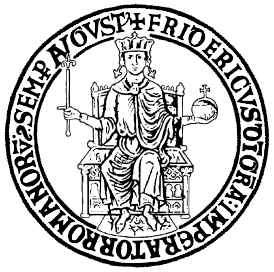

Università degli Studi di Napoli Federico II

Scuola Politecnica e delle Scienze di Base

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE
Corso di Laurea in Ingegneria Aerospaziale

Tesi di Laurea Magistrale
in
Ingegneria Aerospaziale ed Astronautica

Development of a Java Framework for Parametric Aircraft Design

The Performance Analysis Module

Candidato:
Vittorio Trifari
Matricola M53/411

Relatori:
Prof. Fabrizio Nicolosi
Prof. Agostino De Marco

Anno Accademico 2014/2015

Abstract

This thesis has the main purpose of providing a comprehensive overview about the development, in Java, of a software dedicated to the preliminary design of an aircraft, focusing on the performance analysis module.

The point of view from which this subject will be observed expects first to define methodologies and theoretical aspects necessary for the examined performance calculation for then to show, in more detail, the implementation of these latter within the software; this will be seen both from the point of view of the developer, through a detailed explanation of the architecture of the different calculation modules, both of a potential user developer, by showing some commented examples of use supplied with graphical and numerical results suitable, among other things, also to the validation of calculations performed.

Sommario

Questo lavoro di tesi ha lo scopo principale di fornire una panoramica esaustiva circa lo sviluppo, in ambiente Java, di un software dedicato al progetto preliminare di un aeromobile, concentrandosi sull'aspetto dell'analisi delle performance.

Il punto di vista con il quale verrà affrontato questo argomento prevede dapprima di inquadrare le metodologie e gli aspetti teorici necessari per la stima delle performance in esame per poi mostrare, più nel dettaglio, l'implementazione di questi ultimi all'interno del software; ciò sia dal punto di vista dello sviluppatore, attraverso la spiegazione dettagliata dell'architettura dei vari moduli di calcolo, sia dal punto di vista di un potenziale utente, tramite alcuni esempi d'uso commentati e corredati di risultati grafici e numerici atti, tra l'altro, anche alla validazione dei calcoli svolti.

Contents

1 Introduction and software overview **5**
- 1.1 Main purposes . . . 5
  - 1.1.1 Aircraft design ovrewiev . . . 6
  - 1.1.2 Java: motivation and main features . . . 8
- 1.2 Input file structure . . . 10
- 1.3 Java Program toolchain for Aircraft Design (JPAD) Library . . . 14
  - 1.3.1 `aircraft` package . . . 16
  - 1.3.2 `cad` package . . . 18
  - 1.3.3 `calculators` package . . . 18
  - 1.3.4 `database` package . . . 19
  - 1.3.5 `writers` package . . . 19
- 1.4 Aircraft model creation . . . 19
- 1.5 The Graphical User Interface (GUI) . . . 27
  - 1.5.1 Typical work session . . . 28
  - 1.5.2 CAD modelling . . . 30
- 1.6 Optimization process . . . 30

2 Payload-Range **32**
- 2.1 Theoretical background . . . 32
- 2.2 Java class architecture . . . 36
- 2.3 Case study: ATR-72 and B747-100B . . . 39

3 Specific Range and Cruise Grid **45**
- 3.1 Theoretical background . . . 45
- 3.2 Java class architecture . . . 49
- 3.3 Case study: ATR-72 and B747-100B . . . 52

4 High lift devices effects **61**
- 4.1 Theoretical background . . . 62
  - 4.1.1 ΔC_{l0} and ΔC_{L0} calculation . . . 65
  - 4.1.2 $\Delta C_{l\max}$ and $\Delta C_{L\max}$ calculation for trailing edge and leading edge devices . . . 68
  - 4.1.3 $C_{l\alpha,\text{flap}}$ and $C_{L\alpha,\text{flap}}$ calculation . . . 73

4.1.4 $\Delta\alpha_{stall}$ calculation for trailing edge and leading edge devices 73
4.1.5 Further effects calculations 74
4.2 Java class architecture 78
4.3 Case study: ATR-72 and B747-100B 81

5 Take-off and Landing performance **88**
5.1 Theoretical background 88
5.1.1 All Operative Engines (AOE) take-off run 89
5.1.2 One Inoperative Engine (OEI) take-off run and balanced field lenght 94
5.1.3 Landing distance calculation 95
5.2 Java class architecture 97
5.2.1 Take-Off 97
5.2.2 Landing 104
5.3 Case study: ATR-72 107

Conclusions **127**

Appendices

A HDF databse creation and reading **129**
A.1 Creation of a database using MATLAB 129
A.2 Reading data from an HDF database in JPAD 132

Bibliography **134**

Glossary **136**

Acronyms **139**

Chapter 1

Introduction and software overview

There must be a beginning of any great matter,
but the continuing unto the end
until it be thoroughly finished
yields the true glory
– Sir Francis Drake, 1587

1.1 Main purposes

This thesis deals with the development of a software framework that could help, during preliminary and conceptual design phases, in finding a configuration which satisfies several basic requirements and, eventually, a constrained optimum.

Since the end of 80's many software dealing with aircraft design have been produced with the aim of having some design framework to be used for teaching and professional purposes in aircraft design. However, some new software have recently followed innovative approaches, considering concepts like Knowledge-Based Engineering (KBE) and Multi-disciplinary Design Optimization (MDO), and have highlighted the necessity of an efficient graphical user interface in order to make results easily exploitable with external software. The software in exam has been designed to embrace different types of analysis pertaining disciplines such as Aerodynamics, Structures, Weights and Propulsion. Furthermore, since in the conceptual and preliminary design phases a lot of different configurations have to be analyzed, the latter has been conceived to provide results in a relatively short computational time. This need often requires to rely on semi-empirical methods so that, to improve their accuracy, a comprehensive study of the methods available in literature has been firstly carried out; each method has been tested against experimental data (produced in-house or drawn from literature) so that statistical quantities could be estimated either to find the best method currently available or to make a merger of different methods. Whenever possible, application results have been compared with values found in literature or in-house databases.[3]

1.1.1 Aircraft design ovrewiev

Modern aerospace design begins with the desire of mankind to achieve powered, heavier-than-air flight. Early aerospace designs were characterized not only by a great deal of trial and error but also by a not inconsiderable amount of analysis. As the technologies used in aerospace applications have developed, the goals have become vastly more sophisticated. It is possible to view the first 75 years of aerospace design as a series of stages: first came the early pioneers, working at a time when many were skeptical about the benefits of flight or even if it was possible at all. Next came the phase where outstanding individuals dominated aeronautical science and design during the interwar and early postwar years. This was followed in the 1950s and 1960s by a greater emphasis on design teams and the massive expansion of the great aerospace companies that we recognize today. Following these stages, the advent of Computer-Aided Engineering (CAE), design and manufacture in the mid-1970s transformed activities in aerospace with an increasing focus on faster design cycles, more accurate predictive capabilities and the rise of computer science. At the same time, the great drawing offices of the 1950s declined and fewer people were involved in the design of each individual component.[14]

Recent developments in information technology have had a major impact upon the aeronautical design process. Computer aided design techniques are used to produce digital data for, for example, application to numerically controlled machining. Virtual reality concepts are employed to visualise threedimensional installation and operational aspects, at least in part replacing full-scale mockups. While these procedures are particularly relevant to the detail phases of the design process their application commences as soon as a requirement is defined and numerical data are derived from it.[18]

The design of an aircraft is a large undertaking, requiring the team efforts of many engineers having expertise in the areas of aerodynamics, propulsion, structures, flight control, performance, and weights. As the design takes shape, specialists are called into design such components as the crew station, landing gear, interior layout, armament location, and equipment installation. The completed aircraft design is a compromise of the best efforts of many talented engineers. The different design groups they represent must work together to produce the most efficient flight vehicle. It should be clear that the design process is a very involved integration effort, requiring the pulling together and blending of many engineering disciplines.[23]

A comprehensive and concise description of the general design objective of a transport aircraft can be found in [25]; here this objective is described as the transport of a payload over a distance between airports against minimum costs (for example at an optimum speed). The driving parameters to accomplish this goal are the followings.

- Engine characteristics
- Aerodynamic efficiency, $C_{L\max}$, buffet boundary
- Weight

Issues	Civil	Military
Dominant design criteria	Economics and safety	Mission accomplishment and survivability
Performance	Maximum economic cruise	Adequate range and response
	Minimum off-design penalty in wing design	Overall mission accomplishment
Airfield environment	Moderate-to-long runways	Short-to-moderate runways
	Paved runways	All types of runway surfaces
	High-level ATC and landing aides	Often spartan ATC, etc
	Adequate space for ground manoeuver and parking	Limited space available
System complexity and mechanical design	Low maintenance - economic issue	Low maintenance — availability issue
	Low system cost	Acceptable system cost
	Safety and reliability	Reliability and survivability
	Long service life	Damage tolerance
Government regulations and community acceptance	Must be certifiable (FAA, etc.) • Safety oriented	Military standards • Performance and safety • Reliability oriented
	Low noise mandatory	Low noise desirable • Good neighbour in peace • Reduced Detectability in war

Figure 1.1 Transport aircraft design objectives and constraints. Source: American Institute of Aeronautics and Astronautics (AIAA) Paper No 77-1795

It should be emphasised that estimating the weight and aerodynamic characteristics with sufficient accuracy at an early stage of the design is really an art, but an important one and a crucial one; in fact, it determines the initial quality of the design. The airworthiness requirements then require the aircraft to be safe, for example its flight handling characteristics (stability and control) must be satisfactory. The aircraft should also be reliable, which implies that:

- Systems (such as navigation equipment) must be adequate
- Systems hall be reliable and sufficiently redundant

In civil air transport, designing a family of aircraft has become a standard procedure. Two approaches can be recognised.

- In the first approach, several versions of the aircraft are developed more or less simultaneously from the start of a programme. These different versions can have different take-off weights, fuselage lenghts, etc.
- In the second approach, growth versions of the aircraft are developed (long) after the first development round has been completed. These growth versions usually have considerable modifications and associated costs

Figure 1.1 summarizes the main design objectives to be followed in both a civil or a military aircraft project.

Prior to designing any new aircraft, it is essential to see what the competition is offering. If the designed aircraft is not decidedly better than what exists in the market, it would be foolish to “bet your company” on developing a new design. The market survey should rigorously

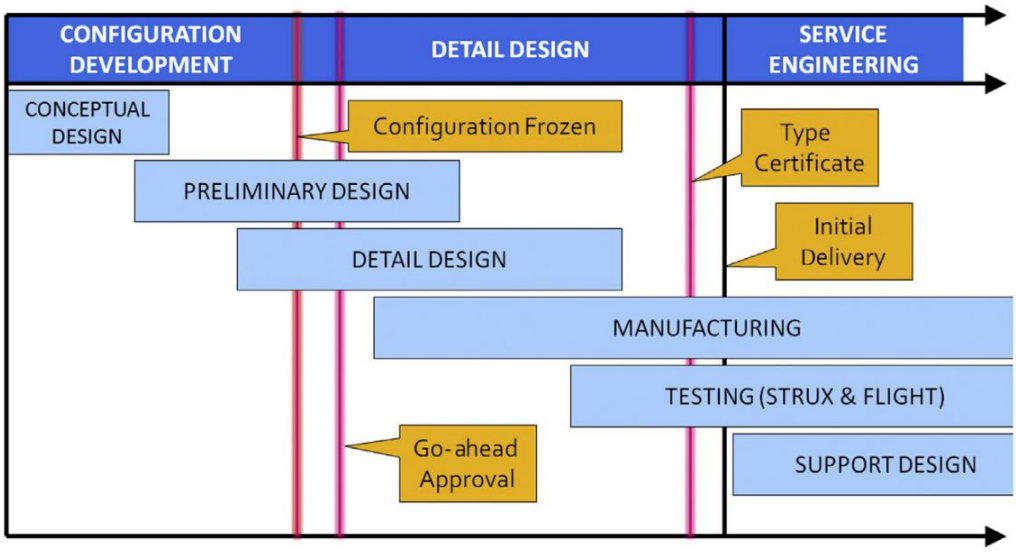

Figure 1.2 Aircraft design process for Torenbeek (1986)

examine three or four existing aircraft for which information is available and which most closely satisfy the assigned mission.[26]

After the market survey phase is complete, the aircraft design process can start. Generally speaking it involves several distinct phases which can be resumed as follows.

1. **Requirements phase**
2. **Conceptual design phase**, which encompasses sizing of the most promising overall aircraft concept and proof of its feasibility. Having a typical duration between 4 to 6 months for a business aircraft and 9 to 12 months for a mid-size airliner, conceptual design is characterized by cyclic design improvements and complexity increasing in time
3. **Preliminary design phase**, which aims at specifying the design concept at the main component level, sometimes including subsystem trades. Preliminary design typically lasts between 12 and 16 months
4. **Detail design phase** which begins when a management decision is taken to continue and give the project go-ahead. This development phase is entered soon after the aircraft is committed to production and lasts between two and three years. The decision to freeze the configuration is taken early in the detail design phase when changes in the product definition are no longer appropriate.
5. **Proof of concept aircraft construction and testing phase**

1.1.2 Java: motivation and main features

The choice of Java as the programming language was driven by several considerations, as explained in [1]. These include the following:

- The language should be widely supported; this to avoid the case of many valid aircraft design applications and libraries that became obsolete due to the aging of the programming language used to build them
- The language should promote the use of open source libraries, especially for I/O tasks and for complex mathematical operations
- The language and the companion Integrated Development Environment (IDE) should provide a widely supportedGUI framework and a GUI visual builder
- The language should support and promote modularity

The Java programming language meets all these requirements; moreover it is backed by Oracle and by a huge community of developers so it is continuously updated. Also, advanced and free IDEs (such as Eclipse or Netbeans) allow programmers to streamline and simplify the development process; in particular, the Eclipse IDE has been chosen to develop software. Being Java a pure object oriented programming language, it greatly encourages and simplifies modularization. Each module (package) can be programmed quite independently so that it is relatively easy to divide the work among several programmers. This is essential since the amount of classes and calculations needed to abstract, manage and analyze the entire aircraft is very large (presently the whole project counts more than 56000 lines of code). For such a reason the establishment of common practices and the adherence to fundamental principle of software development (*Don't Repeat Yourself*, *Separation of Concerns*, *Agile software development*) are equally important.

According to the TIOBE index, in terms of popularity, during the last year the Java language has proven to be the most used among the developers community as shown in the following figure.

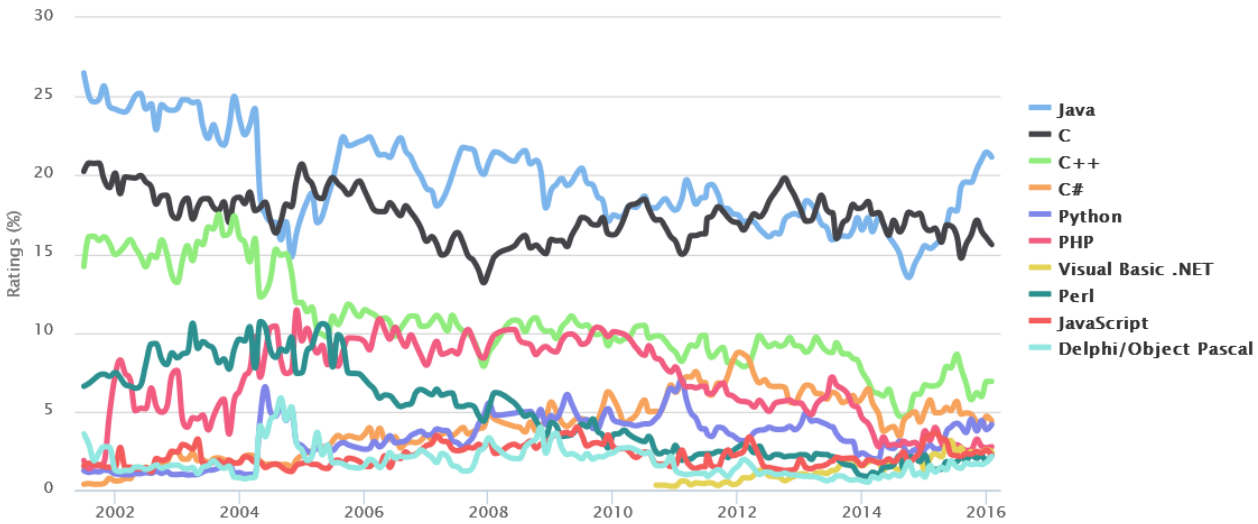

Figure 1.3 TIOBE Programming Community Index (www.tiobe.com)

1.2 Input file structure

In order to read databases, or to read an entire aricraft parametric model, the input file type chosen is the XML. The file extension stands for *eXtensible Markup Language* and is a markup language that defines a set of rules for encoding documents in a format which is both *human-readable* and *machine-readable*; moreover its design goals of emphasize simplicity, generality and usability across the Internet. XML is a textual data format with strong support via Unicode for different human languages and, although the design of XML focuses on documents, it is widely used for the representation of arbitrary data structures such as those used in web services. It is defined *Markup Language* due to the use of tags that describes the content; while is considered *extensible* because the markup symbols are unlimited and self-defining, so that it's possible to use a personal tag for each data. In this way to read an XML file results relatively simple.[6]

The key concepts of an XML File Format are the followings:

- Markup symbol (tag)
- Attribute
- Tree structure

As mentioned, each part of the test is contained between an *opening markup symbol* and an *end markup symbol* that expressed the meaning of the text.

```
<name>Test XML</name>
```

Figure 1.4 Use of markup symbols in XML language.

In addition to tag name, the markup symbols may contain also some *attributes* that introduce more informations such as the unit of measure.

```
<tag attribute1='value' attribute2='value'> text </tag>
```

Figure 1.5 Use of attributes in XML language.

An XML file has a tree structure where there are external knots that branch into internal knots. In order to read an XML file inside JPAD it is necessary, first of all, to give the file path; at this point the class `JPADXmlReader` opens the file and the methods of the class `MyXMLReaderUtils` reads the useful data from the XML having the tag path as input.

It is possible to read data as `Amount`[10], namely with units of measurement or as `double`. The unit of measurement is written in the attributes of data in XML file. Likewise it is possible to write output data on XML file using the `JPADDataWriter` class. First of all it is necessary to define and build the XML tree structure; after that each variable is associated with a name that is the markup symbol of the XML file.

