For Engineering Students and Professionals. In *Cambridge University Press*.
- Messac, A., & Hattis, P. D. (1996). Physical Programming Design Optimization

- for High Speed Civil Transport. *Journal of Aircraft*, 33(2), 446–449. <https://doi.org/10.2514/3.46961>
- Messac, A., & Mattson, C. A. (2002). Generating Well-Distributed Sets of Pareto Points for Engineering Design using Physical Programming. *Optimization and Engineering*, 3(4), 431–450. <https://doi.org/10.1023/A:1021179727569>
- Nagel, B., Böhnke, D., Gollnick, V., Schmollgruber, P., Rizzi, A., La Rocca, G., & Alonso, J. J. (2012). Communication in Aircraft Design: Can We Establish a Common Language? *28th International Congress of the Aeronautical Sciences*. http://www.icas.org/ICAS_ARCHIVE/ICAS2012/ABSTRACTS/201.HTM
- NASA. (n.d.). *Definition Of Technology Readiness Levels*. Retrieved December 26, 2018, from https://esto.nasa.gov/files/trl_definitions.pdf
- Nicolai, L. M., & Carichner, G. E. (2010). *Fundamentals of Aircraft and Airship Design*. American Institute of Aeronautics and Astronautics. <https://doi.org/10.2514/4.867538>
- Oberhauser, M. (2013). *Carpet Plots in Parametric Trade Studies: Development of a Matlab Tool to Create Carpet Plots*. Unpublished semester thesis LS-SA 12/24. Institute of Aircraft Design, Technische Universität München, Garching, Germany.
- Oberhauser, M., Sartorius, S., Gmeiner, T., & Shea, K. (2015). Computational Design Synthesis of Aircraft Configurations with Shape Grammars. *Design Computing and Cognition '14*, 54(2), 21–39. https://doi.org/10.1007/978-3-319-14956-1_2
- Optimization Toolbox (Version 8.2) (8.2 (R2018b))*. (2018). The MathWorks, Inc. <https://www.mathworks.com/products/optimization.html>
- Parkinson, A. R., Balling, R., & Hedengren, J. D. (2018). *Optimization Methods for Engineering Design* (2nd ed.). Brigham Young University. <http://apmonitor.com/me575/index.php/Main/BookChapters>
- Rao, S. S. (2009). *Engineering Optimization: Theory and Practice* (4th ed.). Wiley.
- Raymer, D. (2002). *Enhancing Aircraft Conceptual Design Using Multidisciplinary Optimization* (Issue May). (PhD Thesis). Royal Institute of Technology.
- RTCA SC-205. (2011). *Software Considerations in Airborne Systems and Equipment Certification (DO-178C)*.
- SAE S-18. (1996). *Aerospace Recommended Practice 4761, Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*. <https://www.sae.org/standards/content/arp4761>
- SAE S-18. (2010). *Aerospace Recommended Practice 4754 Rev. A, Guidelines for Development of Civil Aircraft and Systems*. <https://www.sae.org/standards/content/arp4754a/>

-
- Sartorius, S. (2011a). A Tool for Rotorcraft Pre-Design Sizing. *67th Annual Forum of the American Helicopter Society*.
- Sartorius, S. (2011b). *CORE: Conceptual Optimization of Rotorcraft Environment* (0.7). MATLAB Central File Exchange. <https://www.mathworks.com/matlabcentral/fileexchange/41725>
- Sartorius, S. (2015). *Bisection Method Root Finding* (1.14). MATLAB Central File Exchange. <https://www.mathworks.com/matlabcentral/fileexchange/28150>
- Sartorius, S. (2016). *BLEND: utility for smoothly blending functions or creating piecewise functions*. MATLAB Central File Exchange. <https://www.mathworks.com/matlabcentral/fileexchange/56530>
- Sartorius, S. (2019a). *Physical Units Toolbox* (4.1). GitHub. <https://github.com/sky-s/physical-units-for-matlab>
- Sartorius, S. (2019b). *Pseudorandom number generation for engineering estimates*. GitHub. <https://www.github.com/sky-s/randx>
- Sartorius, S. (2011c). Optimization and Design Space Visualization Applied to Early Conceptual Design of Compound Helicopters. *52nd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, 1–10. <https://doi.org/10.2514/6.2011-1832>
- Sartorius, S., & Hornung, M. (2018). A Technique for Capturing Stakeholder Preferences and Knowledge in Early Design Studies. *31st Congress of the International Council of the Aeronautical Sciences*. http://www.icas.org/ICAS_ARCHIVE/ICAS2018/data/preview/ICAS2018_0208.htm
- Schneiderbauer, D. (2013). *Airbus A380 Multi-Objective Redesign*. Unpublished semester thesis LS-SA 12/23. Institute of Aircraft Design, Technische Universität München, Garching, Germany.
- Sorokin, J. (2018). *HDBSCAN*. GitHub. <https://github.com/Jorsorokin/HDBSCAN>
- Sözüer, S., & Thiele, A. C. (2016). The State of Robust Optimization. In *International Series in Operations Research and Management Science* (Vol. 241, pp. 89–112). https://doi.org/10.1007/978-3-319-33121-8_5
- The beta-PERT Distribution*. (n.d.). Retrieved February 12, 2019, from <http://www.riskamp.com/beta-pert>
- Ullman, D. G., & Spiegel, B. P. (2006). Trade Studies with Uncertain Information. *Sixteenth Annual International Symposium of the International Council On Systems Engineering (INCOSE)*.
- Vanderplaats, G. N. (2007). *Multidiscipline Design Optimization*. Vanderplaats Research & Development, Inc.
- Wong, K.-C. (2015). *Evolutionary Multimodal Optimization: A Short Survey*.