In order to simplify, for a potential user developer, the definition of the parametric model of an aircraft with the purpose to give it as input for the library analyses, a large use of XML

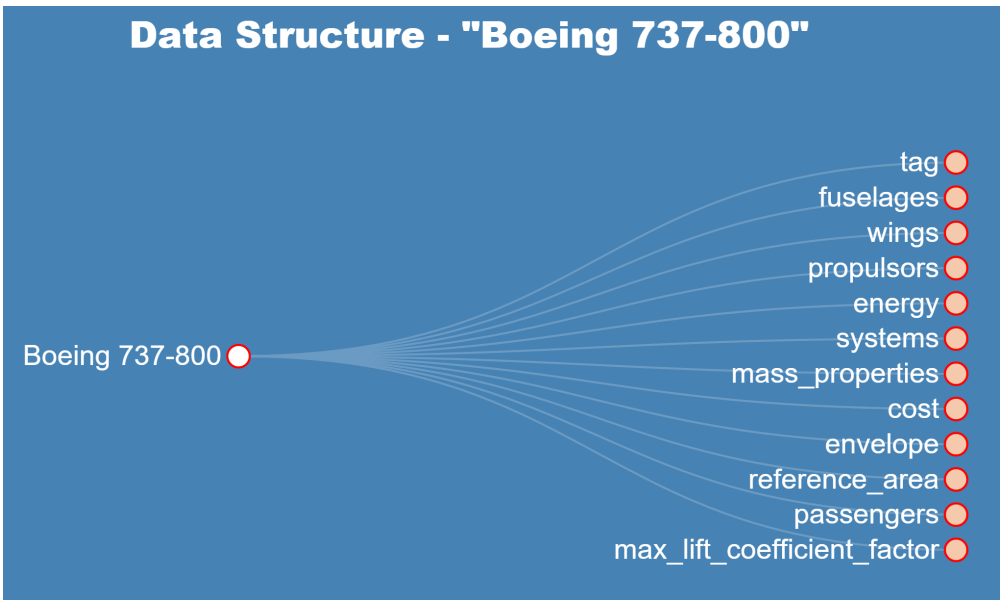

Figure 1.6 Aircraft parametric model in SUAVE

files has been made. More in detail, using as model the input file structure, from figure 1.6, designed by Stanford University in their recent project SUAVE[27], the following folder tree prototype has been designed.

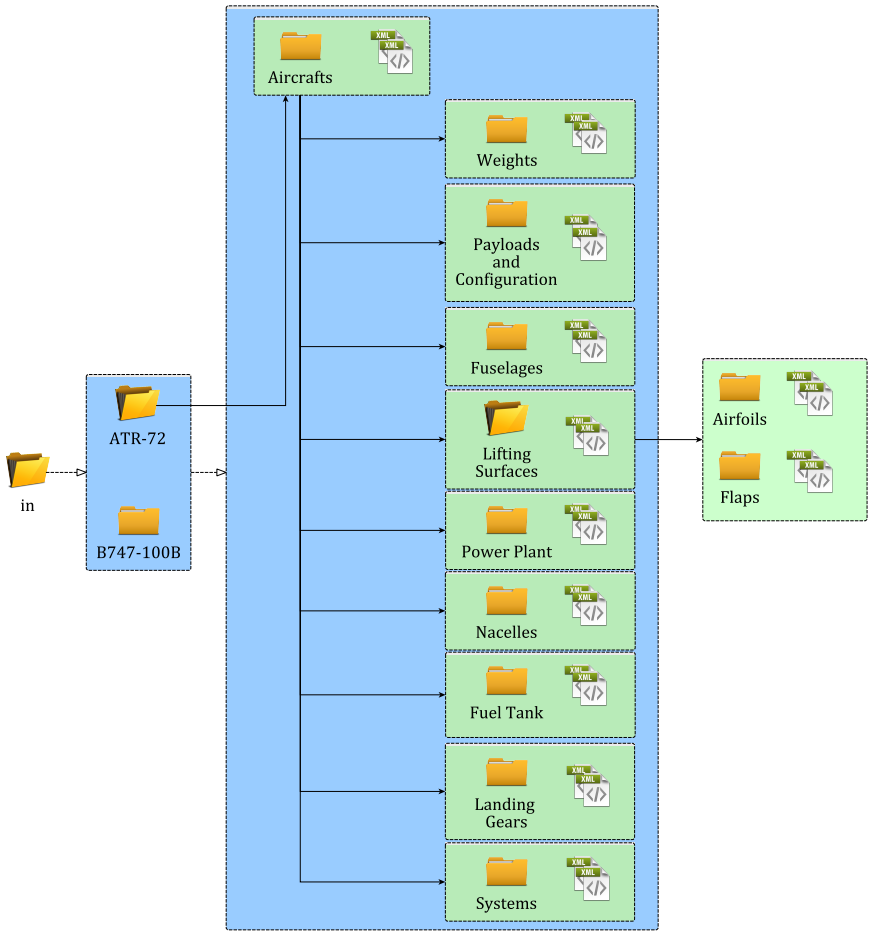

Figure 1.7 Input file folder tree prototype for JPAD

The guideline which has driven the design of this structure is that the user can manage different aircrafts models at the same time by assigning to each of them a related subfolder inside the input main folder. At this point, since the library purpose is also to perform optimization upon the aircraft geometry comparing many different configurations, inside the latter folder a series of subfolders have been created to manage each component that, moreover, can be present in many different versions; for example there can be many fuselages with different lengths in order to analyze a streched version of the initial baseline.

XML files are in charge of store all data realted to each component as well as call other XML files, of a lower level, in order to read their data; for example the wing XML file calls for airfoils XML files as shown in figure 1.7. The possibility of setting up a hierarchy among the different XML files comes in handy in order to reduce the dimension of each file; the idea, in fact, has been to confine data in different files which can be managed in a more intuitive way rather than having a single input file, divided into a lot of tags, that has the purpose of storing all aircraft's data. Moreover, having each component described, the user can simply work, at high level, by assembling the wanted aircraft through combining the chosen components XML files, as shown in listing 1.1.

```
<jpad_config>
    <aircraft>
        <wings>
            <wing type="WING" file="wing.xml">
                <position>
                    <x unit="m">12.0</x>
                    <y unit="m"> 0.0</y>
                    <z unit="m"> 2.0</z>
                </position>
                <rigging_angle unit="deg">2.0</rigging_angle>
            </wing>
            <wing type="HTAIL" file="htail.xml">
                <position>
                    <x unit="m">30.0</x>
                    <y unit="m"> 0.0</y>
                    <z unit="m"> 4.0</z>
                </position>
                <rigging_angle unit="deg">0.0</rigging_angle>
            </wing>
            <wing type="VTAIL" file="vtail.xml">
                <position>
                    <x unit="m">30.0</x>
                    <y unit="m"> 0.0</y>
                    <z unit="m"> 5.0</z>
                </position>
                <rigging_angle unit="deg">0.0</rigging_angle>
            </wing>
        </wings>
        <fuselages>
            <fuselage file="fuselage.xml">
                <position>
                    <x unit="m">0.0</x>
```

```
                    <y unit="m">0.0</y>
                    <z unit="m">0.0</z>
                </position>
            </fuselage>
        </fuselages>
        <propulsors>
            <propulsor file="propulsor.xml">
                <position>
                    <x unit="m">12.0</x>
                    <y unit="m"> 5.0</y>
                    <z unit="m"> 0.0</z>
                </position>
                <tilting_angle unit="deg">2.0</tilting_angle>
                <yawing_angle unit="deg">0.0</yawing_angle>
            </propulsor>
            <propulsor file="propulsor.xml">
                <position>
                    <x unit="m">12.0</x>
                    <y unit="m">-5.0</y>
                    <z unit="m"> 0.0</z>
                </position>
                <tilting_angle unit="deg">2.0</tilting_angle>
                <yawing_angle unit="deg">0.0</yawing_angle>
            </propulsor>
        </propulsors>
    </aircraft>
</jpad_config>
```

Listing 1.1 Prototype of the aircraft XML file

```
<jpad_config>
    <wing mirrored="TRUE">
        <panels>
            <panel id="Inner panel">
                <semispan unit="m">4.0</semispan>
                <dihedral unit="deg">1.5</dihedral>
                <inner_section>
                    <airfoil file="naca63209.xml"/>
                    <geometric_twist unit="deg">0.0</geometric_twist>
                </inner_section>
                <outer_section>
                    <airfoil file="naca63209.xml"/>
                    <geometric_twist unit="deg">0.0</geometric_twist>
                </outer_section>
            </panel>
            <panel id="Outer panel" linked_to="Inner panel">
                <semispan unit="m">7.0</semispan>
                <dihedral unit="deg">3.5</dihedral>
                <outer_section>
                    <airfoil file="naca63209.xml"/>
                    <geometric_twist unit="deg">-2.5</geometric_twist>
                </outer_section>
            </panel>
```

```
24          </panels>
25          <symmetric_flaps>
26              <symmetric_flap id="Inner flap" type="SINGLE_SLOTTED">
27                  <inner_station_spanwise_position type="PERCENT_SEMISPAN"
        ref_to="FULL_SEMISPAN">
28                      <value>0.15<value/>
29                  </inner_station_spanwise_position>
30                  <outer_station_spanwise_position type="PERCENT_SEMISPAN"
        ref_to="FULL_SEMISPAN">
31                      <value>0.45<value/>
32                  </outer_station_spanwise_position>
33                  <percent_chord value="40"/>
34                  <angle_range unit="deg">[0.0,40.0]</angle_range>
35              </symmetric_flap>
36          </symmetric_flaps>
37          <slats>
38              <slat linked_to="Inner flap" type="SINGLE">
39                  <percent_chord value="10"/>
40                  <angle_range unit="deg">[0.0,10.0]</angle_range>
41              </slat>
42          </slats>
43          <asymmetric_flaps>
44              <asymmetric_flap id="Aileron" type="PLAIN">
45                  <inner_station_spanwise_position type="PERCENT_SEMISPAN"
        ref_to="FULL_SEMISPAN">
46                      <value>0.75<value/>
47                  </inner_station_spanwise_position>
48                  <outer_station_spanwise_position type="PERCENT_SEMISPAN"
        ref_to="FULL_SEMISPAN">
49                      <value>0.97<value/>
50                  </outer_station_spanwise_position>
51                  <percent_chord value="20"/>
52                  <angle_range unit="deg">[-5.0,10.0]</angle_range>
53                  <differential_deflection_factor value="1.0"/>
54              </asymmetric_flap>
55          </asymmetric_flaps>
56      </wing>
57  </jpad_config>
```

Listing 1.2 Prototype of the wing XML file

1.3 JPAD Library

In this paragraph an overview of the library JPAD will be presented. JPAD is a Java library developed at the University of Naples Federico II, conceived as a fast, reliable and user friendly computational aid for aircraft designers in the conceptual and preliminary design phases. The ultimate goal of such a tool is to perform a parametric, multi-disciplinary analysis of an aircraft and then search for an optimized configuration. An important design requirement is related to its interoperability with other engineering analysis tools; in fact, the application can be easily integrated into a comprehensive aircraft optimization cycle.

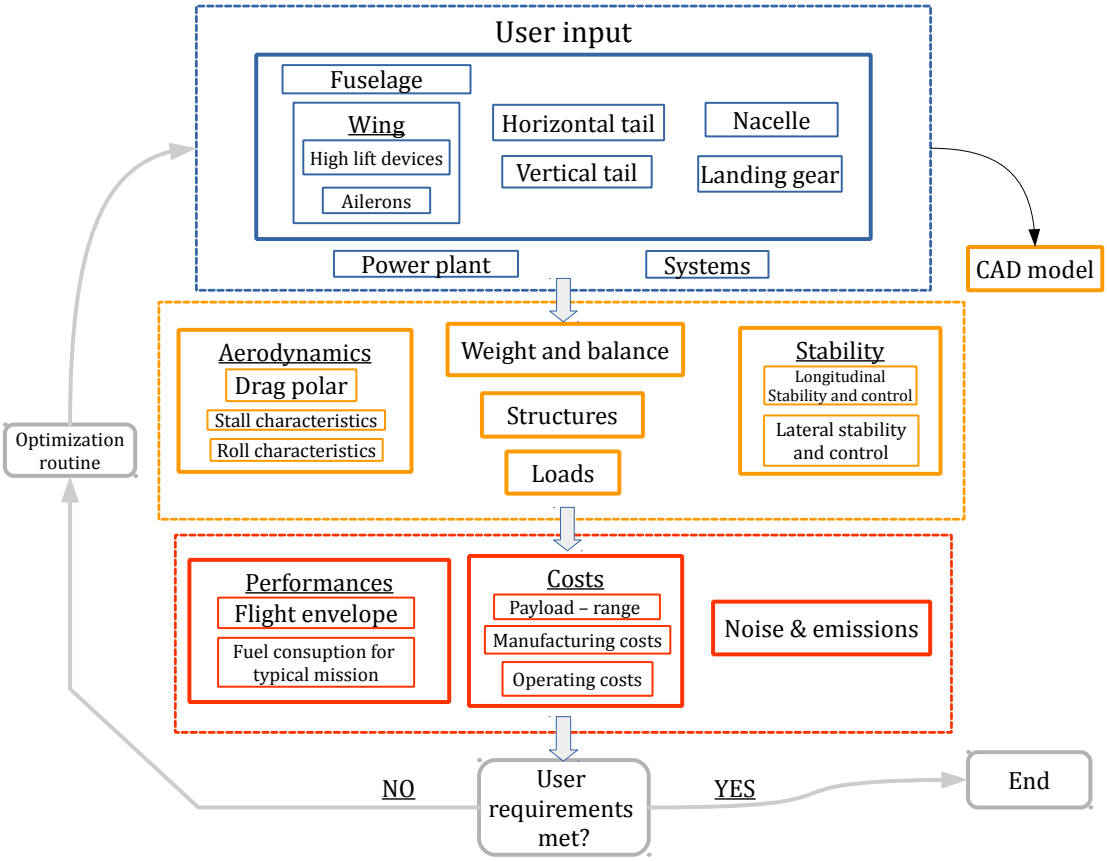

Figure 1.8 Software calculation modules

JPAD library is actually divided into three package: `JPADConfigs`, `JPADCore` and `JPADSandbox`. Each package is organized in several classes, or more packages, in order to have a clear and simple classification. The source code has been extensively commented following Javadoc practices; this enabled the developer to automatically generate documentation using Doxygen[32]. The main package of JPAD library is `JPADCore` where all aircraft's components, manager classes of the each calculator and the calculators themselves are located.

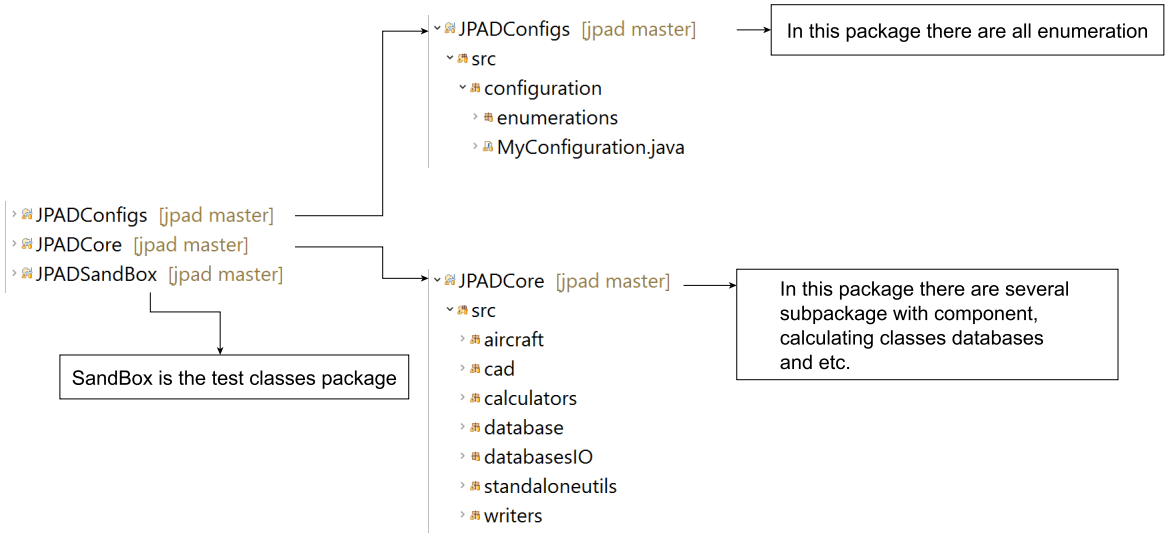

Figure 1.9 Software behavior flowchart

`JPADCore` contains the following sub-packages:

1. `aircraft`, this package contains all aircraft's component classes, their calculators and the related managers and also the operating condition's class.
2. `cad`, this package contains everything concerning the creation of CAD model.
3. `calculators`, in this package there are the classes, divided in package by type of analysis, whose methods implement the calculation formulas.
4. `database`, this package contains the classes that manage the database .h5 whose data are useful for the analysis.
5. `standaloneutils`, in this package there are several calculation classes with mathematical tools to support the analysis.
6. `writers`, this class is in charge of writing files.

1.3.1 `aircraft` package

Each component (for example the fuselage, the wing or the engine) is therefore defined in its own class located in a dedicated sub-package. In each of these there are a class for the object definition and a manager for the analysis; these manager classes uses the utilities located in the `calculator` package to perform the wanted analysis. The sub-package in the `aircraft` package are the following:

- `auxiliary`, which manages the airfoils
- `component`, which contains the `Aircraft` class, the aircraft components packages (fuselage, lifting surfaces, nacelles, power plant, fuel tank, landing gear, systems) and, inside them, their manager analysis classes used to call the related analysis methods.
- `OperatingConditions`, this class contains all the data related to atmosphere conditions (which are currently derived from altitude value using the 1976 ISA model), current speed (Mach number), gravitational acceleration (9,807 m/s^2) and sea level pressure (101325 Pa). An `OperatingConditions` object can be created regardless of the aircraft configuration as the class is entirely self contained: the aircraft could also be undefined at the time of the object creation. If the user wants to run several analysis at different operating conditions, an `OperatingConditions` instance has to be created for each one.

Regarding aircraft's components, their organization inside the `aircraft` package can be described as follows.

Basic data for describing the fuselage are contained in the `Fuselage` class. This holds all the fuselage overall properties, such as the length, maximum width and maximum height, length ratios between the nose and the tail parts length to the constant section part length, number of decks and so on. Other classes in the package manage the shape of fuselage sections and outlines (which define the fuselage shape in the xz and xy planes) and aerodynamics calculations `FusAerodynamicsManager` class).

The data relating to lifting surfaces are contained in the `LiftingSurface` class. Since lifting surfaces of an aircraft share several characteristics, a single class has been created to manage the wing, the horizontal and vertical tails and, eventually, the canard. The lifting surface specific category (that is, wing, horizontal tail etc.) is acknowledged through a `ComponentEnum` variable which has to be specified when creating the `LiftingSurface` object. By default, a lifting surface has three primary span stations: root ($\eta = 0$), middle and tip ($\eta = 1$). The middle station location is user-defined and is used to represent a change in the chords distribution law; such a change can be due to a lifting surface kink or to the beginning of a tapered part. If the wing is simple tapered the middle station can also be omitted.

The data relating to airfoils are handled by the `Airfoil` class, that creates for each lifting surface three airfoils, located respectively at $\eta = 0$, at the middle station and at $\eta = 1$. An `Airfoil` object holds the following quantities.

1. the position along the semispan, η;
2. twist value relative to root, ϵ_g;
3. zero lift angle of attack, $\alpha_{0\ell}$;
4. angle of attack value at the end of the linear part of $C_l(\alpha)$ curve, α^*;
5. stall angle of attack, α_{stall};
6. lift gradient of the linear part of $C_l(\alpha)$ curve, C_{ℓ_α};
7. minimum drag coefficient, $C_{d\,\text{min}}$;
8. lift coefficient at $C_{d,min}$, $C_l@C_{d\,\text{min}}$;
9. lift cofficient at the end of linear part;
10. maximum lift coefficient, $C_{l\,\text{max}}$
11. drag polar K factor;
12. aerodynamic moment coefficient gradient, C_{m_α};
13. aerodynamic center x coordinate, x_{ac};
14. aerodynamic moment coefficient with respect to the aerodynamic center, $C_{m_{ac}}$ or C_{m_0};
15. aerodynamic moment coefficient with respect to the aerodynamic center at stall, $C_{m_{ac},\text{stall}}$;
16. maximum thickness to chord ratio, t/c_{max};
17. x,z non dimensional coordinates.

At the time of writing, all these quantities have to be entered by the user. The `Airfoil` class provides the `populateCoordinateList` method which transforms the non dimensional coordinates, provided by the user, in order to obtain their actual coordinates, which takes

into account of actual chord length, Aicraft Construction Reference Frame (ACRF) position, sweep, twist and dihedral.

The power plant is defined in the package `powerPlant` inside `components`. In this package there is the `Engine` class, which initializes the data related to the single engine; while other class in the same package, named `PowerPlant` creates the parametric model of the power plant using a List[9] of objects creates by the `Engine` class.

1.3.2 `cad` package

Throughout the development of the application, great care has been given to the making of the CAD model for several reasons:

- It enables the user and the user developer to have an immediate feedback about the data provided to the application: if some geometrical parameter is wrong, the CAD model makes it impossible not noticing it
- It allows the user to run a CFD analysis with an external program. The CAD model has been in fact built so that it is ready to be meshed by an external mesher without any further adjustment
- It provides an accurate estimate of the wetted surface of each component

The creation of the CAD model was made possible by the *occjava* library, whose classes and methods have been used to build each component's model.

1.3.3 `calculators` package

This package includes all the calculators classes. Calculator classes have been created to evaluate quantities related with more than one component and can be summarized in the followings.

- `aerodynamics` package
- `cost` package
- `geometry` package
- `performance` package
- `stability` package

Each of these package contains calculators classes and concerning methods that implement all mathematical equations. These methods are called by the analysis manager classes that are in the package of the related components (such as fuselage, lifting surfaces etc.).

1.3.4 `database` package

In order to manage a large quantity of data, it has been necessary to read data from charts available in literature; for such a reason an extensive database has been built over the years by several of colleagues to digitalize this data in order to exploit it in a computer program. The data has been stored in an Hierarchical Data Format (HDF) file. In the `database` package there are all the classes in charge of managing and reading the .h5 files. For more information regarding the creation of the database and its reading method inside JPAD the reader can refer to appendix A.

1.3.5 `writers` package

This package contains the class `JPADDataWriter` and the class `JPADStaticWriteUtils` which are in charge of providing to a potential user developer several methods, based on the to *Apache POI* library, able to produce output files in both XML and XLS format. The second class, in particular, is a collection of static method which allows the user to perform a single specific operation like writing a single variable name, creating a single node inside an XML file or performing check on the output file. The possibility of creating an XLS output file, in addition to the XML, permits to create an output file in which different aircraft's data are stored in columns; in such a way the comparison between two or more aircraft (or simply between slightly different configurations of the same aircraft) is easier and effective.

1.4 Aircraft model creation

In JPAD it is possible to read an .xml file as input or generate an object whose data are written in the code. Both in the first and in the second case all needed variables are initialized with data relating the choosen aircraft. The difference between these two methods is that using an .xml file, user can define its own aircraft having a clear view about the needed data useful for the analysis. Contrariwise in order to perform test of program functionality, to use a default aircraft is the most simple way to generate a work object.

Actually it is possible to define two different aircraft in order to test the functionality of the application: ATR-72 and B747-100B.

The ATR-72, made by the French-Italian aircraft manufacturer ATR, is a stretched variant of the original ATR-42 that entered in service in 1989; it's powered by two turboprop engine and it's used typically as a regional airliner. The main purpose of its design was to increase the seating capacity throughout the stretching of the fuselage by 4.5m, an increased wingspan, more powerful engines and an increased fuel capacity by approximately 10%.[4]

The Boeing 747 is a wide-body commercial jet airliner and cargo aircraft, often referred to by its original nickname, Jumbo Jet, or Queen of the Skies. Its distinctive hump upper deck along the forward part of the aircraft makes it among the world's most recognizable aircraft, and it was the first wide-body produced. Manufactured by Boeing's Commercial Airplane unit in the United States, the original version of the 747 had two and a half times greater

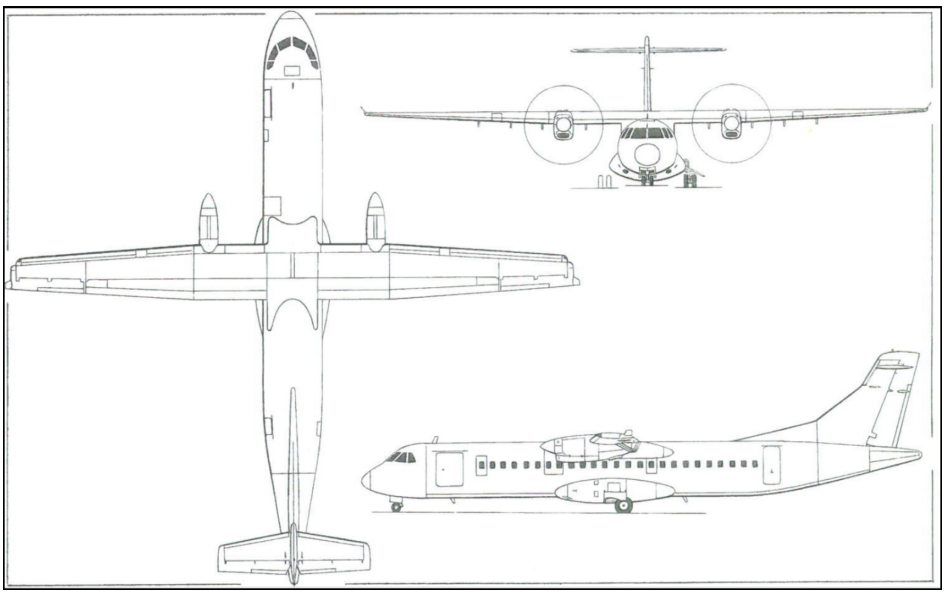

Figure 1.10 ATR-72 views – Jane's All the World's Aircraft 2004-2005[19]

capacity than the Boeing 707, one of the common large commercial aircraft of the 1960s. The four-engine 747 uses a double deck configuration for part of its length and it's available in passenger and other versions. Boeing designed the 747's hump-like upper deck to serve as a first class lounge or extra seating, and to allow the aircraft to be easily converted to a cargo carrier by removing seats and installing a front cargo door. The 747-100B model was developed from the 747-100SR, using its stronger airframe and landing gear design; the type had an increased fuel capacity of 182000 L, allowing for a 5000 nautical mile range with a typical 452 passengers payload, and an increased maximum take-off weight of 340000 kg was offered; unlike the original 747-100, the 747-100B was offered with Pratt & Whitney JT9D-7A, General Electric CF6-50, or Rolls-Royce RB211-524 turbofan engines.[5]

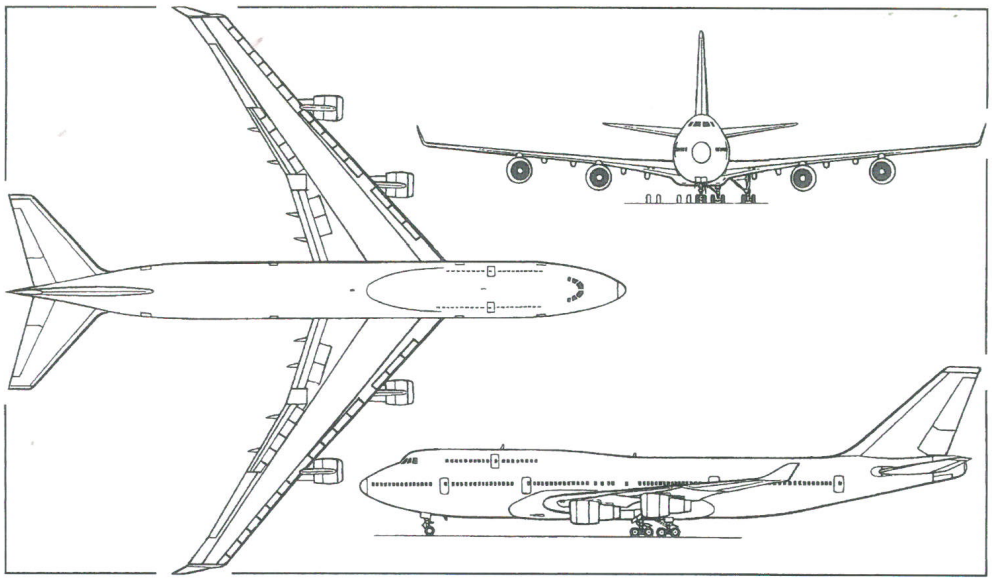

Figure 1.11 B747-100B views – Jane's All the World's Aircraft 2004-2005[19]

	ATR-72	B747-100B
Operating Conditions		
Altitude	6000,000 m	10000,000 m
Cruising Mach number	0.4300	0.8300
Weights		
Maximum Take-Off Mass	23063,579 kg	354991,506 kg
Operating Empty Mass	12935,579 kg	153131,986 kg
Maximum Fuel Mass	5000,000 kg	147409,520 kg
Maximum number of passengers	72	550
Fuselage		
Length	27,166 m	68,630 m
Diameter	2,756 m	6,793 m
Fineness Ratio	9.8566	10.1025
Wing		
Surface	61,000 m^2	511,000 m^2
$\mathcal{AR}$	12.0000	6.9000
Span	27,055 m	59,379 m
Taper Ratio	0.5450	0.2840
Root Chord	2,919 m	14,615 m
Mean Aerodynamic Chord	2,320 m	9,691 m
Sweep_{LE}	2,839 °	38,429 °
$\text{Sweep}_{\text{C/4}}$	1,400 °	35,500 °
$t/c_{\max}$	0.1675	0.1292
C_{D0}	0.03170	0.01820
Oswald Factor	0.7585	0.6277
Airfoil type	Conventional	Modern Supercritical
Horizontal Tail		
Surface	11,730 m^2	136,600 m^2
$\mathcal{AR}$	4.5550	3.5700
Span	7,310 m	22,083 m
Taper Ratio	0.5700	0.2650
Root Chord	2,044 m	9,780 m
Mean Aerodynamic Chord	1,645 m	6,884 m
Sweep_{LE}	3,441 °	38,225 °
$\text{Sweep}_{\text{C/4}}$	0,000 °	32,000 °
$t/c_{\max}$	0.1500	0.1500
Vertical Tail		
Surface	12,480 m^2	77,100 m^2

continued on next page

(*continued from previous page*)

	ATR-72	B747-100B
$\mathcal{AR}$	1.6600	1.3400
Span	4,552 m	10,164 m
Taper Ratio	0.3200	0.3300
Root Chord	4,154 m	11,407 m
Mean Aerodynamic Chord	2,932 m	8,229 m
Sweep_{LE}	40,549 °	53,991 °
$\text{Sweep}_{\text{C/4}}$	28,600 °	45,000 °
t/c_{max}	0.2227	0.2245
Power Plant		
Static Thrust (per engine)	7700,000 N	204000,000 N
Engines Number	2	4
Engines type	Turboprop	Turbofan
BRR	0.0000	5.0000
η_{p}	0.85	0.00

Table 1.1 ATR-72 and B747-100B input data

In order to define a default aircraft in a test class, and use it to check the functionalities of the application, it is necessary to follow some steps. First of all it's necessary to set up all default folders in order to make them achievable from any location inside the library; this is done with the static method `initWorkingDirectoryTree` of the `MyConfiguration` class located in `JPADConfigs` package. In this way, every time the user wants to point at a specific folder, like the input or output directory, it's necessary only to call the static method `getDir`, also from `MyConfiguration` class, which reads the folder name, from a dedicated Enumeration, and associate it with the related directory from a Map named `mapPaths`.

The second step is the creation of an `Aircraft` object choosing between the ones grouped into a dedicated Enumeration, named `AircraftEnum`. This can be done using the method `createDefaultAircraft` from `Aircraft` class. This method defines a new `Aircraft` object and invokes another method which creates all its components using default data. More in detail, inside the method `createDefaultAircraft` there is a calling to the constructor of the `Aircraft` class which initializes the objects of the classes that perform calculations. At this point all the components of the aircraft are created and ready to be used.

Afterwards it is necessary to set the operating conditions, such as the number of Mach of analysis or the altitude, by creating an object of the class `OperatingConditions`. Each default aircraft has a set of default conditions but the user could easly change them by using the *setters* methods of which the `OperatingConditions` class is supplied.

In order to manage all the aircraft related analysis it is necessary to define an object of the class `ACAnalysisManager`. Similarly to the aircraft, exists an analysis manager also for the wing that is

an object of the LSAerodynamicManager class; this results to be necessary since the aerodynamic analysis of the wing are not managed by the ACAnalysisManager in order to allow the analysis also of an isolated wing.

The next step is to define and assign the needed databases. In order to read these databases, and obtain their useful data, it is necessary to define an object of the database reading class (see appendix A.2) and associate it with the object of the analysis. Reading these data correctly is a crucial step; in fact, JPAD allows to work with an Aircraft object or only with an isolated LiftingSurface object. The aircraft is usually composed of a fuselage, lifting surfaces, nacelle and power plant; furthermore, the aircraft and the wing are associated with different classes of calculation like ACAnalysisManager or LSAerodynamicManager. Since it is necessary that these databases are also visible from these classes, as well as the fact that both in aircraft and in wing there is a lifting surface object, databases relative to aerodynamic analysis are associated to LSAerodynamicManager. In order to assign correctly this aerodynamic database and associate it to all analysis managers is necessary to practise the following order.

1. Define an Aircraft Object. This command associates to the Aircraft class an object that defines the aerodynamic. From the wing it is possible to obtain theWing, that is a LiftingSurface object
2. Define an ACAnalysisManager object. All the aircraft computations are managed by this class
3. Define an LSAerodynamicManager object. All the lifting surfaces computations are managed by this class
4. Associate database to LSAerodynamicManager
5. Eventually do analysis. It's important to highlight that, in order to perform only an aerodynamic analysis, the user has to define manually the position of the center of gravity.

The purpose of this structure is to have only a way to assign the aerodynamic databases to an aircraft. Inasmuch as the wing is always present, the chosen strategy is to assign the database to the aerodynamic manager of the wing; moreover, in order make the database usable also by the aircraft, it is assigned at the aerodynamic manager of the aircraft by calling the method doAnalysis. For the same reason, if the user has to get the database from the wing, the LSAerodynamicManager object created previously has to be set as the aerodynamic analysis of the wing object. In this way it's possible to call the database using equally the following codes:

- theWingObject.getAerodynamics.get_Database
- theAircraftObject.get_theAerodynamic.get_Database
- theLSManagerObject.get_Database
- theACManagerObject.get_Database

Although is the most important, the aerodynamic database is not the only one to be considered. To perform the high-ligt devices effects analysis (see chapter 4), as well as to carry out the integration of the equation of motion durign the take-off run (see chapter 5), also an high-lift devices database has to be read. Since the latter is strictly related to the wing, it has to be assigned to `LSAerodynamicManager` as well.

The following listing shows a summary of the steps described previously in order to help a potential user to easly implement a test class inside JPAD.

```
1  public static void main(String[] args) {
2    // ------------------------------------------------------------
3    // Define directory
4    MyConfiguration.initWorkingDirectoryTree();
5    // ------------------------------------------------------------
6    // Generate default Aircraft
7    Aircraft aircraft = Aircraft.createDefaultAircraft("B747-100B");
8    LiftingSurface theWing = aircraft.get_wing();
9    // ------------------------------------------------------------
10   // Default operating conditions
11   OperatingConditions theConditions = new OperatingConditions();
12   // ------------------------------------------------------------
13   // Define an ACAnalysisManager Object
14   ACAnalysisManager theAnalysis = new ACAnalysisManager(theConditions);
15   theAnalysis.updateGeometry(aircraft); // required to populate aircaft derived data
16   // ------------------------------------------------------------
17   // Define an LSAerodynamicsManager Object
18   LSAerodynamicsManager theLSAnalysis = new LSAerodynamicsManager (
19       theConditions,
20       theWing,
21       aircraft
22       );
23   // ------------------------------------------------------------
24   // Setup database(s)
25   theLSAnalysis.setDatabaseReaders(
26       new Pair(DatabaseReaderEnum.AERODYNAMIC,
27                        "Aerodynamic_Database_Ultimate.h5"),
28       new Pair(DatabaseReaderEnum.HIGHLIFT,
29                        "HighLiftDatabase.h5")
30       );
31   // ------------------------------------------------------------
32   // Do analysis
33   theAnalysis.doAnalysis(aircraft,
34       AnalysisTypeEnum.AERODYNAMIC);
35 }
```

Listing 1.3 Generation of default aircraft

In conclusion the following pages shows two flowcharts useful to better understand what has been described in this paragraph.