<http://arxiv.org/abs/1508.00457>

- Wood, K. L., Antonsson, E. K., & Beck, J. L. (1989). Representing Imprecision in Engineering Design – Comparing Fuzzy and Probability Calculus. *Research in Engineering Design*, 1(3–4), 187–203. <https://doi.org/10.1007/BF01581211>
- Wood, K. L., Otto, K. N., & Antonsson, E. K. (1992). Engineering Design Calculations with Fuzzy Parameters. *Fuzzy Sets and Systems*, 52(1), 1–20. [https://doi.org/10.1016/0165-0114\(92\)90031-X](https://doi.org/10.1016/0165-0114(92)90031-X)
- Yatsuka, T., Ishigaki, A., Ijuin, H., Kinoshita, Y., Yamada, T., & Inoue, M. (2018). Mathematical Modeling of Multi-player Multi-objective Decision Making by Linear Physical Programming. *Proceedings - 2018 7th International Congress on Advanced Applied Informatics, IIAI-AAI 2018*. <https://doi.org/10.1109/IIAI-AAI.2018.00147>

Appendix A: WISE Documentation

A1 README

DUEL WISE

Copyright 2018 Sky Sartorius. All rights reserved. No license offered.

Contact: www.mathworks.com/matlabcentral/fileexchange/authors/101715

DUEL is the Design Understanding and Exploration Library. WISE, the Well-Informed Search Environment, is a DUEL tool that facilitates optimization and search of the design space.

Part of WISE core functionality is facilitating a technique for capturing and using additional knowledge, captured in the `DesignVariable` class, such as preferences and uncertainties, for automated design space exploration. The `DesignSpace` class is the container for a design problem. It has as key properties a handle to the function that physically models the system and the vector of `DesignVariables`. The `DesignSpace` class has most methods needed for working with the design.

Demos

Three demos demonstrate the workflow and features using a personal homebuilt airplane design test case, a winglet design test case, and a regional jet test case.

Dependencies

DUEL WISE requires several dependencies. Some dependencies are required for specialized functionality only, while others are required for even basic functionality.

`.dependencies` directory

Most dependencies are in the `.dependencies` directory. Many of the files that are p code here have source available on the [File Exchange](#).

Matlab version

DUEL-WISE is built primarily on and for R2018b but should work on later versions.

Installed toolboxes required

- Matlab Optimization Toolbox
- [Physical Units Toolbox](#)

A2 DesignVariable help block

A **DesignVariable** describes the variable as well as contains preferences for the variable. Also, how it interfaces with the system function.

In design, there is typically a LOT of information to be collected for each variable beyond simply a nominal baseline value. Most of this information is normally easily known a priori, so typically the only effort is the input process, without significant research and analysis effort needed.

DesignVariable properties:

name = The system model works with name as a field of the I/O struct.
description = Optional long description (user reference only).
label = Optional (TeX-interpreted) string to use for labeling of e.g. plot axes instead of using name.
units = String representing preferred working units for the variable. Evaluated by str2u when set (stored in hidden unitsDv property).
ioType = I/O expectation of system model: input, output, <, >, or =. Default ioType is "input".
[closing](#) - Variable is a closing variable (for ioType <, >, or =).
[discrete](#) - Set to true if variable is discrete.
parameter = Set to true for assumption-like input variables that have uncertainty but should hold constant for a given local search.

free = Value is allowed to vary in searching.
fixed = ~free (fixed pure outputs enforced with equality constraints).
preferenceOn = If set to false, preferences evaluate to zero (flat preference), but bounds and, if on, ratcheting bounds, are still active.

[ratcheting](#) - Toggle ratcheting behavior on/off for this variable.

[lowerMultiBounds](#) - Inferred lower bounds when ratcheting.

upperMultiBounds = See lowerMultiBounds.

[lowerBound](#) - Lower bound.

startLowerBound = Used for distribution of search starting points for search. Inputs only.

value = Nominal value. Index for discretized.

startUpperBound = Upper bound of distribution of search starting points.

upperBound = Upper bound.

lbvub = A shorthand 5-element vector for setting (or getting) [lowerBound startLowerBound value startUpperBound upperBound].

[distribution](#) - PDF of starting point for inputs.

[smallestMeaningfulStep](#) - Affects min step size for finite differences.

Preference map properties:

prefType = Specifies preference as "penalty" (default) or "value" (negative penalty).

[prefAbscissa](#) - Preference map abscissa values (1-by-n).

[prefOrdinate](#) - Preference map ordinate values (1-by-n).

prefMap = A shorthand to set [prefAbscissa', prefOrdinate']. May have unexpected behavior with discretized.

[prefSmoothing](#) - Width(s) of smoothing region between preference map segments.

[prefUncertainty](#) - Parameter to characterize uncertainty of prefOrdinate.

prefInfo = Array of info associated with each preference map node.

[prefFlags](#) - Flags on preference map segments that should be highlighted in results (n+1 elements).

prefSlope = Slopes of preference map (n-1 elements). Setting this changes only the ordinate value immediately to the right of the segment.

flatPreference = True if all ordinate values are the same. Setting this (to anything but false) erases ordinate data and sets all ordinate points to zero.

DesignVariable methods:

DesignVariable = Constructor where first argument is name and following arguments are property/value pairs.

`runchecks` = Checks if **DesignVariable**(s) make sense and cleans up.

[plot](#) - Plots preference map.

[editmap](#) - Simple interface for editing a preference map.

`flip` = Multiplies ordinate values by -1, switches prefType between "value" and "penalty", and adjusts origin.

[export](#) - Create mcode that captures the **DesignVariable**.

`DesignVariable` static methods are a convenient shorthand for capturing certain common variable types:

[assumption](#) - Assumption-like input parameter.

[error](#) - Output enforced to be equal to zero.

[margin](#) - Output enforced to be greater than zero.

See also [DesignSpace](#), [duel wise demos](#).

A3 Select DesignSpace help documentation

A3.1 DesignSpace help block

A **DesignSpace** object contains a vector of DesignVariables and has the methods in place to facilitate searching, exploration, understanding, and optimization of the design as analyzed by the systemModel function.

DesignSpace properties:

variables = Vector of DesignVariables.
[v](#) - Struct for quick name-based access to variables.
 systemModel = Function handle to the system model function. The system model function has the syntax S = systemModel(S), where S is the I/O struct.

[extraModelArguments](#) - Extra arguments for **DesignSpace**.postevaluation.
[extraSystemFunctions](#) - Extra functions for **DesignSpace**.postevaluation.

Run settings, mostly for **DesignSpace**.searching:

[useRunMemoization](#) - Memoizes systemModel function.
[usePreferencesMemoization](#) - Memoizes preference map interpolation.
[useConvertXMemoization](#) - Memoizes conversion from design vector to I/O struct.
[useCloseDesignMemoization](#) - Memoizes initial closing of design.
[ratchetBounds](#) - Uses lower- and upperMultiBounds for variables with ratcheting set to true. Default = false.

Useful read-only properties (mostly set by **DesignSpace**.processvariables):

prefFuncs = Function handles that yield preference penalty values.
 inputVariables = Vector of DesignVariables that are inputs.
 free = Free input variables (logical array).
 active = Struct with various useful indices of active inputs.
 LB = Input variables lower bounds.
 UB = Input variables upper bounds.
 ind = Struct with various useful indices of variables.
 n = Struct with various useful numbers of variables.
 names = Struct with various useful names of variables.
 vars = Struct with various useful DesignVariable vectors.
 unitsDvs = Struct for storage of eveled units.
 abscissas = Struct with various useful abscissas.

Properties for handling results:

cachedResults = Table containing previous results.
[results](#) - Struct with results of searching.

DesignSpace methods:

DesignSpace = Constructor with property/value pairs.
[processvariables](#) - Processes variables and populates useful properties.
[run](#) - Runs the systemModel function.
[batchrun](#) - Run Monte Carlo or parameter sweep batch runs.
[closedesign](#) - Return a closed design.
[searching](#) - Explore the design space by optimizing from various start points.

Post-searching tools:

[appendresults](#) - Append searching results to other results obtained with similar settings. Use for expanding results set.
[cluster](#) - Use clustering algorithm to cluster results.
[scattermatrix](#) - Plots a scatter matrix visualization of design points.
 postroutine_cluster = A script that implements cluster and scattermatrix with some typical settings.
[buildresulttable](#) - Create / copy for export formatted table of results.
[dominantstruct](#) - Returns a struct of only dominant designs from specified results substruct.

Tools for evaluating individual designs:

[preferences](#) - Returns raw preferences for a given design.

[postevaluation](#) - Runs systemModel with extraModelArguments and extraSystemFunctions.

Iteration tools:

sendresultstocache = Saves a copy in cachedResults and clears results.
toggle = Presents a text menu-based prompt for changing the fixed/free property of variables.

Working with DesignVariables (mostly leveraging DesignVariable methods):

plot = Calls plot method on variables.
refreshplot = Refreshes plot data.
[editmap](#) - Calls editmap method for variable specified by name.
[add](#) - Appends specified DesignVariable to the variables vector.

See also [DesignVariable](#), [duel wise demos](#).

A3.2 DesignSpace.run help block

run Runs the system model function.

[S, Sin, X] = run(o, x, active, X)

o.run(x, free, X) populates the free elements of X with x, converts the result to an input struct for the system function, and runs the system function with that input. x is the design point vector for a shrunken problem that only deals with the free variables, while X is the full design vector that has enough information to populate a full input struct.

If X is not provided, x must be a full design vector with enough information to populate an input struct.

This function is memoized such that two runs in a row with the same point don't evaluate the system model function twice (doubles the speed, since systemModel is required both for fun and nonlcon functions, which solvers call separately).

See also [DesignSpace](#), [DesignSpace/scattermatrix](#).

A3.3 DesignSpace.batchrun help block

batchrun Runs the system model with select input struct fields as vectors, closing the design if needed and possible. An output struct is returned.

Monte Carlo

batchrun(o, n) Does a Monte Carlo run of n trials using randomized (based on their respective distributions), vectorized inputs for free parameter input variables.

batchrun(o, n, dim) Makes sure that vectors fed into the system model are non-singleton along the dimension specified by dim.

Additional name/value pairs are passed to bisection.

If n is -2, batchrun runs every combination of sLb and sUb for every parameter (2^nParameters cases total). If n is -3, batchrun runs every combination of sLb, val, and sUb (3^nParameters points). batchrun(o, -n) Runs every combination of linspace(sLb, sUb, n) for every parameter.

Parameter sweeps

batchrun(o, n, dim, names) Runs with vectors on named input variables (cellstr or string array) that are on the range of sLb to sUb for that particular named DesignVariable.

Pass extra name/value pairs sweepStarts and sweepEnds to get away from the default start and end of input sweep vectors. Value should be a cell array of the same number of elements as names, with empty elements to use the default. Points should carry dimensions.

Append 'bisectionArgs', ArgsCell to any call to pass extra arguments and options to the bisection function.

Example usage for sweep:

```
o.batchrun(100, 1, ["param1" "param2"], 'sweepStarts', {[[] 5*u.ft});
```

See also [bisection](#), [histcounts](#), [histfitpert](#).

A3.4 DesignSpace.searching help block

`o.searching` Search the design space by running minimizations with varying start points and uncertainty factor scaling.

If no output, the results are written to the DesignSpace results property.

Additional options to pass in name/value pairs:

<code>nStarts:</code>	Number of random starting points. Default <code>nStarts = 1</code> . If <code>nStarts = 1</code> , the start is at the nominal DesignVariable values, search is without uncertainty scaling, and <code>o.results</code> is not written and the output is simply returned instead.
<code>closing:</code>	"first" (default): Closes start point before searching. "on": Closes for every point evaluated during search. "off": No closing behavior.
<code>closingOptions:</code>	Cell list of additional arguments passed to <code>optimoptions</code> for closing.
<code>runMinimization:</code>	Run a search to minimize preferences. Default = true. Set to false if all you want is a closed point.
<code>algorithm:</code>	Type of minimization algorithm as used by <code>fmincon</code> . Default is <code>fmincon</code> default 'interior-point'. Additional option of 'simplex' uses <code>fmincon</code> to find a feasible starting value that is fed to <code>fminsearchcon</code> .
<code>backup:</code>	Saves each iteration of the sometimes very long loop, allowing, e.g., interrupting the loop but preserving the points run thus far. Set to "on" (default) to back up (will not back up with <code>nStart = 1</code>), "off" to not back up. Set to "recover" or "restore" to skip search and recover the last search from <code>.searching_backup.mat</code> . Set to a file name for load to restore from some other <code>.mat</code> file.
<code>showProgressBar:</code>	If set (default), shows progress bar for <code>nStarts > 1</code> .
<code>searchOptions:</code>	Cell list of additional arguments to pass to <code>optimoptions</code> (or <code>optimset</code> for simplex search).