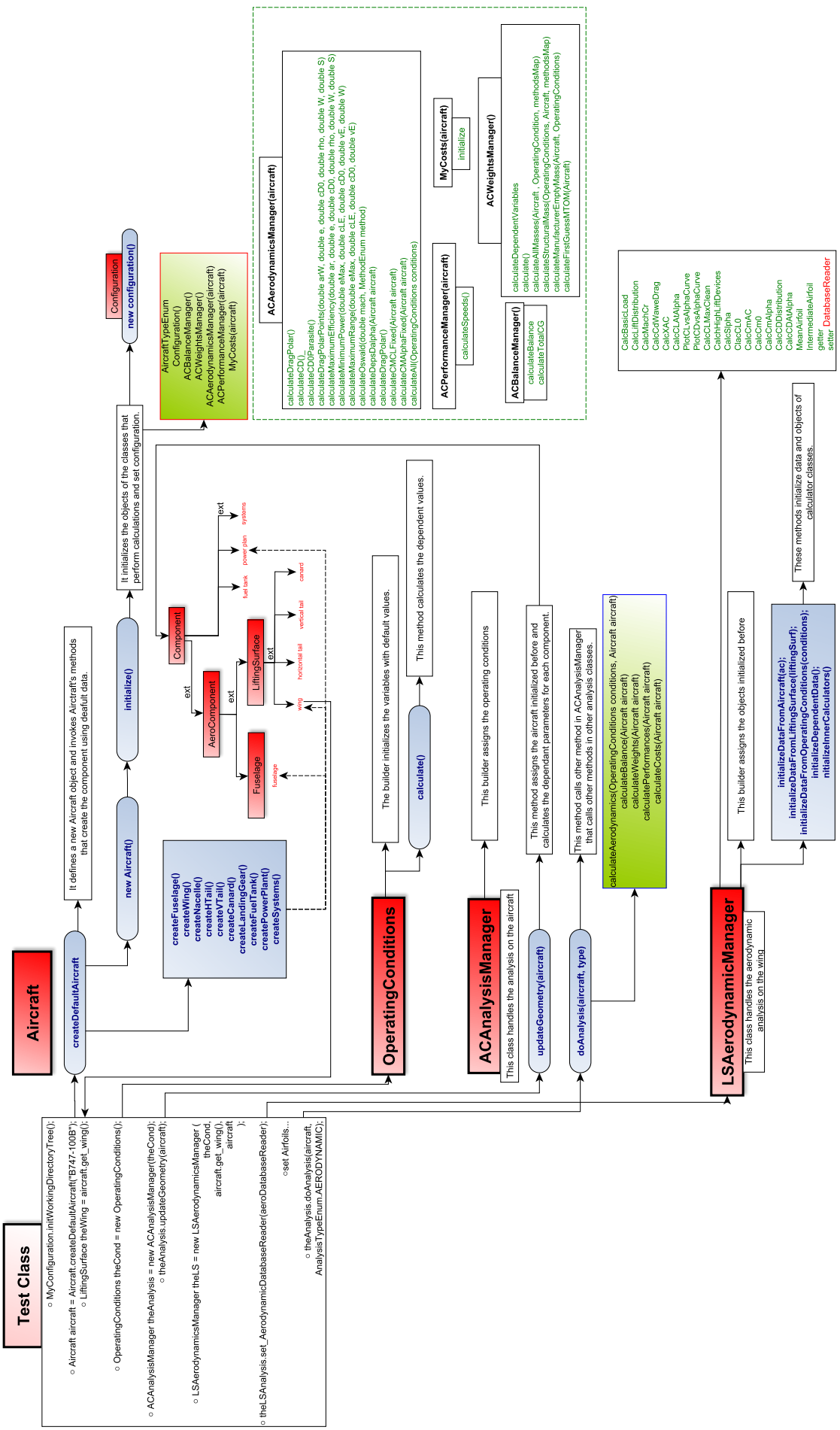

Figure 1.12 Flowchart of the creation of default Aircraft

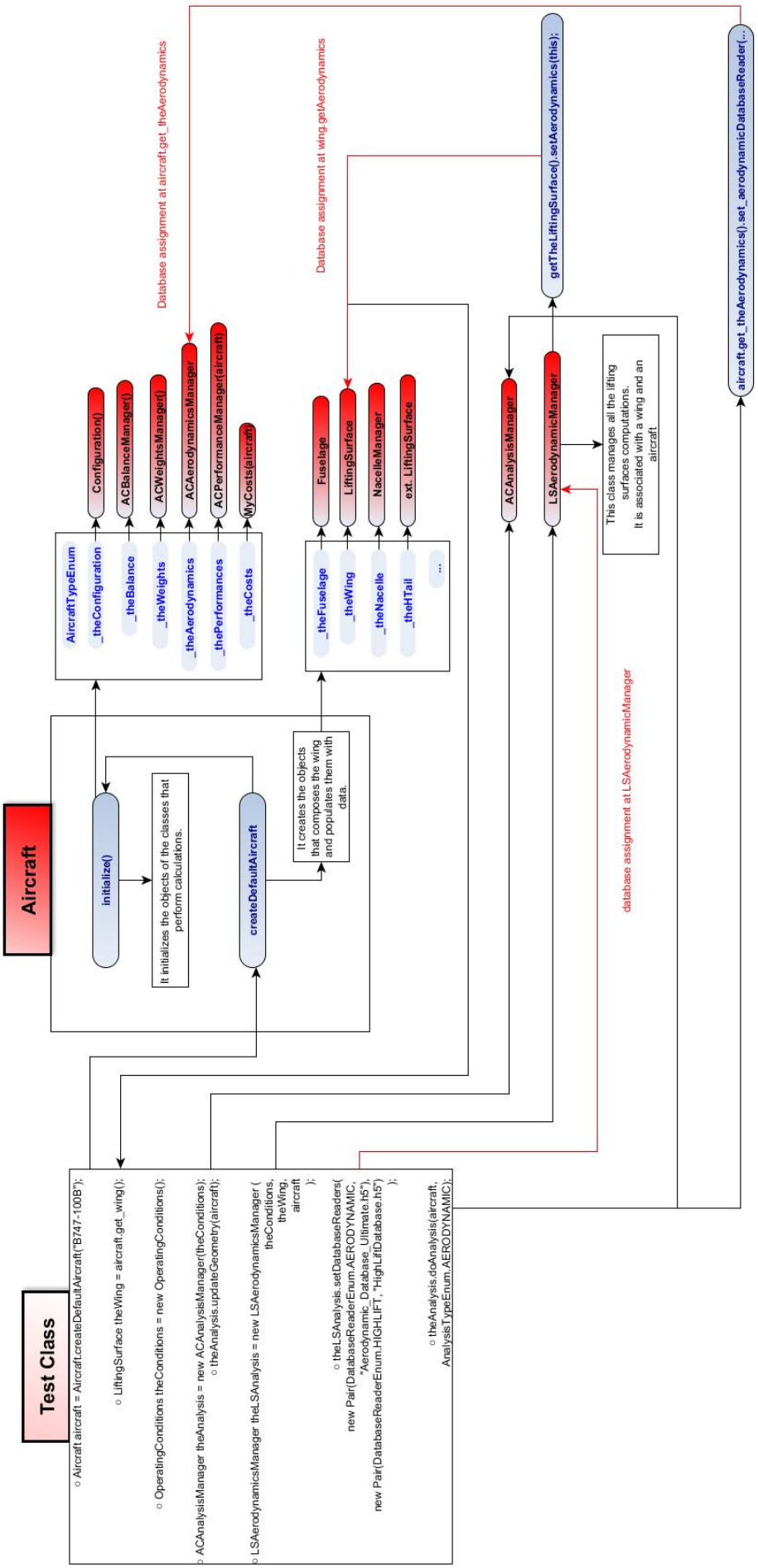

Figure 1.13 Flowchart of database assignment

1.5 The GUI

An extensive work has been done to set up an effective GUI to reduce the time the user has to spend to obtain relevant results. It should be noted that this GUI is only a prototype, although fully functional, of a possible future interface still in development.

The Java programming language greatly helped to build the GUI: several open source libraries (SWT, JFace) allowed to build a functional yet pleasant GUI which allows the user to easily change the aircraft parameters, to view a 3D model of the defined geometry, to launch a new analysis and view the corresponding results. The current GUI appearance is shown in figure 1.14. It is composed of several items:

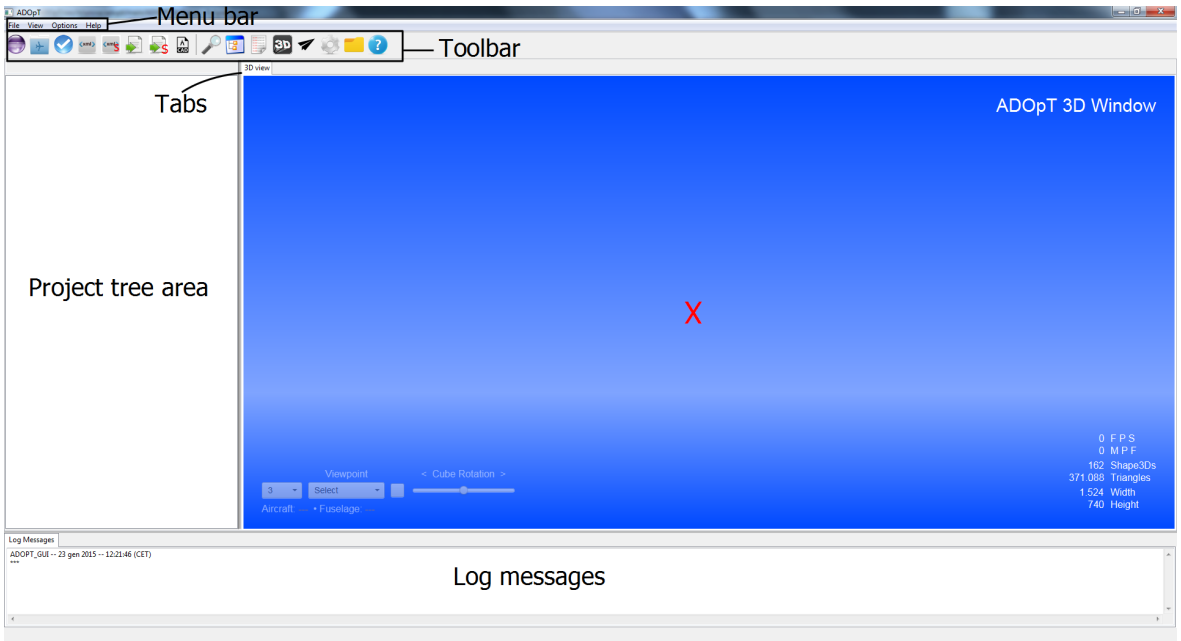

Figure 1.14 The GUI when the application is started

1. **Menu bar** (on top), which holds all the available actions divided in sub-categories
2. **Toolbar** (below the menu bar), which holds the most important actions needed to interact with the application. The toolbar, as the menu bar, is always visible to the user
3. **Project tree** (on the left), a key component of the GUI since it provides access to all the components of the aircraft and the analysis results any time during the execution of the application. The project tree appears once an aircraft is created and can be eventually hidden
4. **3D view**, which shows the CAD model of the aircraft selected by the user. The CAD model can be updated each time the user wants to check the changes made to the aircraft geometry
5. **Log message window**, placed at the bottom, that tells the user the status of pending operations. This window can be hidden
6. **Tab folder**, which contains all the windows opened using the project tree; these windows can be closed and reopened any time

1.5.1 Typical work session

In developing the application we focused on making the user's typical work session as simple as possible. Few basics steps are required for running an analysis:

1. Create a new aircraft. This can be done using the corresponding button () in the toolbar, which instantiates the default aircraft

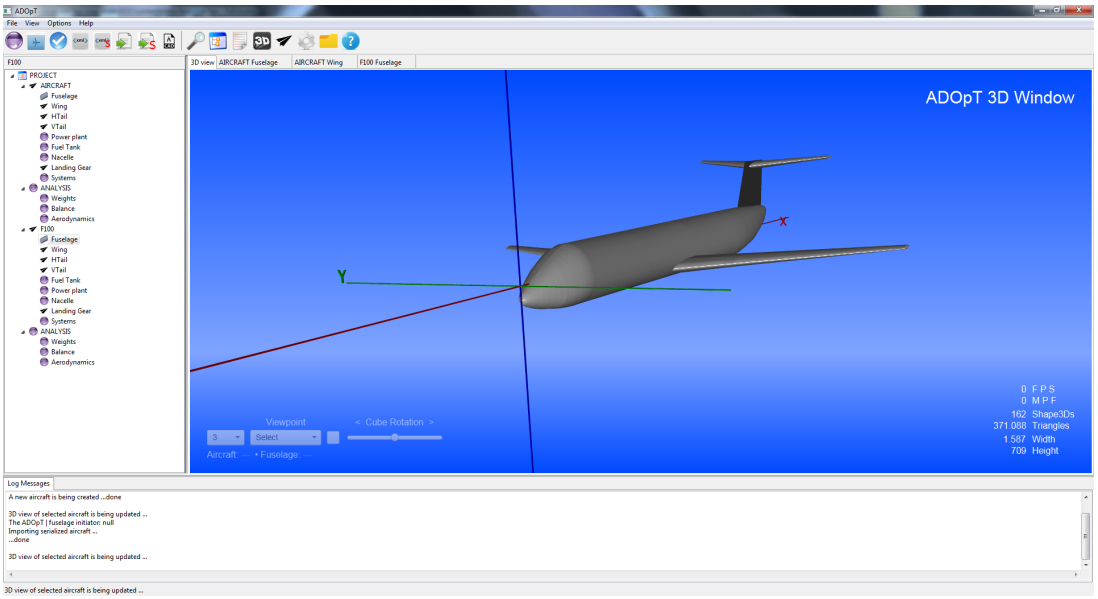

Figure 1.15 The GUI as it appears when an aircraft is created

2. Set the parameters that define the aircraft model using the corresponding window opened using the tree

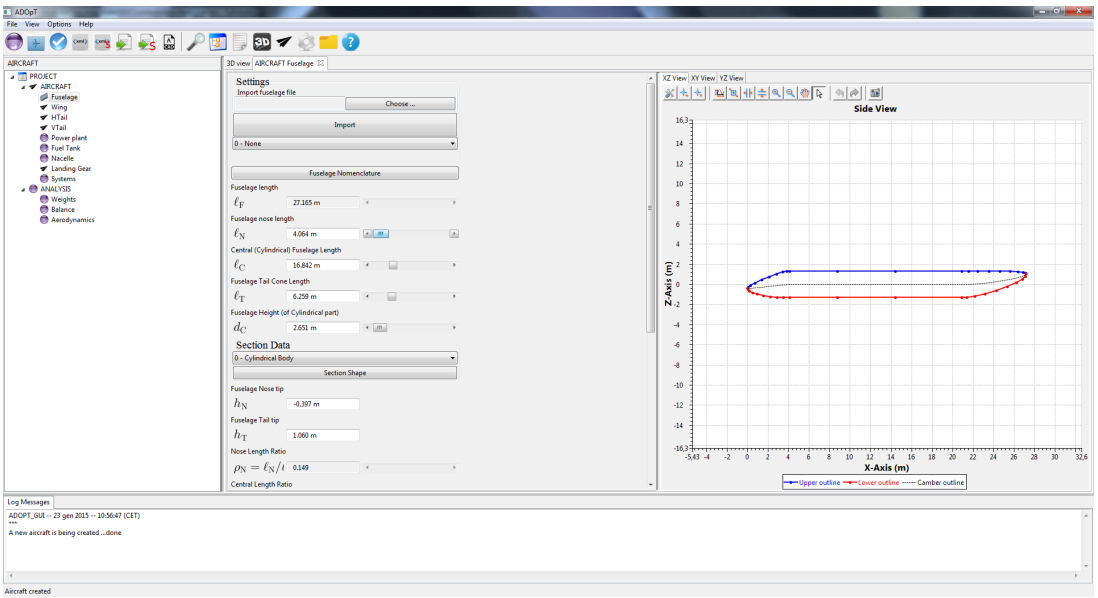

Figure 1.16 The window for changing the fuselage parameters

3. Explore the 3D model () of the aircraft and eventually change some parameters if there is some error

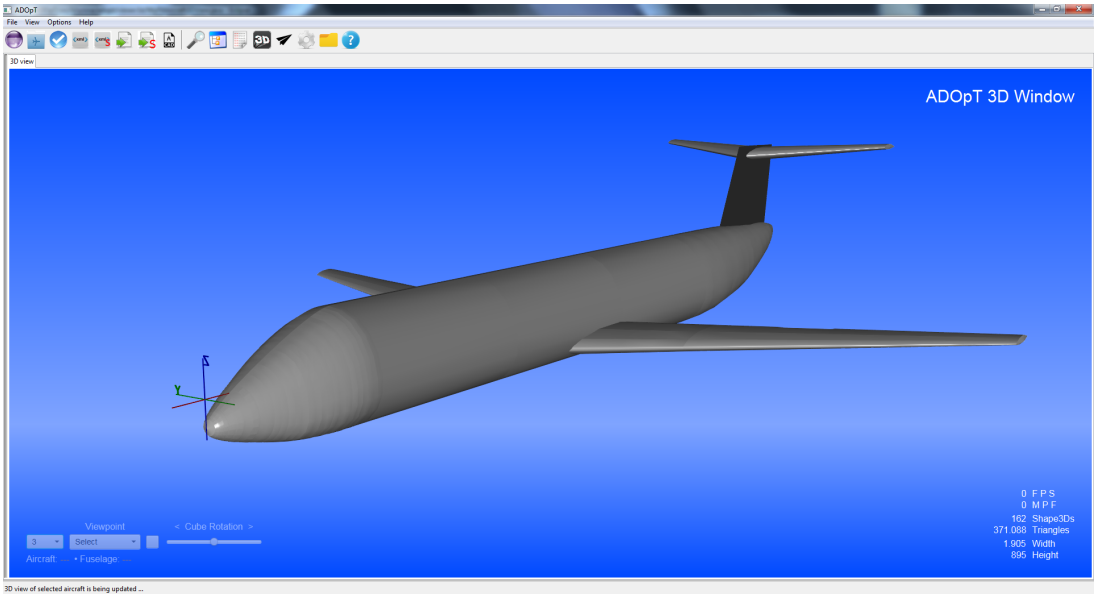

Figure 1.17 The aircraft 3D view (log window and project tree hidden)

4. Execute a complete analysis ( ) of the aircraft previously defined
5. Export the analysis results to an XML and/or an XLS file ( )
6. Eventually export the CAD model of the aircraft ( )
7. Save the current aircraft to an XML file ( )

At this point the user could simply change the current configuration until the analysis results are satisfactory. The application can however help the user in finding such a configuration, since it can hold multiple configurations simultaneously, analyse all of them and compare them side by side. To accomplish this task, once the first aircraft has been created, the user should:

1. Import the aicraft previously saved or create an entirely new aircraft
2. Change the aircraft parameters from the previous step and run a new analysis on it
3. Study the results and eventually change some of the parameters
4. Save every configuration and the corresponding results to file
5. Export both aircraft and the corresponding results to an XLS file

Tabs that are related to the same kind of component are distinguished by the aircraft name, put before the component's description, see figure 1.18.

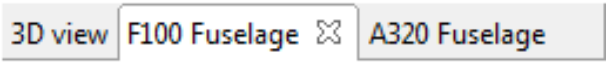

Figure 1.18 Same component tab belonging to different aircraft

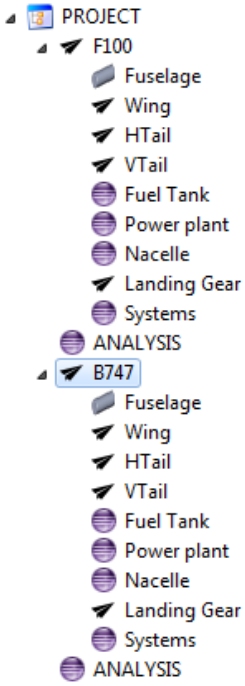

Figure 1.19 The project tree holding two different aircraft

There is no limit to the number of aircraft the application can handle; each aircraft is added to the project tree as shown in figure 1.19 providing access to the corresponding components and analysis.

1.5.2 CAD modelling

The application can also be used as a basic parametric CAD modeler. The capability to change the aircraft parameters using the corresponding controls in the GUI, coupled with the 3D view, allows the user to change each component shape and dimension, view the updated CAD model and eventually export it to file once some satisfactory results have been obtained. The CAD model, once saved to a STEP or BRep file, can be imported in CAD suites like FreeCAD, MeshLab, SketchUp, Blender, SolidWorks, or CATIA. The file can also be used to further study the aircraft in a CAE software, such as CD-adapco STAR-CCM+ or ANSYS suite.