See also [fmincon](#), [DesignSpace](#), [DesignSpace/appendresults](#).

A3.5 DesignSpace.cluster help block

`cluster` Clusters results using selected algorithm.

```
[clusterId, clusterCenter, clusterSize, (hdbObjb)] = cluster(o, varargin)
```

After the DesignSpace object, parameters are passed as name/value pairs:

<code>designPoints:</code>	Struct array of design points (S). If a string, <code>o.results.(designPoints)</code> data is used, including <code>o.results.(designPoints).S</code> . Default = 'valid'.
<code>paramInd:</code>	Logical indices of DesignVariables (<code>o.variables</code>) that are of interest for clustering. Default <code>paramInd = o.ind.input</code> (non-varying are automatically removed). Or, if a string array, the field names in the <code>designPoints</code> struct that are of interest.
<code>writeTo:</code>	If provided, writes <code>clusterId</code> and <code>clusterCenters</code> to <code>o.results.(writeTo)</code> . Default is <code>designPoints</code> if <code>designPoints</code> is a string.
<code>clusterSubresult:</code>	A new results substruct, <code>o.results.(clusterSubresult)</code> , containing only the cluster center design points. Default is 'interesting'.
<code>normalize:</code>	If set, will normalize data to all be from 0 to 1 prior to

clustering. Default normalize = true.
 algorithm: Algorithm to use for clustering. Default = 'neighbor chain'.
 Other options are 'HDBSCAN' or 'ffcmw'.

Additional parameters are passed to the clustering algorithm to adjust clustering settings.

For 'neighbor chain', example parameters are:

threshold: The percentile of the distances between all pairs below which two points are considered connected neighbors. Default = 0.25 (lower quartile).
 minNeighbors: Minimum number of neighbors for a point to not be considered an outlier. Default = 1.

For 'HDBSCAN', example parameters are:

minpts: The nearest 'minpts' neighbor used for core distance calculation for each point in X. Default = 5. I have the best luck using the minimum (2).
 minclustsize: The minimum # of points necessary for a cluster to be deemed valid. Default = 5.
 minClustNum: The minimum # of clusters to be realized. Default = 1.
 outlierThresh: A cutoff value between [0,1], where any X(i) with an outlier score (see below) greater than 'outlierThresh' is assigned as an outlier (ID = 0). Default = 0.9.

For 'ffcmw', parameters are:

nClusters: Number of clusters (providing this is MANDATORY).
 options: Options vector for ffcmw function.
 outlierThresh: If provided, points with maximum grade of membership less than outlierThresh are considered outliers.

See also [HDBSCAN](#), [neighborchainclustering](#), [ffcmw](#), [colormap](#), [scattermatrix](#).

A3.6 DesignSpace.scattermatrix help block

scattermatrix Plots a scatter matrix visualization of multiple design points.

scattermatrix(o, varargin), where o is a DesignSpace object.

Additional arguments in name/value pairs:

designPoints: Struct array of design points. If a string, o.results.(designPoints) data is used, including o.results.(designPoints).S. Default = 'valid'.
 paramInd: Logical indices of DesignVariables (o.variables) that are of interest for plotting. Default paramInd = o.ind.careAboutPrefs & o.ind.preferenceOn. Or, if a string array or cellstr, the field names in the designPoints struct that are of interest.
 plotType: 'scatter' (default) or 'corner'.
 plotPrefMaps: If true (default), preference maps are overlaid on the diagonal histograms.
 linking: If true, scatter matrix axis linking and data brushing will be used. Default linking = false (for speed).
 CData: CData passed directly to scatter plots unless CData is a string. 'rawprefs' will use o.results.rawPrefSum. 'flagged' uses ~cellfun('isempty', o.results.(designPoints).prefFlags. 'clusters' uses o.cluster with some defaults (get more control over the clustering by running o.cluster yourself with your own settings then using o.results.(writeTo).clusterId with zeros replaced by NaNs), with outliers as NaN. Other strings will use [o.results.S.(CData)]. Default CData = [] to skip any customization (all will be one color).
 GData: Indices of points whose color should be made quite gray. Default is o.results.(designPoints).dominated. Only works with n x 3 CData (or 'flagged').
 SizeData: If provided, will override scatter SizeData. If SizeData = 'flagged', flagged points will have their SizeData value halved.

variationTol: Parameters (indicated by paramInd) will not be used if there is no variation, as determined by variationTol. Default variationTol = 1e-5 (set to 0 to include everything).

```
[s,ax,bigAx,h,hAx,hP,vars] = scattermatrix(o)
```

See also [scatter](#), [histogram](#), [colormap](#), [DesignVariable/plot](#).

Appendix B: Example Case Source Code

B1 Regional airliner system model function

```

function S = system_model(S_in, varargin)
% This system model models sizing of a jet. It is intended to demonstrate three
% general categories of usually under-captured knowledge and preferences in
% early design.
%
% Copyright 2018 Sky Sartorius. All rights reserved.
% Contact: www.mathworks.com/matlabcentral/fileexchange/authors/101715

%% Inputs.
% We can define all of our inputs here so we can run this by itself, e.g. for
% development or debugging, but we make sure to override anything here with
% function inputs. Note that not everything has to be overwritten, so an inputs
% list at the top can be quite long.

S.range = 2500 * u.nmi;
S.passengers = 100;
S.maxLiftToDrag = 17;
S.sfc = 0.5 * 1/u.hr;
S.grossMass = 50*u.tonne;

if nargin
    nms = fieldnames(S_in);
    for i = 1:numel(nms)
        S.(nms{i}) = S_in.(nms{i});
    end
end

%% Parameters and assumptions.

% Average payload including cargo, furnishings, etc.
S.payloadMassPerPax = 95*u.kg;

% Average crew per 14 CFR § 121.391.
S.crewMass = 80*u.kg * (2 + 0.5 + S.passengers/50);

% Empty weight fraction trend parameters.
A = 0.902;
C = -0.0385;

%% Analysis and outputs.
S.emptyWeightFraction = A * (S.grossMass/u.lbm).^C;
S.fuelFraction = ifr_mission_profile(S);
S.payloadFraction = 1 - (S.fuelFraction + S.emptyWeightFraction);

S.passengerMass = S.passengers * S.payloadMassPerPax;
S.payloadMass = S.passengerMass + S.crewMass;
S.grossMass = S.payloadMass ./ S.payloadFraction;

S.utility = S.passengerMass .* S.range;

%% Populate additional useful parameters.
S.emptyMass = S.emptyWeightFraction .* S.grossMass;
S.fuelMass = S.grossMass .* S.fuelFraction;
S.usefulLoad = S.fuelMass + S.payloadMass;

S.techFactor = S.maxLiftToDrag./S.sfc/u.hr;

end

function fuelFraction = ifr_mission_profile(S)
% Fuel fraction for IFR flight profile with 200 nm diversion and several fixed
% assumptions.

cruiseSpeed = 450*u.kts;

```

```
startupTaxiFrac = 0.98;
takeoffClimbFrac = 0.985;
cruiseFrac = exp(-S.range.*S.sfc./(cruiseSpeed.*0.866*S.maxLiftToDrag));
diversionFrac = exp(-200*u.nmi.*S.sfc./(cruiseSpeed.*0.866*S.maxLiftToDrag));
% SFC improvement factor of 0.9 for lower Mach (per Raymer).
reserveFrac = exp(-45*u.min.*(S.sfc*0.9)./S.maxLiftToDrag);
descentLandingFrac = 0.99;

missionFrac = startupTaxiFrac.*takeoffClimbFrac.*cruiseFrac.*diversionFrac...
    .*reserveFrac.*descentLandingFrac;

% Find fuel fraction with 6% margin added.
fuelFraction = 1.06*(1 - missionFrac);
end
% NBAA IFR profile reference:
% awin.aviationweek.com/portals/awin/pdfs/bc\_05\_01\_2013\_p33a-33f\_howtochart.pdf
```

B2 Regional airliner setup script

```

%% Set up workspace.
cch all
clear global % Reset memoization.
rng(0); % Make repeatable.
displayUnits = {'lb', 'nmi', 'lbf'};

%% Set up design space.
o = DesignSpace('systemModel', @duel_wise_demos.regional_jet.system_model);

%% Define design variables.
% Define "range" design variable.
v = DesignVariable('range', 'units', 'nmi');
v.prefType = 'value';
v.prefUncertainty = 2;
v.prefMap = {
... absc.   ord.   info
    400     0     ''
    1000   0.7    'still a useful part of fleets'
    1500    1     'can do vast majority of routes'
    3000   1.3    'can do some really long routes'
};
v.lowerBound = 400;
v.startLowerBound = 500;
v.value = 2500;
v.startUpperBound = 3500;
v.upperBound = 4000;
% v.lbvub = [400 500 2500 3500 4000]; % Alternate shorthand of 5 lines above.

o.v.range = v; % Assign design variable to design space.

% Define "passengers" design variable.
v = DesignVariable('passengers');
...
v.prefUncertainty = 2;
v.prefAbscissa = [60 75 95 100 105 135 150];
v.prefOrdinate = [1 0.5 0.5 0.2 0.5 0.5 0.7];
v.prefSlope = 0.001; % Slightly favor smaller aircraft
v.prefInfo{1} = "encroaching on firm's existing portfolio";
v.prefInfo{4} = 'original requirement specification';
v.prefInfo{7} = 'competing with established narrow-bodies';
v.lbvub = [50 70 100 140 180];

% The field name used for assignment to the utility dependent property v does
% not matter.
o.v.pax = v;

% grossMass
v = DesignVariable("grossMass", 'units', 'tonne', 'label', 'm_0');
v.ioType = "=";
v.prefUncertainty = 4;
v.closing = 1;
v.prefAbscissa = [0 60]; % This is my "currency".
v.prefOrdinate = [0 .8];
v.lbvub = [1 40 40 40 inf];
v.units = 'lb'; % Do a unit conversion (may throw a warning).

o.v.w0 = v;

% utility
v = DesignVariable("utility", 'units', 'lb-nmi');
v.ioType = "output";
v.prefUncertainty = 4;
v.prefAbscissa = [0 100*200*2500];
v.prefOrdinate = [.4 0];
v.flip;

o.v.utility = v;

% sfc
v = DesignVariable();

```

```
v.name = "sfc";
% v.label = "Specific fuel consumption";
v.units = 'lb/hr/lb';
v.parameter = true;
v.distribution = 8;
v.lbvub(2:4) = [0.45 0.50 0.65];
% Raymer for high BPR turbofan: 0.5/hr for cruise, 0.4/hr for loiter.

o.v.sfc = v;

% maxLiftToDrag
v = DesignVariable("maxLiftToDrag", 'label', 'L/D_{max}');
v.parameter = true;
v.prefUncertainty = 3;
v.startLowerBound = 13;
v.startUpperBound = 22;
v.distribution = 3;
v.value = 17;

v.prefAbscissa = [10 17 21 25];
v.prefOrdinate = [0 0.1 0.2 .3];

v.prefFlags(4) = "Suspiciously high L/D";

o.v.maxLiftToDrag = v;
```