1.6 Optimization process

It has always been the engineer's dream to have all aspects of analysis done in a relatively short time period so that many different configurations can be examined and the best suitable product can be delivered on time. Although this may still be a dream, actual design turn-around time has become shorter due to the use of mathematical optimization techniques which have been introduced into the design process.[28]

According to [18], the optimisation of the configuration of the aircraft within the constraints imposed by the specification is an essential feature of the project definition process. Optimisation implies that the proposed design concept not only meets the specification, but does so when a target criterion has been imposed, usually the minimisation of mass or some aspect of

costs. The basis of optimisation is a comparison of different design concepts and configuration variations within a given concept to determine the one which best meets the specification. Broadly the process is undertaken at two levels.

- **Concept/configuration studies** At this level alternative concepts and configurations are investigated to establish the one which would seem best suited to meet the requirements. For example a military combat requirement might possibly be met by aircraft of conventional tail, foreplane or tailless layout. Mostly the configuration of transport aircraft is well established. This phase of the study may be included at the feasibility level.

- **Parametric studies within a given configuration** At this level the dominant parameters are varied to ascertain the best combination of them. These parameters include such items as wing geometry determined by aspect ratio, sweep, taper and thickness as well as variations in fuselage layout, powerplant installation and so on. The benefits of selecting near optimum values of certain parameters are "traded-off" against the implied penalties imposed upon other parameters. The parametric studies may be commenced during feasibility investigations and certainly form a major part of the project definition phase. The two fundamental design characteristics which drive the parametric studies are:

  - Wing loading, that is wing area as a function of take-off mass or weight
  - Thrust or power loading, defined as the ratio of the basic powerplant to the aircraft mass or weight

It is, precisely, the latter type of optimization which will be the main feature to implement inside JPAD in the near future. As all analysis modules inside the `JPADCore` package will be completed and tested, the final purpose of the code will be to allow users to define a certain numbers of macroscopical geometrical parameters, along with a given objective function, and to receive as output the best set of the previous parameters which suits the wanted objective.

Chapter 2

Payload-Range

All men dream: but not equally. Those who dream by night in the dusty recesses of their minds wake in the day to find that it was vanity: but the dreamers of the day are dangerous men, for they may act their dreams with open eyes, to make it possible. This I did.

– T.E Lawrence "Lawrence of Arabia"

The Payload-Range diagram is a very important resource in aircrafts design because it allows to illustrate an aircraft operational limits by showing the trade-off relationship between the payload that can be carried and the range that can be flown; moreover, in synergy with the Direct Operative Cost (D.O.C.) diagram, it's very useful to the future aircraft customers to have an idea of expected profits derived by their investment. Payload-Range is also a key feature of the design requirements so it's possible to state that its relevance is felt through all the conceptual and preliminary design phases.

2.1 Theoretical background

To better understand the theoretical background behind this diagram, the first step is to focus the attention upon the link between the range and the fuel consumption, which is influenced by aircraft weight through Breguet formulas. It's possible to imagine, at first, an aircraft in which payload and fuel weights can be managed at will, so that, in this completely flexible aircraft, the more the required range is, the less is the payload in order to carry more fuel at a specified maximum take-off weight; this leads to the diagram in figure 3.1. Because the payload is carried in the fuselage while the fuel is carried in wing tanks, the more the payload, the more the weight in the fuselage; this means that wings are incrementally more bending loaded so that a heavier structure is required. However, in this way the operating empty weight grows up limiting the payload the aircraft can carry, at that maximum take-off and fuel weights, for that range; the consequence of this is that, in particular for long range missions, the payload is too low, providing low profits. In order to avoid this situation it's possible to decide, preventively, the maximum payload weight the aircraft can carry in fuselage so that the maximum bending load for wings and, in consequence their structural weight, is set; this particular weight is

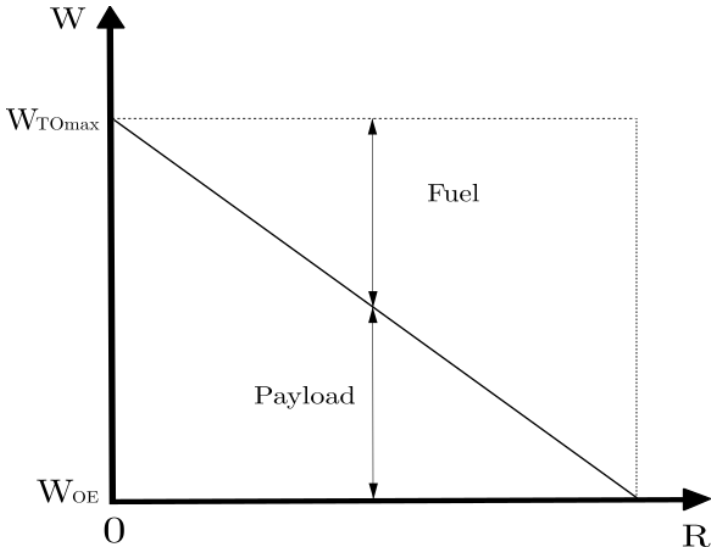

Figure 2.1 Payload-Range for a completely flexible aircraft

named Maximum Zero Fuel Weight (MZFW). As a result of this, the previous diagram changes in the one reported in figure 3.2 which starts with the maximum zero fuel weight and keep it constant until the maximum take-off weight is reached. From this point on, the far the aircraft have to go, the more the fuel it needs, but, since the maximum take-off weight is set, the only way to increase fuel weight is to reduce the payload; this condition is represented by the second segment of the diagram and, along this, the aircraft weight is always equal to the maximum take-off weight. With further increase of the required range for the aircraft, the fuel tanks maximum capacity will be reached prevent to store more fuel; the only solution at this problem is to put the additional fuel into the fuselage, reducing, consequently, the payload weight in order to respect the maximum zero fuel weight limitation. Along this last segment the payload weight decreases more rapidly with increasing range reaching, ideally, the value of zero at which the flight mission is useless; moreover the aircraft weight is not equal to the maximum take-off weight anymore but to a lower one.

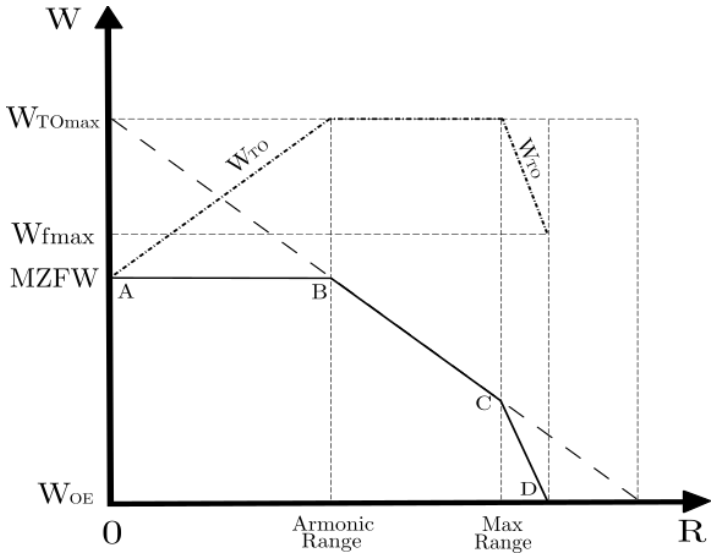

Figure 2.2 Payload-Range with MZFW limitation

Now that the theory behind the Payload-Range is explained, it's interesting to see how to build up the diagram. For this purpose at least the preliminary weight estimation has to be done in order to know the value of the following weights.

- W_{TO}
- W_{OE}
- $W_{Payload}$

Furthermore four fundamental couples of payload and range values have to be defined:

- Point A, in which the payload is the maximum allowed and the range is zero.
- Point B, in which the maximum range, with maximum take-off weight and maximum payload, is reached. This range value is called Armoinc Range.
- Point C, in which the fuel tanks full capacity is reached with a payload lower than the maximum one and still the maximum take-off weight.
- Point D, in which the take-off weight decreases because of the reduction of payload to zero in order to store more fuel in the fuselage.

Since point A is very simple to obtain, it's more interesting to analyze how to calculate points B, C and D coordinates. For point B it's necessary to focus the attention upon Breguet formulas in order to calculate the maximum range the aircraft can fly at maximum take-off weight with maximum payload; these are different for jet or propeller aircraft and can be written as follow, with the range R in km, Specific Fuel Consumption (SFC) in $\frac{\text{lb}}{\text{hp·h}}$ and Jet Specific Fuel Consumption (SFCJ) in $\frac{\text{lb}}{\text{lb·h}}$.

$$R = 603.5 \cdot \left(\frac{\eta_p}{SFC}\right)_{\text{cruise}} \cdot \left(\frac{L}{D}\right)_{\text{cruise}} \cdot \ln\left(\frac{W_i}{W_f}\right) \quad \text{if propeller engine driven} \tag{2.1a}$$

$$R = \left(\frac{V}{SFCJ}\right)_{\text{cruise}} \cdot \left(\frac{L}{D}\right)_{\text{cruise}} \cdot \ln\left(\frac{W_i}{W_f}\right) \quad \text{if jet engine driven} \tag{2.1b}$$

Knowing the specific fuel consumption, the aerodynamic and propeller efficiency, or speed for jet aircraft, in cruise condition, the only unknown left for calculate the range is the weight ratio. In order to calculate this it's necessary to start by evaluating the amount of fuel the airplane can take on board, at maximum take-off weight and maximum payload, with the following formula.

$$W_{TO,\text{max}} = W_{OE} + W_{\text{payoad,max}} + W_{\text{fuel}} \tag{2.2}$$

Once the used fuel weight is known, it's possible to use the fuel fraction method from the weight estimation design phase in order to find the total weight ratio across the entire aircraft mission.

$$W_{\text{fuel}} = \left(1 - M_{ff}\right) \cdot W_{TO} \tag{2.3}$$

M_{ff} is the fuel fraction of the entire mission profile given by the following expression.

$$M_{ff} = \frac{W_1}{W_{TO}} \cdot \frac{W_2}{W_1} \cdot \frac{W_3}{W_2} \cdot \frac{W_4}{W_3} \cdot \frac{W_5}{W_4} \cdot \frac{W_6}{W_5} \cdot \frac{W_7}{W_6} \cdot \frac{W_8}{W_7} \cdot \frac{W_9}{W_8} \cdot \frac{W_{10}}{W_9} \cdot \frac{W_{\text{final}}}{W_{10}} \tag{2.4}$$

From this point on, it's necessary to have a table, like the one of table 3.1, in which all weight ratios can be found statistically except for cruise, alternate cruise and loiter phases; these three unknown ratios are those that build up the Breguet weight ratio required by the range formula.

Airplane type	Start, Warm-up	Taxi	Take-off	Brief descent	Brief climb	Landing, Taxi and Shutdown
Homebuilts	0,998	0,998	0,998	0,995	0,995	0,995
Single Engine	0,995	0,997	0,998	0,992	0,993	0,993
Twin Engine	0,992	0,996	0,996	0,990	0,992	0,992
Agricultural	0,996	0,995	0,996	0,998	0,999	0,998
Business Jets	0,990	0,995	0,995	0,980	0,990	0,992
Regional TBP's	0,990	0,995	0,995	0,985	0,985	0,995
Transport Jets	0,990	0,990	0,995	0,980	0,990	0,992
Mil. Trainers	0,990	0,990	0,990	0,980	0,990	0,995
Fighters	0,990	0,990	0,990	0,960-0,900	0,990	0,995
Mil.Patrol,Bmb and Trspt	0,990	0,990	0,995	0,980	0,990	0,992
Flying boats,Amph. and Floats	0,992	0,990	0,996	0,985	0,990	0,990
Supersonic Cruise	0,990	0,995	0,995	0,920-0,870	0,985	0,992

Table 2.1 Suggested fuel fractions

For the evaluation of point C, similar steps have to be followed with the difference that, in this case, the fuel is the maximum that the wing tanks can store and the payload is the unknown of the (2.5).

$$W_{TO,\max} = W_{OE} + W_{\text{payload}} + W_{\text{fuel,max}} \tag{2.5}$$

From this equation it's possible to calculate point C ordinate, while for the range abscissa the same procedure used for point B range has to be followed with the difference that now it's necessary to use the fuel weight relative to the maximum fuel tank capacity which is known from the wing fuel tank design. Finally for point D, the payload is set at zero so that is

possible to evaluate the WTO, relative to maximum fuel weight with no passengers, from the following equation.

$$W_{TO} = W_{OE} + W_{\text{fuel,max}} \tag{2.6}$$

This new take-off weight generates a lower M_{ff}, from fuel fraction equation, which leads to a bigger weight ratio to be used in Breguet formulas with the result of a bigger range.

2.2 Java class architecture

In this paragraph the implementation in the JPAD software of the Payload-Range diagram, through the `PayloadRangeCalc` Java class, is presented. The idea has been to create a dedicated Java class which is demanded of the calculation of the four couple of range and payload values presented before; moreover it has to confront the user chosen Mach number condition with the best range one and to parametrize the analysis at different maximum take-off weight in order to help users making design decisions about different version of the same aircraft. The class core consists of the following three principal methods which have to evaluate points B, C and D coordinates using the procedure explained before.

- `calcRangeAtMaxPayload`
- `calcRangeAtMaxFuel`
- `calcRangeAtZeroPayload`

Each of these requires in input the table 3.2 data, a database which collects all data from table 3.1, named `FuelFractionDatabaseReader`, and an engine database, for turboprop or turbofan, in which all specific fuel consumption values, at given Mach number and altitude, are stored. From this point on, the evaluation of (2.2), (2.5) and (2.6), for each method, and of (2.3), for all of them, is performed in order to set up the following calculations and to obtain the unknown value of the payload in point C case; after that the fuel fractions database is read in order to obtain all known data necessary to calculate the required weight ratio to be used in (2.1a) or (2.1b) depending on the aircraft engine type. It's important to highlight that the database reading process is made up so that it can recognize all different kind of aircraft from table 3.1 and read the related line values.

Since each method has to compare the chosen Mach number condition with the best range one, the boolean variable presented before is used in order to distinguish between different application cases; in this way, when the user specify the aircraft engine type and the boolean variable, the method reads the specific fuel consumption value, at given Mach number and altitude, from the related engine database and performs the calculation of the lift and drag wing coefficients which are then used to obtain the aerodynamic efficiency value. At this point is possible to calculate the range that represents point B, C or D abscissa. It should be noted that, in case of a true value of the boolean variable, the evaluation of the lift and drag wing coefficients, as well as the best range Mach number that replaces the user chosen one, is performed by calculating the parabolic drag polar characteristic points and choosing those that maximizes the range (point E for the turboprop and point A for the turbofan); whereas for a

`maxTakeOffMass`	Maximum take-off mass
`sweepHalfChordEquivalent`	Equivalent wing sweep angle at half chord
`surface`	Wing surface
`cd0`	Wing c_{D0}
`oswald`	Wing oswald factor
`cl`	The current lift coefficient in cruise configuration
`ar`	Wing aspect ratio
`tcMax`	Mean maximum thickness of the wing
`byPassRatio`	Engine by-pass ratio (when needed)
`eta`	Propeller efficiency (when needed)
`altitude`	Cruise altitude
`currentMach`	The actual Mach number during cruise
`isBestRange`	A boolean variable that is true if the evaluation of the best range condition is performed, otherwise it's false

Table 2.2 Input data

false value of the boolean variable, the lift coefficient is calculated by using the current flight condition angle of attack into the linear part of the wing lift curve, while the drag coefficient is evaluated from the aircraft total drag polar taking also into account potential wave drag sources.

Now that all points coordinates are known, the two following methods are demanded to build up abscissas and ordinates values arrays to be used in plotting the diagram.

- `createRangeArray`
- `createPayloadArray`

Range Array	Payload Array
0,0	Maximum payload, in number of passengers
`calcRangeAtMaxPayload` output	Maximum payload, in number of passengers
`calcRangeAtMaxFuel` output	Payload calculated in `calcRangeAtMaxFuel`
`calcRangeAtZeroPayload` output	0,0

Table 2.3 Payload and range array components

They use the Java List interface to generate an ordered collection of values which are then populated with the following values.

Finally the two arrays are used as input, together with another one from the best range condition analysis, for the last method, named `createPayloadRangeCharts_Mach`, that has to plot

the diagram; this uses the `JFreeChart` Java library[15] to generate a Portable Network Graphics (PNG) output image into the output folder of the software and is also able to create a TikZ version of the diagram to be used in LaTeX. As said before, this class finds another purpose in parameterizing the diagram in different maximum take-off weight conditions. To do that, the three core methods are used again inside a new one, named `createPayloadRangeMatrices`, which implements a recursive calculation of the three points coordinates at different maximum take-off mass conditions generated by decreasing the original mass by 5% until it reaches a total decrease of 20%. This new method requires the same input data reported in table 3.2 except for the maximum take-off mass which is generated inside the method itself. The output matrices are then used in a plotting method, equivalent to previous one but named `createPayloadRangeCharts_MaxTakeOffMass`, which generates the PNG and TikZ output images by receiving as input two matrices and not two arrays as before.

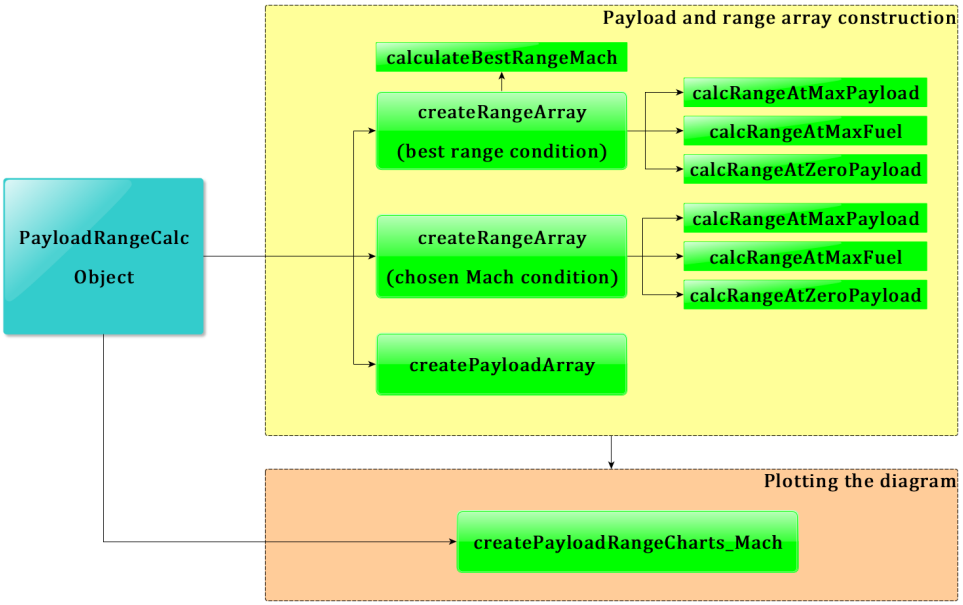

Figure 2.3 Payload-Range java class flowchart for best range and chosen Mach conditions comparison

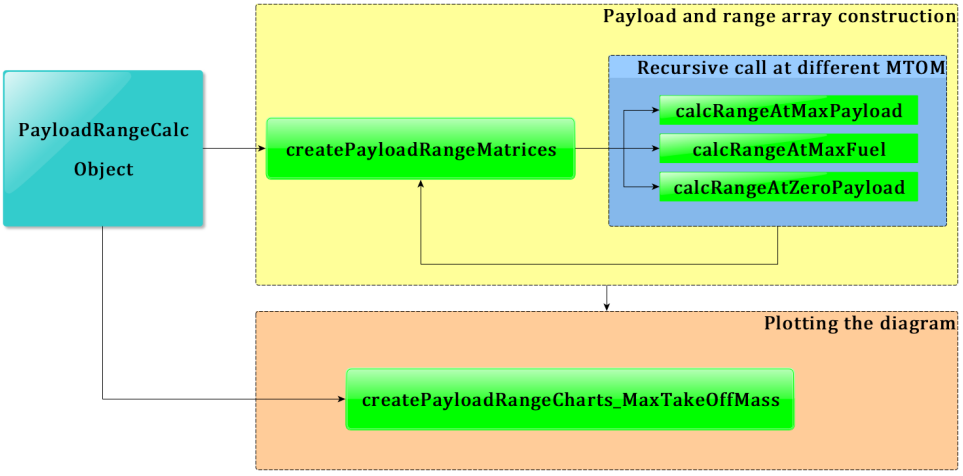

Figure 2.4 Payload-Range java class flowchart for maximum take-off mass parameterization

2.3 Case study: ATR-72 and B747-100B

With the purpose of validating calculations presented before, two case studies have been taken into account; the first one is on the ATR-72 and the second one on the B747-100B.