B3 Wing redesign with winglet system model function

```
function S = sys_winglet(S_in, varargin)

%% Inputs.

S.allowWeightIncrease = 1;
S.tipSpan = 0.5*u.m; % 2.15/2 = 1.075 for -700 winglet.

S.tipChordShrinkRatio = 0.6; % 0.73 for -700 winglet.
% 1 means continue straight LE and TE. 0.73 for baseline winglet. 0.6 for
% baseline wing.

S.tipTeKink = 0*u.deg; % 0 means straight TE.
S.tipTwist = 0*u.deg;
S.tipExtraDihedral = 0*u.deg;

S.wingletSpan = 2.315*u.m;
S.wingletRootInc = 0*u.deg;
S.wingletTwist = 0*u.deg;
S.wingletAngle = 86*u.deg;
S.wingletExtraLeSweep = 0*u.deg; % Min 0.
S.wingletTaperRatio = 0.33;

S.wingTwist = 0*u.deg;

S.surfaceMassPerArea = 10*u.lb/u.sqft;
% Checked on Roskam. Checks out on
% aircraftengineering.wordpress.com/category/boeing/boeing-737/page/2/

%% Replace inputs above with those provided in S_in.
if nargin
    nms = fieldnames(S_in);
    for i = 1:numel(nms)
        S.(nms{i}) = S_in.(nms{i});
    end
end

%% Fixed parameters

m0 = 70*u.t;
% ~70 t burning 1/3rd of the fuel for a 737-700-like aircraft.
designMass = m0 - 5*u.t;

sectionCd = 0.011;
nonWingCd = 0.01;
sRef = 140.54*u.sqm; % From my base planform; JAWA ref: 125.23 sqm.
bRef = 112*u.ft+7*u.in; % 737-700 spec w/o winglets (w/ winglets: 117'5")
% x_ac = 4.24*u.m; % From my base planform.
cRef = 4.829*u.m; % From my base planform.
arRef = bRef.^2./sRef;

leSweep = 25.3*u.deg; % Measured from drawing.
% qcSweep = 25*u.deg; % Spec.
leSweepInboard = 39.8*u.deg; % Measured from drawing.
% kinkBl = 15*u.ft+10*u.in; % 4.826 m
kinkBl = 10.1*u.m/2;
dihedral = 6*u.deg; % Spec.
% wingRootC = 8.86*u.m; % - 2.387*u.m; % Measure. Alt: 25*u.ft+10.12*u.in;
basicTipSpan = 0.5*u.m; % Outboard of slats and ailerons.
fuseWidth = 3.8*u.m; % 3.76*u.m;
bodySideC = 7.27*u.m; %wingRootC-fuseWidth/2*tan(leSweepInboard);

% wingTipC = 4*u.ft + 1.25*u.in;

% More 737 info for sanity checking at
% http://www.b737.org.uk/techspecsdetailed.htm

% ICAO Annex 14 - Aerodrome Reference Code
% (Aeroplane Wingspan; Outer Main Gear Wheel Span)
%
```

```

% Code A - < 15m (49.2'); <4.5m (14.8')
% Code B - 15m (49.2') - <24m (78.7'); 4.5m (14.8') - <6m (19.7')
% Code C - 24m (78.7') - <36m (118.1'); 6m (19.7') - <9m (29.5')
% Code D - 36m (118.1') - <52m (170.6'); 9m (29.5') - <14m (45.9')
% Code E - 52m (170.6') - <65m (213.3'); 9m (29.5') - <14m (45.9')
% Code F - 65m (213.3') - <80m (262.5'); 14m (45.9') - <16m (52.5')

%% Load stored results from baseline run for later calculation of deltas.
baseline = duel_wise_demos.winglet.baseline_values;

%% Initialize VLModel.
vm = VLModel;
vm.title = 'Winglet VL';
vm.iYsym = 1;
vm.sRef = double(sRef);
vm.Cref = double(cRef);
vm.Bref = double(bRef);

%% Build base wing
P0 = FlyingSurfacePanel;
P0.name = 'fuseCarryover';
P0.span = fuseWidth/2;
P0.rootChord = bodySideC;
P0.location = u.ft*[0 0 0];
P0.x = -P0.rootChord/3; % Approximation for getting root bending moments.

% Root to kink.
P1 = FlyingSurfacePanel;
P1.name = 'rootToKink';
P1.span = kinkBl - P0.span;
P1.rootChord = P0.tipChord;
P1.sweepLoc = 0;
P1.sweep0 = leSweepInboard;
P1.tipChord = P1.rootChord - P1.span.*tan(leSweepInboard);
P1.dihedral = dihedral;
P1.location = [P0.xTip, P0.yTip, P0.zTip];

% Kink to outboard aileron.
P2 = FlyingSurfacePanel;
P2.name = 'kinkToTip';
P2.span = bRef/2 - (P1.span + P0.span) - basicTipSpan;
P2.rootChord = P1.tipChord;
P2.tipChord = 1.75*u.m;
P2.sweepLoc = 0;
P2.sweep0 = leSweep;
P2.dihedral = dihedral;
P2.location = [P1.xTip, P1.yTip, P1.zTip];
P2.tipInc = S.wingTwist;

% Tip panel.
P3 = FlyingSurfacePanel;
P3.name = 'tipPanel';
P3.location = [P2.xTip, P2.yTip, P2.zTip];
P3.span = S.tipSpan;
P3.rootChord = P2.tipChord;
P3.rootInc = P2.tipInc;
P3.taperAngle = P2.taperAngle;
P3.tipChord = P3.tipChord.*S.tipChordShrinkRatio;
P3.sweepLoc = 1;
P3.sweep100 = P2.sweep100 + S.tipTeKink;
P3.tipInc = S.tipTwist + P2.tipInc;
P3.dihedral = dihedral + S.tipExtraDihedral;

P = [P0 P1 P2 P3];

wing = Wing;
wing.panels = P;
s1 = wing.makeVLSurface([], 'Nspan', 18, 'SSpace', -2, 'name', 'wing');
s1.Ydupl = [];

%% Winglet.
wl = FlyingSurfacePanel;

```

```

wl.name = 'winglet';
wl.location = [P3.xTip, P3.yTip, P3.zTip];

wl.sweepLoc = 0;
wl.xRot = S.wingletAngle;
wl.span = S.wingletSpan;
wl.rootChord = P3.tipChord;
wl.tipChord = wl.rootChord.*S.wingletTaperRatio;

wl.rootInc = S.wingletRootInc;
wl.tipInc = S.wingletRootInc + S.wingletTwist;
wl.sweep0 = leSweep + S.wingletExtraLeSweep;

winglet = Wing;
winglet.panels = wl;
s2 = winglet.makeVLSurface([], 'Nspan', 6, 'SSpace', -1, 'name', 'winglet');
s2.Ydupl = []; % Get rid of default mirroring.
% Conservatively ignoring parasite drag for bending moments.

%% Estimate weight delta.
baseTipMass = baseline.panels(4).area.*S.surfaceMassPerArea;
S.tipMass = P3.area.*S.surfaceMassPerArea;
S.wingletMass = wl.area.*S.surfaceMassPerArea;
S.addedVehicleMass = 2*(S.tipMass - baseTipMass + S.wingletMass);
S.emptyWeightChange = S.addedVehicleMass./(38*u.t); %*

if S.allowWeightIncrease
    m0 = m0 + S.addedVehicleMass;
    S.usefulLoadDeltaPax = 0;
else
    S.usefulLoadDeltaPax = -S.addedVehicleMass/(100*u.kg); %**
end

% Assuming a limit load factor per 14 CFR § 25.337 calculation negligibly
% affected by weight change and not clipping to range of [2.5 3.8], though
% acceptable due to primarily only examining deltas.
n = 2.1 + 24000*u.lb/(m0 + 10000*u.lb);
limitLoad = n.*m0*u.g0;

if S.allowWeightIncrease
    % Also increase analysis weight.
    analysisWeight = u.g0*(designMass + S.addedVehicleMass);
else
    analysisWeight = u.g0*designMass;
    % Assume that you can take out all that extra mass in fuel for this
    % particular design point.
end
nForBendingCase = limitLoad./analysisWeight;

%% Determine run conditions.
fc = FlightCondition;
fc.h = u.FL*380;
fc.wS = analysisWeight./sRef;
fc.M = 0.78;
% Assuming same cruise speed is mostly limited by Mach and won't change with
% weight or drag polar.

rn = {sprintf('a c %.15f', fc.cL), ...
    sprintf('a c %.15f', nForBendingCase.*fc.cL)};

%% Run vortex lattice.
if S.wingletSpan >= 5*u.cm % This is necessary for AVL to not die.
    vm.components = {[s1 s2]};
else
    vm.components = {s1};
end

vm.writeavl;
[fn, st] = runanalysis(vm.name, rn, {'fn' 'st'}, false); % 21 s

```

```

S.vlModel = vm;
S.vlRuns = rn;
S.stabilityOutput = st;

% Debugging plots:
% vm.showinavl;
% plot([P P4]); aircraftview plan

%% Get some interesting outputs.
S.panels = [P wl];
% Get area of panels. Note: cos(dihedral) = S/area.
areas = [S.panels.area];
S.flyingSurfaceArea = sum(areas);

qS = fc.q.*sRef;

% Root bending moment.
tipLoad = n.*S.tipMass*u.g0;
wingletLoad = n.*S.wingletMass*u.g0;

bendingRunInd = 2;
rollMoment = qS.*bRef.*fn(bendingRunInd).Cl + ... % negative moment.
    (tipLoad.*P3.y_bar_abs + wingletLoad.*wl.y_bar_abs);

% Using st for pitching moment instead of cm since it has extra sig figs.
pitchMoment = qS.*cRef.*st(bendingRunInd).Cmtot/2 + ...;
    (tipLoad.*P3.x_ac_abs + wingletLoad.*wl.x_ac_abs);

S.rootBendingMoment = sqrt(rollMoment.^2 + pitchMoment.^2);
% Real limit load will likely put some significant deflection into the wing
% s.t. the load distribution looks quite different. Also, this also omits other
% elements such as drooped aileron. Much of this can be captured with
% preferences that flag a "definitely ok" zone, a maybe zone, and a danger zone.
S.rootBendingChange = S.rootBendingMoment./baseline.rootBendingMoment - 1;

% Fwd spar location on winglet root chord
sparSweep = P2.sweepX(0.2); % Hypothetical fwd spar location.
fwdSparTipX = (P2.xTip + 0.2*P2.tipChord) + P3.span.*sin(sparSweep);
S.fwdSparOnWingletRoot = (fwdSparTipX - wl.xRoot)./wl.rootChord;

% Aft spar location on winglet root chord
sparSweep = P2.sweepX(0.7); % Hypothetical aft spar location.
aftSparTipX = (P2.xTip + 0.7*P2.tipChord) + P3.span.*sin(sparSweep);
S.aftSparOnWingletRoot = (aftSparTipX - wl.xRoot)./wl.rootChord;

% Drag and L/D. Turn this into something useful.
parasiteDragArea = sum(areas).*sectionCd;
CD0 = 2*parasiteDragArea./sRef; % 2 x for mirroring.
% drag = qS.*fn(1).CD; % Not useful due to rounding in fn.CD.
% CDi = st(1).CDind
% e = st(1).e;
CDi = fc.cL.^2./(pi*st(1).e.*arRef); % e has the most sig figs.
S.liftToDragRatio = fc.cL./(CD0+CDi+nonWingCd); % 16.3 for A320-200 in cruise.
S.liftToDragChange = S.liftToDragRatio./baseline.liftToDragRatio - 1;

S.cruiseDrag = analysisWeight./S.liftToDragRatio;
S.cruiseDragChange = S.cruiseDrag./baseline.cruiseDrag - 1;

% Overall span.
S.span = max(wl.yTip*2,P3.yTip*2);

% Tip aspect ratio
S.tipAspectRatio = (wl.structuralspan + P3.structuralspan).^2./...
    (wl.area + P3.area);
% Limit this to something like 4.5 or 5;

S.wingletStructuralAr = wl.structuralspan.^2./wl.area;
% Limit this to something like 4;