The two presented tests share the same architecture so that a general guideline can be followed; this strategy has been designed with the purpose of making a general procedure in order to simplify user's work. Thus, for more clarity, the listings of test is reported, as an exemplificative practice, only in the case of the ATR-72.

The preliminary steps required to build up the test class are the same that have been discussed in the paragraph 1.4. However, since the class `PayloadRangeCalc` requires that data from table 3.1 are available, another database has to be read. This database is named `FuelFraction.h5` and is read by the class `FuelFractionDatabaseReader` created following the guideline of appendix A.2. Furhermore, this database is strictly connected with the fuel tank so that it has been decided to link the object of the class `FuelFractionDatabaseReader` to this aircraft component as follows.

```
1    aircraft.get_theFuelTank().setFuelFractionDatabase(fuelFractionReader);
```

Listing 2.1 Attachment of the `FuelFractionDatabaseReader` to the fuel tank component of the aircraft

At this point all data required by the `PayloadRangeCalc` class are now ready to be used and it's possible to create an object of this class accessing, in this way, to all its methods, which have been explained in the previous paragraph.

```
1    // Creating the Calculator Object
2    PayloadRangeCalc test = new PayloadRangeCalc(
3                theCondition,
4                aircraft,
5                AirplaneType.TURBOPROP_REGIONAL
6                );
```

Listing 2.2 Excerpt of the ATR-72 Payload-Range test - Payload-Range class istance creation

The first check that is implemented has the purpose of inspect wether or not the chosen cruise Mach number is bigger than the crest critical one; in this case a warning message is launched in order to inform the user of the situation. The method demanded of this is `checkCriticalMach` which accepts in input a given Mach number, compares it with the one calculated with Kroo method [7], starting from the cruise lift coefficient, and return a boolean variable that is true only if the chosen Mach number is bigger than the crest critical one indeed.

```
1    // ----------CRITICAL MACH NUMBER CHECK-------------
2    boolean check = test.checkCriticalMach(theCondition.get_machCurrent() );
3    if (check)
4      System.out.println("\n\n-------------------------"
5      +"\nCurrent Mach lower then critical Mach number"
6      +"\nCurrent Mach = " + theCondition.get_machCurrent()
```

```
7         +"\nCritical Mach = " + test.getCriticalMach()
8         +"\n\n\t CHECK PASSED -> PROCEDING TO CALCULATION "
9         +"\n\n"
10        +"-----------------------------------------------");
11      else{
12        System.err.println("\n\n-------------------------"
13        +"\nCurrent Mach bigger then critical Mach number"
14        +"\nCurrent Mach = " + theCondition.get_machCurrent()
15        +"\nCritical Mach = " + test.getCriticalMach()
16        +"\n\n\t CHECK NOT PASSED -> WARNING!!! "
17        +"\n\n"
18        +"-------------------------------------");
19      }
```

Listing 2.3 Excerpt of the ATR-72 Payload-Range test - critical Mach number check

```
1  Current Mach is lower then critical Mach number.
2  Current Mach = 0.43
3  Critical Mach = 0.6659791543529567
4
5     CHECK PASSED --> PROCEDING TO CALCULATION
```

Listing 2.4 Excerpt of the ATR-72 Payload-Range test results - critical Mach number check

```
1  Current Mach is lower then critical Mach number.
2  Current Mach = 0.84
3  Critical Mach = 0.8490814607347361
4
5     CHECK PASSED --> PROCEDING TO CALCULATION
```

Listing 2.5 Excerpt of the B747-100B Payload-Range test results - critical Mach number check

At this point, if the user wants to compare the chosen Mach number condition with the best range one, all that it's needed is to invoke the `createRangeArray` method, for both the operative conditions, and the `createPayloadArray` one; after that the diagram is generated from the method `createPayloadRangeCharts_Mach`.

```
1       // ----------------BEST RANGE CASE-------------------
2       System.out.println();
3       System.out.println("-------BEST RANGE CASE--------");
4       List<Amount<Length>> vRange_BR = test
5               .createRangeArray(
6                       test.getMaxTakeOffMass(),
7                       test.getSweepHalfChordEquivalent(),
8                       test.getSurface(),
9                       test.getCd0(),
10                      test.getOswald(),
11                      aircraft.get_theAerodynamics().getcLE(),
12                      test.getAr(),
13                      test.getTcMax(),
```

```
14                      test.setByPassRatio(0.0),
15                      test.getEta(),
16                      test.getAltitude(),
17                      test.calculateBestRangeMach(
18                              EngineTypeEnum.TURBOPROP,
19                              test.getSurface(),
20                              test.getAr(),
21                              test.getOswald(),
22                              test.getCd0(),
23                              test.getAltitude()),
24                      true);
25      // -------------USER CURRENT MACH-------------------
26      System.out.println();
27      System.out.println("------CURRENT MACH CASE-------");
28      List<Amount<Length>> vRange_CM = test
29              .createRangeArray(
30                      test.getMaxTakeOffMass(),
31                      test.getSweepHalfChordEquivalent(),
32                      test.getSurface(),
33                      test.getCd0(),
34                      test.getOswald(),
35                      test.getCl(),
36                      test.getAr(),
37                      test.getTcMax(),
38                      test.setByPassRatio(0.0),
39                      test.getEta(),
40                      test.getAltitude(),
41                      test.getCurrentMach(),
42                      false);
43      // -------------------PLOTTING----------------------
44      // Mach parameterization:
45      List<Double> vPayload = test.createPayloadArray();
46      test.createPayloadRangeCharts_Mach(
47              vRange_BR,
48              vRange_CM,
49              vPayload,
50              test.calculateBestRangeMach(
51                      EngineTypeEnum.TURBOPROP,
52                      test.getSurface(),
53                      test.getAr(),
54                      test.getOswald(),
55                      test.getCd0(),
56                      test.getAltitude()),
57              test.getCurrentMach());
58    }
59  }
```

Listing 2.6 Excerpt of the ATR-72 Payload-Range test - Payload-Range diagram evaluation and plot

Otherwise, if it's the maximum take-off mass parameterization the wanted analysis, the required call is for `createPayloadRangeMatrices` method followed by the plotting method `createPayloadRangeCharts_MaxTakeOffMass`.

```
    // -----------------MTOM PARAMETERIZATION----------
    test.createPayloadRangeMatrices(
            test.getSweepHalfChordEquivalent(),
            test.getSurface(),
            test.getCd0(),
            test.getOswald(),
            test.getCl(),
            test.getAr(),
            test.getTcMax(),
            test.setByPassRatio(0.0),
            test.getEta(),
            test.getAltitude(),
            test.getCurrentMach(),
            false
            );
    // ------------------PLOTTING----------------------
    // MTOM parameterization:
    test.createPayloadRangeCharts_MaxTakeOffMass(
            test.getRangeMatrix(),
            test.getPayloadMatrix()
            );
  }
}
```

Listing 2.7 Excerpt of the ATR-72 Payload-Range test - maximum take-off mass parameterization

In conclusion, following pages shows a summary of input data used for each analyzed aircraft, together with their related results in terms of Payload-Range diagrams for both the chosen Mach number and best range comparison and max take-off mass parameterization.

Variables	Values
Range (point A)	0 nmi
Payload (point A)	72 passengers
Range (point B)	502.587 nmi
Payload (point B)	72 passengers
Range (point C)	1246.827 nmi
Payload (point C)	52 passengers
Range (point D)	1831.171 nmi
Payload (point D)	0 passengers
C_L	0.421
C_D	0.0388
Efficiency	10.853
SFC	0.424

Variables	Values
Range (point A)	0 nmi
Payload (point A)	550 passengers
Range (point B)	4714.449 nmi
Payload (point B)	550 passengers
Range (point C)	4715.799 nmi
Payload (point C)	550 passengers
Range (point D)	6093.423 nmi
Payload (point D)	0 passengers
C_L	0.385
C_D	0.0296
Efficiency	13.00518
SFC	0.626

Table 2.4 Review of ATR-72 (left) and B747-100B (right) results at chosen Mach number conditions

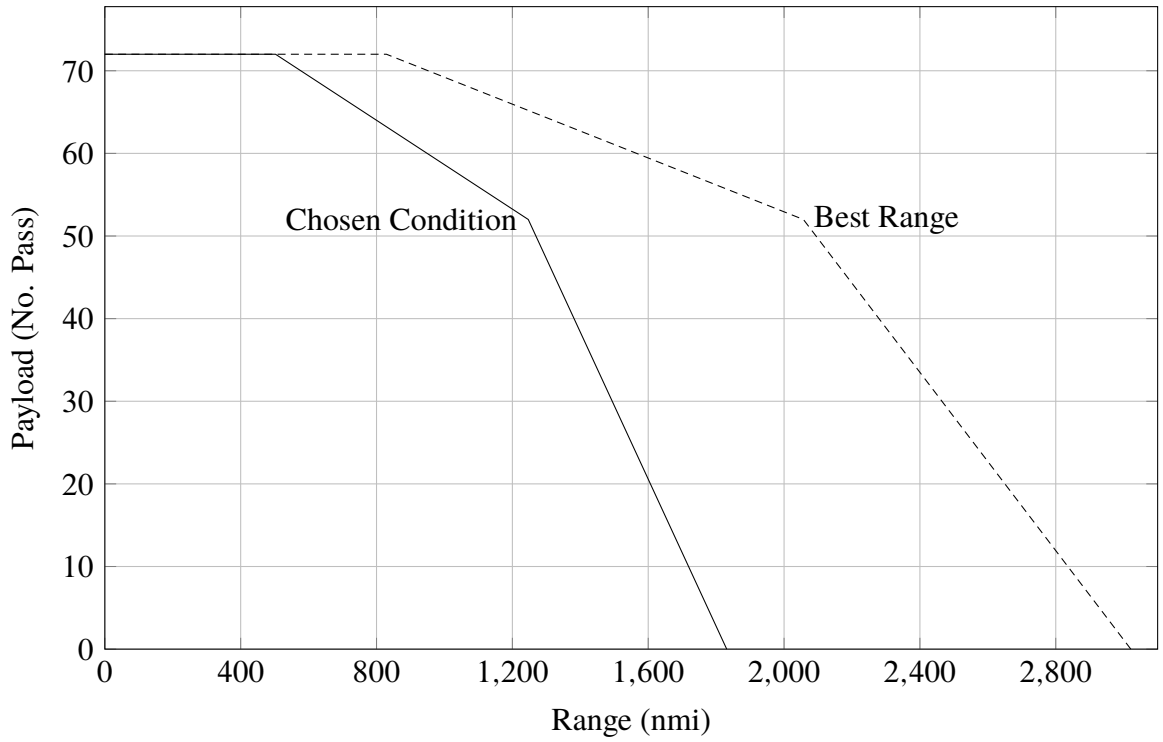

Figure 2.5 ATR-72 Payload-Range - Chosen Mach number and best range comparison

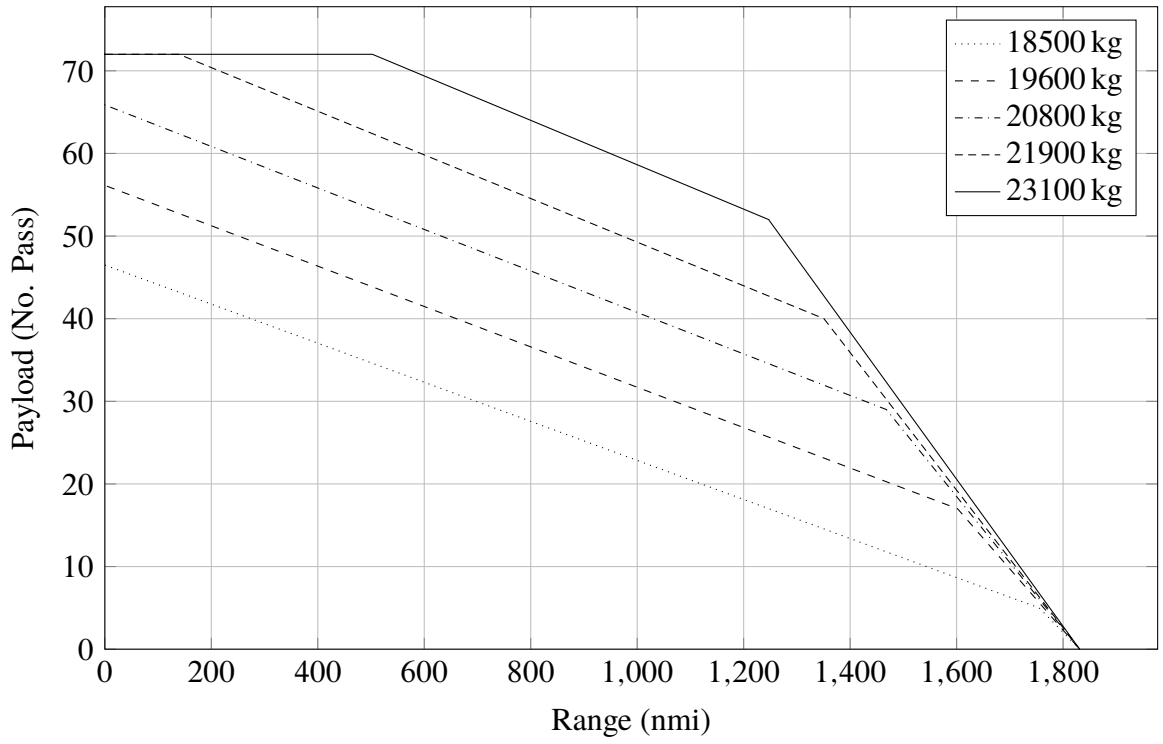

Figure 2.6 ATR-72 Payload-Range - Maximum take-off mass parameterization

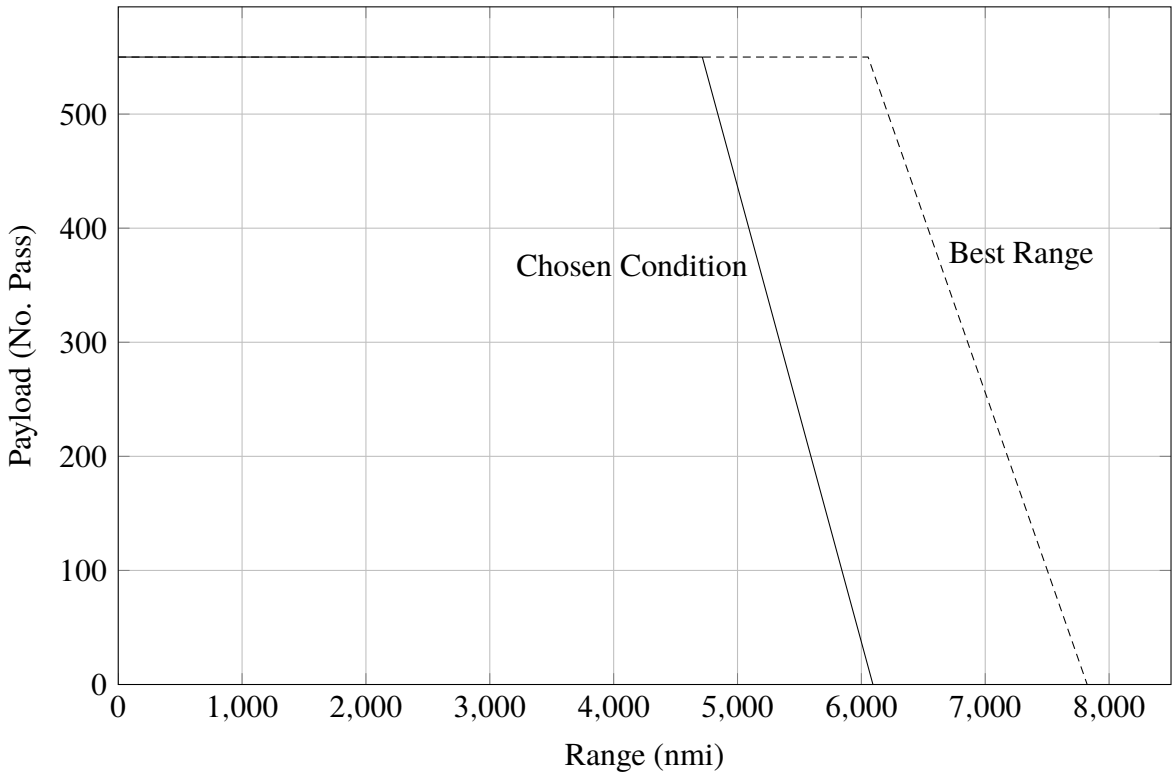

Figure 2.7 B747-100B Payload-Range - Chosen Mach number and best range comparison

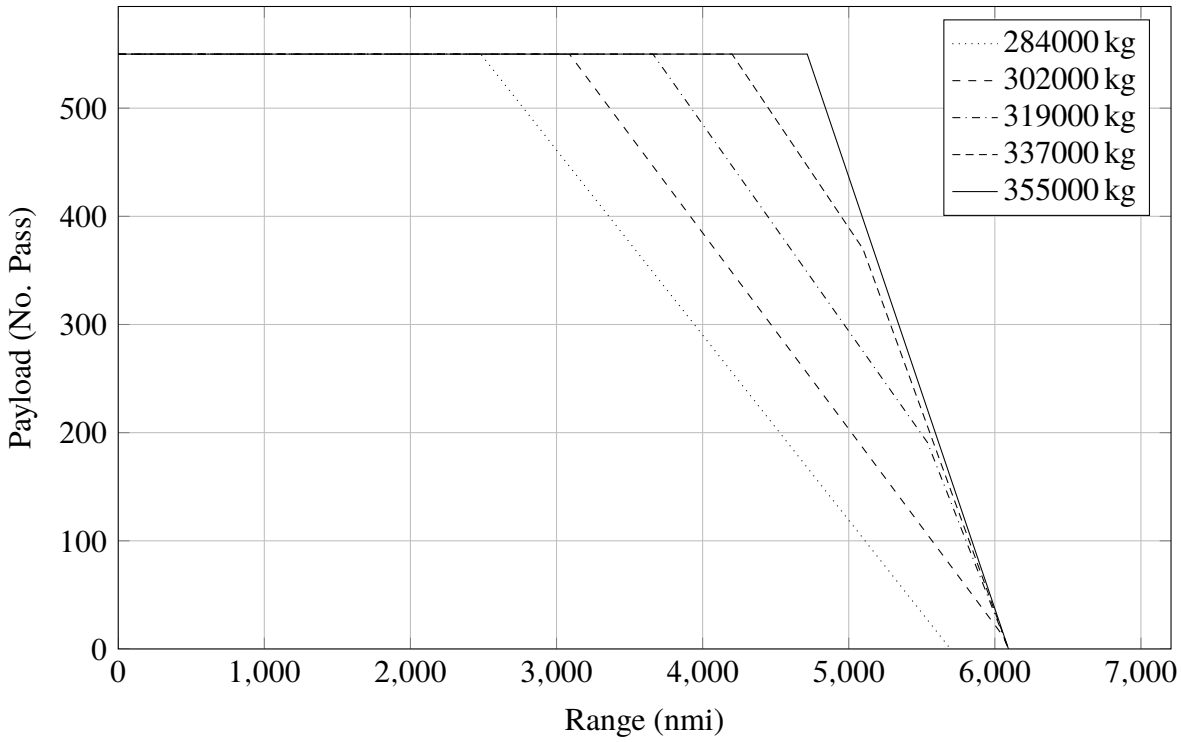

Figure 2.8 B747-100B Payload-Range - Maximum take-off mass parameterization

Chapter 3

Specific Range and Cruise Grid

Once you have tasted flight, you will forever walk the earth with your eyes turned skyward, for there you have been, and there you will always long to return.