```


B4 Wing redesign with winglet setup script

```

cch all
clear global % Reset memoization.
rng(0); % Make repeatable.

%% SET UP DESIGN SPACE.
o = DesignSpace();
o.systemModel = @duel_wise_demos.winglet.sys_winglet;

%% SET UP DESIGN VARIABLES.
% MAJOR INPUTS: tipSpan, tipChordShrinkRatio, wingletSpan, wingletRootInc,
% wingletTwist, wingtipMassPerArea

% OTHER INPUTS: allowWeightIncrease, tipTwist, tipExtraDihedral, wingletAngle,
% wingletExtraLeSweep, wingletTaperRatio

% The preference currency is percentage points of cruise drag improvement.

%% %%%% INPUTS %%%%
dx = 0.05; % Smallest meaningful step for low-precision method.

%% tipSpan
v = DesignVariable("tipSpan", 'label', 'tip span', 'units', 'm');
v.lbvub = [.2 .3 0.5 2.5 6]; % Set UB to > ~1.3 to allow for >36 m span.
% 737-700 winglet: 2.15/2

% Overall span limit will take care of UB.
% LB is just so VL elements don't get too squished.

v.smallestMeaningfulStep = dx;

o.v.tipSpan = v;

%% tipChordShrinkRatio
v = DesignVariable("tipChordShrinkRatio", 'label', 'tip \lambda ratio');
v.lbvub = [.2 .3 .6 1 1];
% 737-700 winglet: 0.73.

v.smallestMeaningfulStep = dx;

o.v.tipChordShrinkRatio = v;

%% wingletSpan
v = DesignVariable("wingletSpan", 'label', 'winglet span', 'units', 'm');
v.lbvub = [.05 0.06 2.315 4 5]; % 5 cm span still runs without killing AVL.

v.prefAbscissa = [0 0.3 2];
v.prefOrdinate = [0 1 1];
v.prefInfo{2} = 'If it is going to be that small, better nothing at all.';

v.prefUncertainty = 2;

v.smallestMeaningfulStep = dx;

o.v.wingletSpan = v;

%% wingletRootInc
v = DesignVariable("wingletRootInc", 'label', '\alpha_{inc,root}', 'units', '°');
v.lbvub = [-5 -1 2 4 10];

v.smallestMeaningfulStep = dx;%*2;

o.v.wingletRootInc = v;

%% wingletTwist
v = DesignVariable("wingletTwist", 'label', 'twist', 'units', '°');
v.lbvub = [-10 -6 -3 0 4];

v.smallestMeaningfulStep = dx;%*2;

o.v.wingletTwist = v;

```

```

%% surfaceMassPerArea
v = DesignVariable("surfaceMassPerArea", 'label', 'areal mass', 'units', 'lb/sqft');
v.parameter = true;
v.value = 10;
v.startLowerBound = 4;
v.startUpperBound = 20;
v.free = 0; % Fix this to keep this test case conceptually simple.
v.distribution = 6;

o.v.surfaceMassPerArea = v;

%% allowWeightIncrease
v = DesignVariable("allowWeightIncrease");
v.discrete = true;
v.value = 1;
v.startLowerBound = .5;
v.startUpperBound = 2.5*(1-eps);

v.prefAbscissa = [false true];
v.prefOrdinate = [0 0]; % Pref taken care of by useful load prefs.
v.prefUncertainty = 2;

o.v.allowWeightIncrease = v;

%% wingletAngle
v = DesignVariable("wingletAngle", 'units', '°');
v.lbvub = [6 45 86 90 96];
v.free = 0;

v.smallestMeaningfulStep = .1;

o.v.wingletAngle = v;

%% %%%% OUTPUTS %%%%
% MAJOR OUTPUTS: usefulLoadChange, rootBendingChange, fwdSparOnWingletRoot,
% aftSparOnWingletRoot, cruiseDragChange, span
% OTHER OUTPUTS: liftToDragChange

%% usefulLoadChange
v = DesignVariable("usefulLoadDeltaPax", 'label', '\DeltaUL (pax)');
v.ioType = 'output';
v.prefAbscissa = [-1 -0.5 0 1];
v.prefOrdinate = 0.2*[0.15 0.05 0 -0.02];
v.prefInfo{1} = 'losing a pax';
v.prefUncertainty = 2;

o.v.usefulLoadChange = v;

%% addedVehicleMass
v = DesignVariable("addedVehicleMass", 'label', '\DeltaW_{tips}', 'units', 'kg');
v.ioType = 'output';

o.v.addedVehicleMass = v;

%% rootBendingChange
v = DesignVariable("rootBendingChange", 'label', '\DeltaM_{root}', 'units', '%');
v.ioType = 'output';
v.prefAbscissa = [-1 0 5 10 15];
v.prefOrdinate = [0 0 .3 1 2];
v.upperBound = 15;
v.prefInfo{4} = 'probably the limit of what can be be glossed over for
aeroelasticity';
v.prefFlags{5} = 'getting unmanagably large bending moments';
v.prefFlags{6} = 'getting infeasibly large bending moments';
v.prefUncertainty = 3;

o.v.rootBendingChange = v;

%% fwdSparOnWingletRoot
v = DesignVariable("fwdSparOnWingletRoot", 'label', 'x_{spar, fwd}', 'units', '%', ...

```

```

        'description','put the spar in the meaty bit if possible.');
```

v.ioType = 'output';
v.prefAbscissa = [0 12.5 20 70 80 100];
v.prefOrdinate = 0.2*[1 1 0 0 1 1];
v.prefUncertainty = 2;

o.v.fwdSparOnWingletRoot = v;

%% aftSparOnWingletRoot
v = DesignVariable("aftSparOnWingletRoot","label",'x_{spar,aft}','units','%','...
 'description','put the spar in the meaty bit if possible.');

v.ioType = 'output';
v.prefAbscissa = o.v.fwdSparOnWingletRoot.prefAbscissa;
v.prefOrdinate = o.v.fwdSparOnWingletRoot.prefOrdinate;
v.prefUncertainty = o.v.fwdSparOnWingletRoot.prefUncertainty;

o.v.aftSparOnWingletRoot = v;

%% cruiseDragChange
v = DesignVariable("cruiseDragChange","label",'Δ_{cruise}','units','%','...
 'description','surrogate for cruise fuel burn');

v.ioType = 'output';
v.upperBound = 0; % Don't allow any designs that increase cruise fuel burn.
v.prefAbscissa = [-2 0];
v.prefOrdinate = [-2 0];

o.v.cruiseDragChange = v;

%% span
v = DesignVariable("span","units",'m');

v.ioType = 'output';
v.upperBound = 36;
v.upperBound = 52;
v.prefAbscissa = [26 36 38 48];
v.prefOrdinate = [1 1.1 4 4.1]*2/3;
v.prefUncertainty = 2;

o.v.span = v;

%% wingletStructuralAr
v = DesignVariable("wingletStructuralAr","label",'AR_{wl,structural}');

v.ioType = 'output';
v.upperBound = 4.5;
v.prefAbscissa = [0 3 4 5];
v.prefOrdinate = [0 .1 .3 .8];
v.prefUncertainty = 3;

v.prefFlags{4} = 'winglet aspect ratio getting quite high for structure';

o.v.wingletStructuralAr = v;