– Leonardo da Vinci

In this chapter an anlysis of the specific range is performed with the aim of obtain useful information about cruise performance. In particular, starting from generalized performance evaluation and interfacing them with the fuel consumption, the final objective will be to define the specific range as function of Mach number, obtaining the so called *cruise grid chart* which is a very important tool for pilots because it allows to choose the correct speed, during cruising phase, in order to follow some mission objectives like minimum fuel consumption or a fast cruise.

3.1 Theoretical background

The first step that has to be done in order to obtain the *cruise grid chart* is to define generalized performance in terms of thrust and drag. The *generalized* attribute given to these quantities stands for the fact that they are independents from altitude and this result is reached through the parameter δ which represents a ratio between the total pressure at compressor inlet and the standard pressure at sea level.

Regarding the thrust, the generalized version can be obtained by dividing it by δ as shown below.

$$\frac{T}{\delta} = \frac{T_f}{\delta} - M \cdot a_0 \cdot \frac{\sqrt{\theta \cdot m_a}}{\delta} \tag{3.1}$$

where

- T, is the net thust
- T_f, is the gross thrust
- M, is the mach number

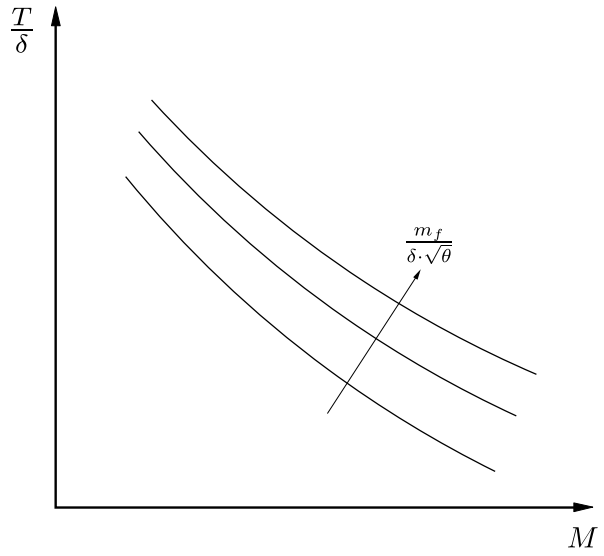

Figure 3.1 Qualitative trend of the generalized thrust v.s. Mach number parameterized in generalized fuel flow rate

- a_0, is the sound speed at sea level
- $\frac{\sqrt{\theta \cdot m_a}}{\delta}$, is the genralized air flow rate which is function of the generalized fuel flow rate given by $\frac{m_f}{\delta \cdot \sqrt{\theta}}$

In this way that the genralized thrust results as a function of Mach number and generalized fuel flow rate.

$$\frac{T}{\delta} = \frac{T_f}{\delta} \cdot f\left(\frac{m_f}{\delta \cdot \sqrt{\theta}}\right) \tag{3.2}$$

As shown in figure 3.1 the generalized thrust decreases with Mach number, at given fuel flow rate, and grows with the latter, at given Mach number. This because if the Mach number grows at fixed fuel flow rate, the air flow rate grows reducing the thrust; otherwise, if fuel flow rate grows at fixed Mach number, air flow rate is lower giving more thrust. With this function it's possible to correlate generalized thrust to hourly fuel consumption which is a main keypoint in building the cruise grid chart of an endurance based aircraft such as Unmanned Aerial Vehicles (UAV). Since transport aircrafts rely more on range performance, it's necessary to obtain the same relationship between generalized thurst and fuel consumption referred to the generalized specific range indicated with $\delta\bar{s}$.

Dividing the generalized fuel flow rate, which is dimesionally equal to $\frac{\text{kg}}{\text{s}}$, by a velocity, the result has a dimension of $\frac{\text{m}}{\text{kg}}$ that represents the reciprocal of the specific range. In this way it's possible to state the following relation.

$$\frac{m_f}{\delta \cdot \sqrt{\theta}} = \frac{M \cdot a_0}{\delta\bar{s}} \tag{3.3}$$

As expectd from the relation (3.3) the thrust has a trend which is the inverse of the previous parameterization. In fact now, for a given Mach number, if the pilot wants to go farther he has to decrease the thrust in order to reduce the fuel consumption; otherwise, at a given distance to reach, the pilot has to increase the thrust in order to make the Mach number grows.

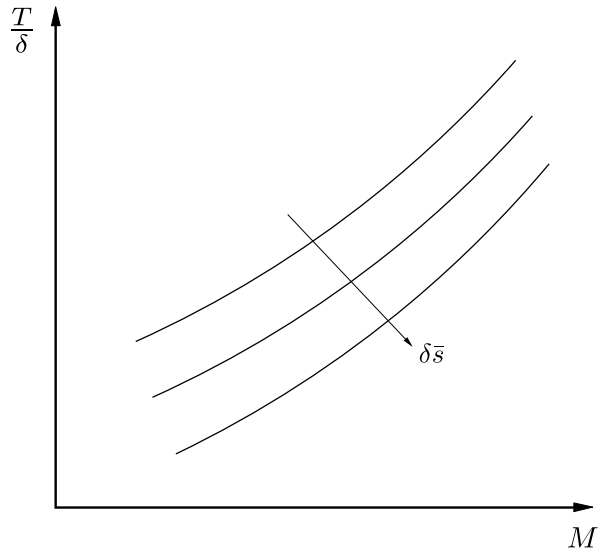

Figure 3.2 Qualitative trend of the generalized thrust v.s. Mach number parameterized in generalized specific range

Since the thrust has always to be compared with the drag in order to evaluate if the aircraft can fly in a specific cruise condition without loosing speed and altitude, it's necessary to obtain a generalized drag trend as well. This ai very similar to the drag trend, as can be seen from figure 3.3, and it depends from aircraft weight as well; but, since these have to be generalized quantities, the weight is a generalized weight too.

By the overlap of figure 3.2 and figure 3.3 charts, it's easy to note that the best Mach number, for a given generalized weight, is located at the intersection of the two curves as reported in figure 3.4. In fact, in order to obtain a bigger specific range at fixed weight, the generalized drag would be higher than the generalized thrust; otherwise, if the pilot wants to fly faster at given weight, he has to increase thrust so that the specific range will decrease due to the increasing fuel consumption. Since during the cruise phase the aircraft weight decreases continuously, the pilot has to gain altitude in order to leave the generalized weight unchanged;

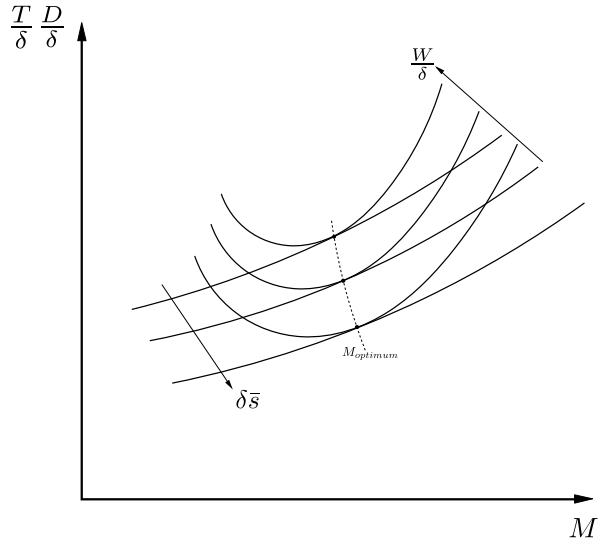

Figure 3.3 Qualitative trend of the generalized drag v.s. Mach number parameterized in generalized weight

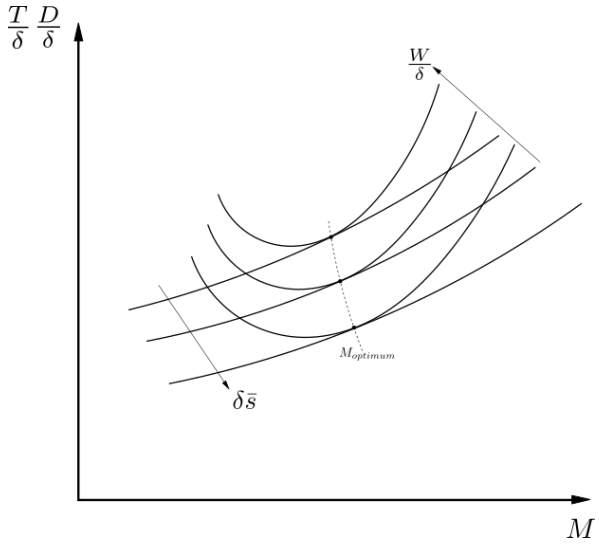

Figure 3.4 Comparison between generalized drag and generalized thrust as functions of Mach number

this explains why during cruise the aircraft continues to climb.

The specific range can also be connected to Breguet formulas (2.1) as it can be obtained by dividng the Autonomy Factor (A.F.) by the aircraft weight; in particular the autonomy factor, groups three main aircraft efficiency and can be written as follow.

$$A.F. = \frac{\eta_p}{SFC} \cdot \left(\frac{L}{D}\right) \qquad \text{if propeller engine driven} \tag{3.4a}$$

$$A.F. = \frac{V}{SFCJ} \cdot \left(\frac{L}{D}\right) \qquad \text{if jet engine driven} \tag{3.4b}$$

where

- η_p, is propeller efficiency
- SFC, or $SFCJ$, is related to propulsive efficiency
- $\left(\frac{L}{D}\right)$, is the aerodynamic efficiency

At given generalized weight and generalized specific range, the optimum Mach is known as explained before and so the autonomy factor can be calculated by multiply $\frac{W}{\delta}$ and $\delta\bar{s}$. Repeating this operation for different generalized weight conditions, allows to define the autonomy factor trend as function of the generalized weight in which each point of the chart is related to an optimum Mach number for the specific range. As can be seen from figure 3.5 the autonomy factor has a maximum at a specific generalized weight which is the one that the pilot should maintain during the cruise phase. If the altitude is fixed, and so δ is constant, the chart in figure 3.5 can be seen as function of Mach number for a given aircraft weight; at this point, knowing that the autonomy factor leads to the specific range if divided by the aircraft weight, it's possible to define the specific range trend as funcion of the Mach number parameterized in aircraft weight.

The chart, so obtained, in figure 3.6 is the one upon which the cruise grid is defined; on the latter, in fact, four lines are drawn each of which is related to a precise mission objecive. It's

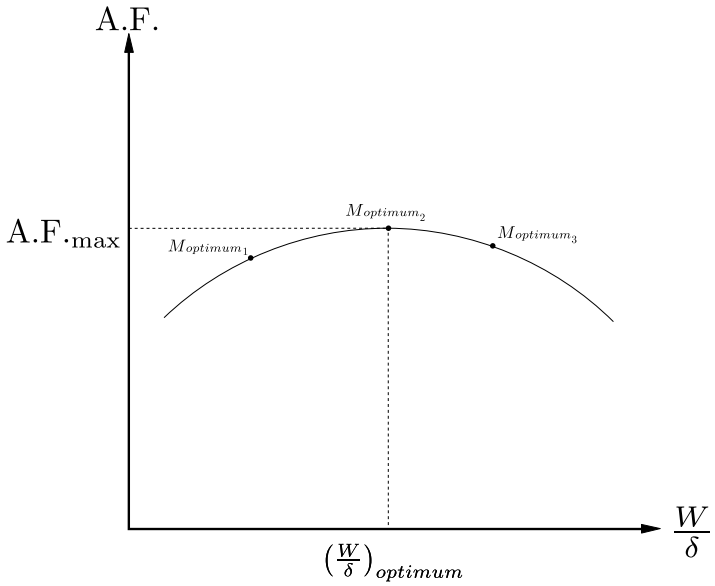

Figure 3.5 Autonomy factor trend as function of the generalized weight

important to highlight that, on long distances, the maximum distance line is not often followed during the cruise because it is tied to a low speed which adversely affects the total flight time increasing the D.O.C.; in order to avoid this condition, pilots prefer to follow the long range line which has only 1% of penalty on the specific range but, at the same time, allows to fly at significantly higher speed with benefits on flight time and, as a result, on the D.O.C..

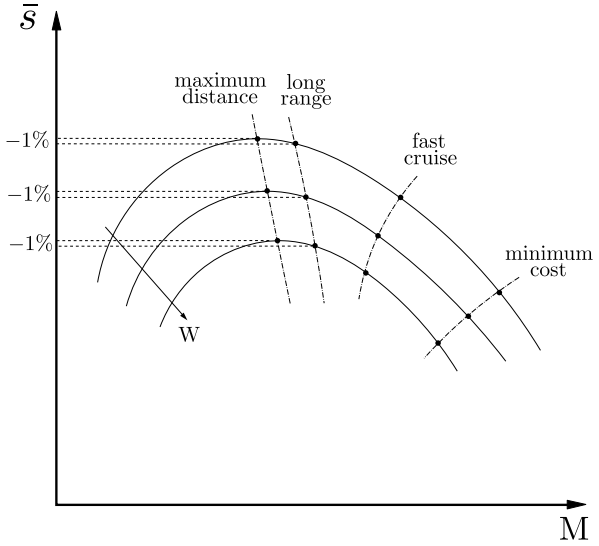

Figure 3.6 Specific range as function of the Mach number parameterized in aricraft weight

3.2 Java class architecture

After having introduced the theory behind the cruise grid chart, a presentation of the related Java class inside JPAD is shown. Using the same philosophy of the previous chapter, a dedicated class, named `SpecificRangeCalc`, has been implemented inside which a series of static methods provide the needed calculation tools. Generaly speaking, the giudeline followed in creating this class is to assign an array of Mach numbers, starting from the one realitve to

the minimum cruising speed and ending with the maximum cruising speed at that altitude and weight, and then evaluate the SFC, from engine database, for each Mach number; from here the $A.F.$ is built after the evaluation of the aerodynamic efficinecy for each value of the same Mach array. Finally the specific range is calculated, in $\frac{\text{nmi}}{\text{lbs}}$, dividing the A.F. by the aircraft max take off mass.

The first static method presented is `calculateEfficiencyVsMach`. It allows to evaluate the aerodynamic efficiency value for each Mach number of a given array through the evaluation of the C_L and the relative C_D from the aircraft total drag polar; in particular the two aerodynamic coefficients are calculated by calling other two static methods which come from two classes of the aerodynamic calculator package of `JPADCore` named, respectively, `LiftCalc` and `DragCalc`. The static `LiftCalc` method, named `calculateLiftCoeff`, performs the C_L calculation using the following formula, valid in cruise phase.

$$C_L = \frac{2W}{\rho S V^2} \tag{3.5}$$

where V, is the True AirSpeed (TAS) derived from the actual Mach number of the array at that the given altitude. On the other hand, the static `DragCalc` method, named `calculateCDTotal`, performs the C_D calculation using the total drag polar expression.

$$C_D = C_{D0} + \frac{C_L^2}{\pi \text{Æ} e} + C_{D\text{wave}} \tag{3.6}$$

In particular the $C_{D\text{wave}}$ is calculated as presented in [17].

The second static method implemented is `calculateSfcVsMach`, which accepts as input data reported in table 3.2 allows to evaluate the SFC of a turboprop aircraft, or the SFCJ of a turbofan aircraft, for each Mach number of the given array by reading data from the related engine database.

Finally, the two previous method leads to the last one named `calculateSpecificRangeVsMach` which allows to calculate the A.F. and, from it, the specific range by implementing the (3.4a) or

`maxTakeOffMass`	Maximum take-off mass
`sweepHalfChordEquivalent`	Equivalent wing sweep angle at half chord
`surface`	Wing surface
`cd0`	Wing c_{D0}
`oswald`	Wing oswald factor
`mach`	An array of Mach numbers
`ar`	Wing aspect ratio
`tcMax`	Mean maximum thickness of the wing
`altitude`	Cruise altitude
`airfoilType`	The wing airfoil type from the related `AirfoilType` Enumeration

Table 3.1 `calculateEfficiencyVsMach` input data

`mach`	An array of Mach numbers
`altitude`	Cruise altitude
`bpr`	Engine By-Pass ratio
`engineType`	The engine type from the related `EngineType` Enumeration

Table 3.2 `calculateSfcVsMach` input data

the (3.4b) depending on the given engine type. To do this, it's necessary to give as input the Mach numbers array, the aerodynamic efficiency array calculated with `calculateEfficiencyVsMach` and the SFC array calculated with `calculateSfcVsMach` as shown in table 3.3.

`maxTakeOffMass`	Maximum take-off mass
`mach`	An array of Mach numbers
`efficiency`	An array of aerodynamic efficiency values
`sfc`	An array of SFC values
`bpr`	Engine By-Pass ratio
`altitude`	Cruise altitude
`eta`	Propeller efficiency (set to zero in case of turbofan)
`engineType`	The engine type from the related `EngineType` Enumeration

Table 3.3 `calculateSpecificRangeVsMach` input data

The class is completed by other four static methods demanded of plotting the results; in particular, using the same approach shown in the previous chapter, a PNG and a TikZ output images are created for each of the SFC, the aerodynamic efficiency and the specific range. It's important to highlight that the first of these plotting methods, which is named `createSpecificRangeChart` and is demanded of plotting the cruise grid chart, has implemented, inside it, the evaluation of the maximum range condition as well as the long range one; this by calculating all maximum points and, from them, all points at -1% of penalty obtained through the evaluation of the bigger of the two intersction points that the line at -1% of penalty defines on each specific range curve. The fourth method is, insetad, used for plotting the intersection between the required and available thrust, giving as input a speed array, the altitude and two List of custom Map named `DragMap` and `ThustMap`; these are two classes used to store all data related to drag and thrust curves like the aircraft weight, the current altitude, the speed array on which the drag or thrust are evaluated and, finally, the drag or thrust array itself.

In conclusion a flowchart of the described class is shown in figure 3.7 in order to simplify its understanding.

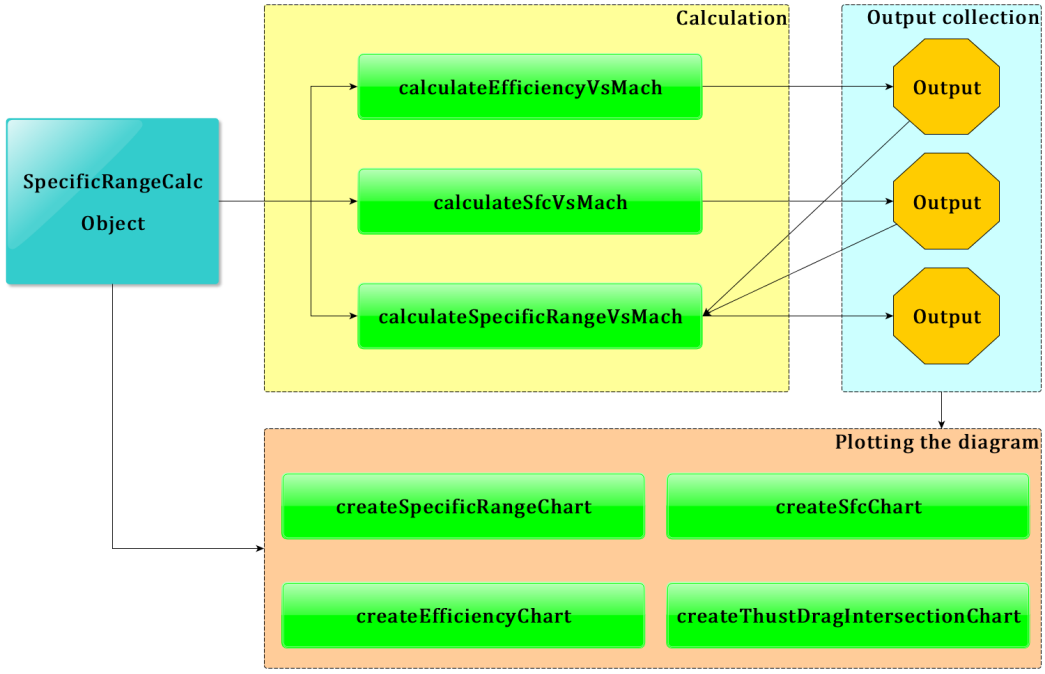

Figure 3.7 `SpecificRangeCalc` class flowchart

3.3 Case study: ATR-72 and B747-100B

In order to validate all the teoretical concepts explained in the previous paragraphs, as well as to give to the reader a useful example of how to handle this class usage, the following pages shows two application of the class methods made upon the two default aircraft described in paragraph 1.4.

Once all preliminary steps have been completed, as explained in paragraph 1.4, it's possible to start the test of this class. First of all, since the cruise grid chart is parameterized in aircraft weight, an array of aircraft masses has to be created; in particular, in this test, a variation of mass of -10% has been implemented, starting from the aircraft maximum take-off mass until it reaches the 60% of the latter. The second step is to define the independent variable, which is the Mach number, and in particular it's necessary to identify the upper and lower limitations of its array; this can be done by evaluating the required thrust, which is equal to the drag during cruise, and comparing it with the available thrust supplied by the power plant. By the intersections of the latter, the two required speeds can be derived; otherwise, if the minimum speed intersection can't be found, the cruising stalling speed, at that weight and altitude condition and for a specified $C_{L\max}$, replaces the missing intersection abscissa. In terms of code, the last step can be carried out using the static method, of the `JPADCore` class `PerformanceCalcUtils`, named `calculateDragThrustIntersection`; the latter requires as input the following data:

- An array of altitudes
- An array of speeds
- An array of weights

```
double[] maxTakeOffMassArray = new double[5];
for (int i=0; i<5; i++)
  maxTakeOffMassArray[i] =  aircraft.get_weights().get_MTOM()
          .plus(aircraft.get_weights().get_MLM())
          .divide(2)
          .getEstimatedValue()
          *(1-0.1*(4-i));

double[] weight = new double[maxTakeOffMassArray.length];
for(int i=0; i<weight.length; i++)
  weight[i] = maxTakeOffMassArray[i]*AtmosphereCalc.g0.getEstimatedValue();
```

Listing 3.1 Mass variation in Specific Range test - B747-100B

- An array of throttle settings
- An array of flight conditions chosen from the `EngineOperatingConditionEnum` Enumeration
- The engine by-pass ratio, set to zero in case of propeller aircraft
- The wing surface
- Wing C_{Lmax} in clean configuration
- A `List` of custom data Map named `DragMap`
- A `List` of custom data Map named `ThrustMap`

The last two input are two collections of Maps designed to store all data necessary to manage curves of drag, or thrust, as function of speed; in particular, in the reported example, the drag and thrust arrays passed to each of them are calculated using the static method of the `JPADCore` classes `DragCalc` and `ThrustCalc` named, respectively, `calculateDragVsSpeed` and `calculateThrustVsSpeed`. The first one implements the classic formula of drag as function of speed and of the C_D calculated with the (3.6), while the second one recognizes the engine type and reads the thrust value from the related external database.

```
// Drag Thrust Intersection
double[] speed = MyArrayUtils.linspace(
    SpeedCalc.calculateTAS(
        0.05,
        theCondition.get_altitude().getEstimatedValue()
        ), // start
    SpeedCalc.calculateTAS(
        1.0,
        theCondition.get_altitude().getEstimatedValue()
        ), // ending
    250 // points
    );

List<DragMap> listDrag = new ArrayList<DragMap>();
for(int i=0; i<maxTakeOffMassArray.length; i++)
  listDrag.add(
```

```
17          new DragMap(
18              weight[i],
19              theCondition.get_altitude().getEstimatedValue(),
20              DragCalc.calculateDragVsSpeed(
21                  weight[i],
22                  theCondition.get_altitude().getEstimatedValue(),
23                  aircraft.get_wing().get_surface().getEstimatedValue(),
24                  aircraft.get_theAerodynamics().get_cD0(),
25                  aircraft.get_wing().get_aspectRatio(),
26                  aircraft.get_theAerodynamics().get_oswald(),
27                  speed,
28                  aircraft.get_wing().get_sweepHalfChordEq().getEstimatedValue(),
29                  aircraft.get_wing().get_maxThicknessMean(),
30                  AirfoilTypeEnum.MODERN_SUPERCRITICAL
31                  ),
32              speed
33              )
34          );
35
36      List<ThrustMap> listThrust = new ArrayList<ThrustMap>();
37      for(int i=0; i<maxTakeOffMassArray.length; i++)
38        listThrust.add(
39          new ThrustMap(
40              theCondition.get_altitude().getEstimatedValue(),
41              1.0, // phi
42              ThrustCalc.calculateThrustVsSpeed(
43                  aircraft.get_powerPlant()
44                          .get_engineList().get(0).get_t0().getEstimatedValue(),
45                  1.0, // phi
46                  theCondition.get_altitude().getEstimatedValue(),
47                  EngineOperatingConditionEnum.CRUISE,
48                  EngineTypeEnum.TURBOFAN,
49                  aircraft.get_powerPlant().get_engineList().get(0).get_bpr(),
50                  aircraft.get_powerPlant().get_engineNumber(),
51                  speed
52                  ),
53              speed,
54              aircraft.get_powerPlant().get_engineList().get(0).get_bpr(),
55              EngineOperatingConditionEnum.CRUISE
56              )
57          );
58
59      List<DragThrustIntersectionMap> intersectionList = PerformanceCalcUtils
60          .calculateDragThrustIntersection(
61              new double[] {theCondition.get_altitude().getEstimatedValue()},
62              speed,
63              weight,
64              new double[] {1.0},
65              new EngineOperatingConditionEnum[]
66                      {EngineOperatingConditionEnum.CRUISE},
67              aircraft.get_powerPlant().get_engineList().get(0).get_bpr(),
68              aircraft.get_wing().get_surface().getEstimatedValue(),
69              cLmax,
```

```
70              listDrag,
71              listThrust
72              );
73      // Definition of a Mach array for each maxTakeOffMass
74      List<Double[]> machList = new ArrayList<Double[]>();
75      for(int i=0; i<maxTakeOffMassArray.length; i++)
76        machList.add(MyArrayUtils.linspaceDouble(
77            intersectionList.get(i).getMinMach(), // start
78            intersectionList.get(i).getMaxMach(), // ending
79            250)); // points
```

Listing 3.2 Intersection of drag and thrust curves in Specific Range test - B747-100B

At this point all static methods of the `SpecificRangeCalc` class are called in order to follow the flowchart steps of figure 3.7 with the purpose of calculating and plotting the cruise grid points.

```
1       // Calculation of the SFC for each Mach array
2       List<Double[]> sfcList = new ArrayList<Double[]>();
3       for(int i=0; i<maxTakeOffMassArray.length; i++)
4         sfcList.add(SpecificRangeCalc.calculateSfcVsMach(
5             machList.get(i),
6             theCondition.get_altitude().getEstimatedValue(),
7             aircraft.get_powerPlant().get_engineList().get(0).get_bpr(),
8             EngineTypeEnum.TURBOFAN
9             ));
10      // Calculation of the Efficiency for each Mach array
11      List<Double[]> efficiencyList = new ArrayList<Double[]>();
12      for(int i=0; i<maxTakeOffMassArray.length; i++)
13        efficiencyList.add(SpecificRangeCalc.calculateEfficiencyVsMach(
14            Amount.valueOf(maxTakeOffMassArray[i],SI.KILOGRAM),
15            machList.get(i),
16            aircraft.get_wing().get_surface().getEstimatedValue(),
17            theCondition.get_altitude().getEstimatedValue(),
18            aircraft.get_wing().get_aspectRatio(),
19            aircraft.get_theAerodynamics().get_oswald(),
20            aircraft.get_theAerodynamics().get_cD0(),
21            aircraft.get_wing().get_maxThicknessMean(),
22            aircraft.get_wing().get_sweepHalfChordEq(),
23            AirfoilTypeEnum.MODERN_SUPERCRITICAL));
24      // Calculation of the Specific range:
25      List<Double[]> specificRange = new ArrayList<Double[]>();
26      for (int i=0; i<maxTakeOffMassArray.length; i++)
27        specificRange.add(SpecificRangeCalc.calculateSpecificRangeVsMach(
28            Amount.valueOf(maxTakeOffMassArray[i], SI.KILOGRAM),
29            machList.get(i),
30            sfcList.get(i),
31            efficiencyList.get(i),
32            theCondition.get_altitude().getEstimatedValue(),
33            aircraft.get_powerPlant().get_engineList().get(0).get_bpr(),
34            0.85,
35            EngineTypeEnum.TURBOFAN));
36
```

```
37      // PLOTTING:
38      // building legend
39      List<String> legend = new ArrayList<String>();
40      for(int i=0; i<maxTakeOffMassArray.length; i++)
41        legend.add("MTOM = " + maxTakeOffMassArray[i] + " kg ");
42
43      SpecificRangeCalc.createSpecificRangeChart(specificRange, machList, legend);
44      SpecificRangeCalc.createSfcChart(sfcList, machList, legend,
        EngineTypeEnum.TURBOFAN);
45      SpecificRangeCalc.createEfficiencyChart(efficiencyList, machList, legend);
46      SpecificRangeCalc.createThrustDragIntersectionChart(
47              theCondition.get_altitude().getEstimatedValue(),
48              maxTakeOffMassArray,
49              listDrag,
50              listThrust,
51              speed
52              );
53    }
54    // END OF THE TEST
```

Listing 3.3 Intersection of drag and thrust curves in Specific Range test - B747-100B

In conclusion, the following images shows the cruise grid, and all the evaluated quatities, calculated with this class and referred to the ATR-72 and of the B747-100B. It's has to be noted that, as explained in the first paragraph, the maximum range condition is actually related to a low speed and also that, with a penalty of -1% in range, the specific range is about the same with a significantly higher speed; for example, in figure 3.15, the cruising Mach number, related to the maximum range condition of the B747-100B, varies, with the aircraft weight, between 0.74 and 0.80 while the long range one varies between 0.80 and 0.82, being more similar to the real cruising Mach number. Moreover, the specific fuel consumption, from figures 3.10 and 3.11, grows with the Mach number, as expected, due to the major thrust given by engines; while the efficiency, from figures 3.12 and 3.13, decreases rapidly in the B747-100B case due to the increasing wave drag, not present in the ATR-72 case.

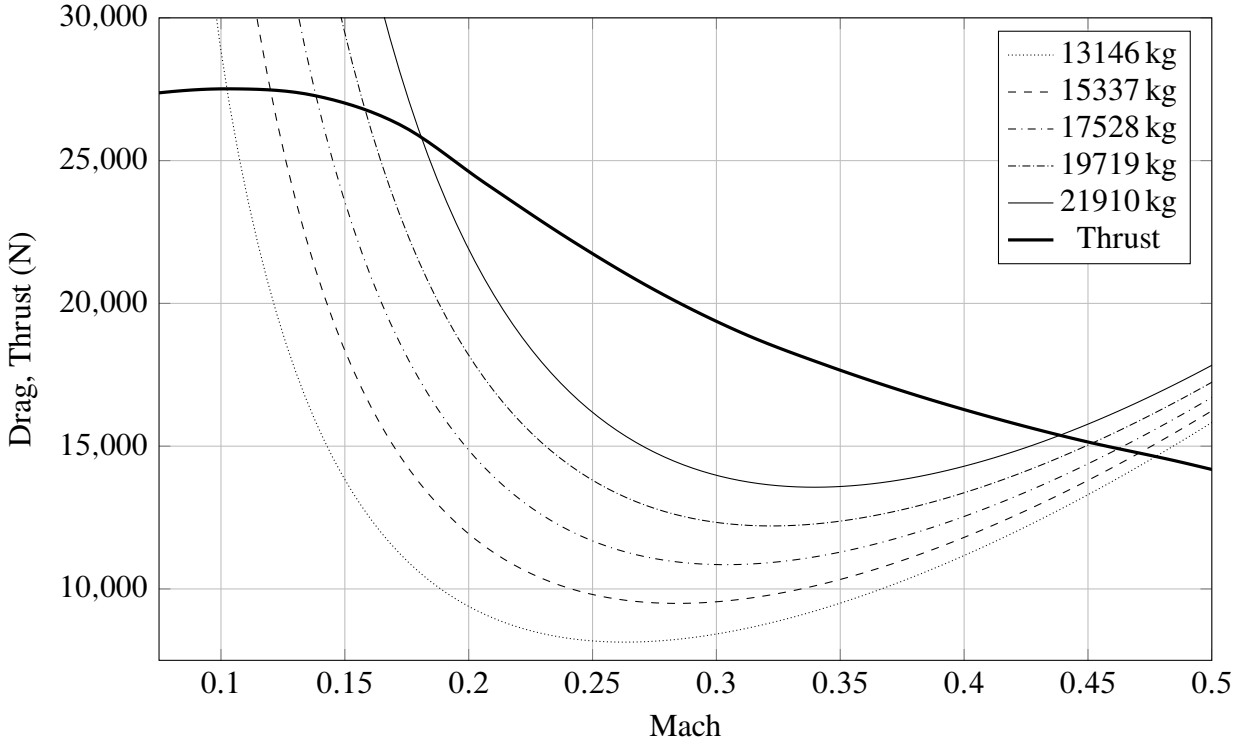

Figure 3.8 Intersection of drag and thrust curves at 6000m - ATR-72

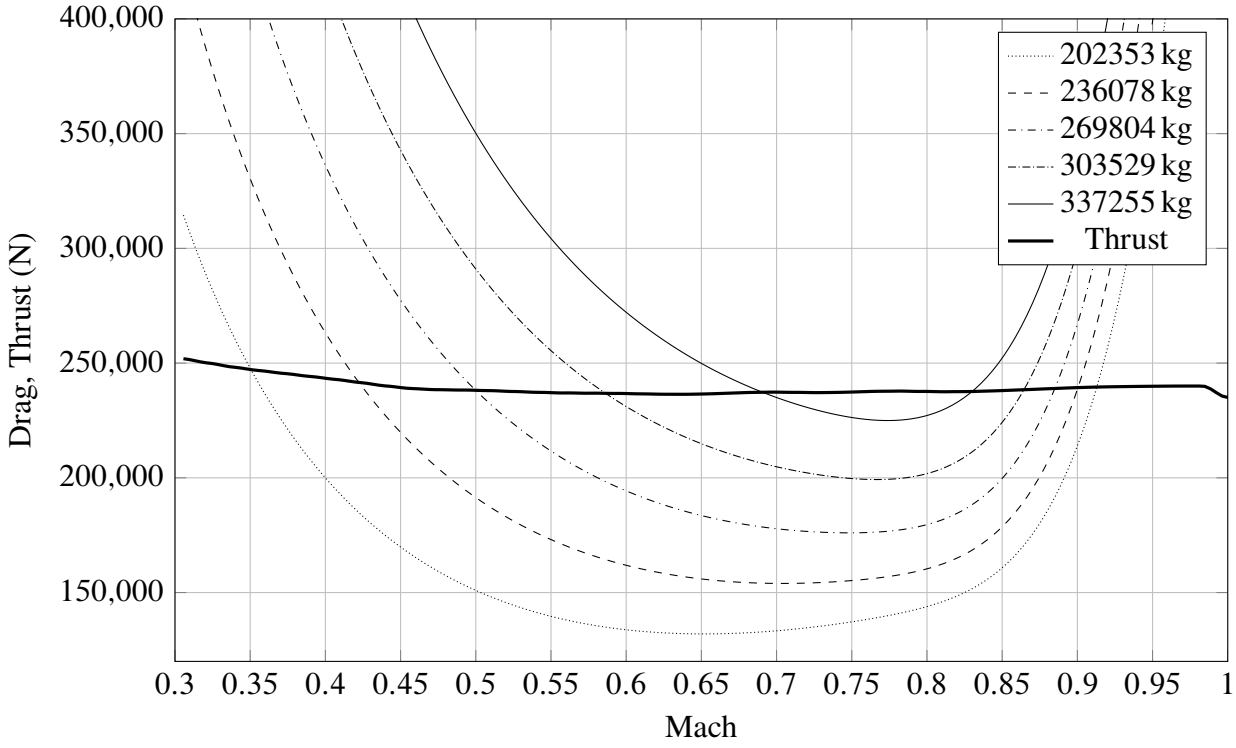

Figure 3.9 Intersection of drag and thrust curves at 10000m - B747-100B

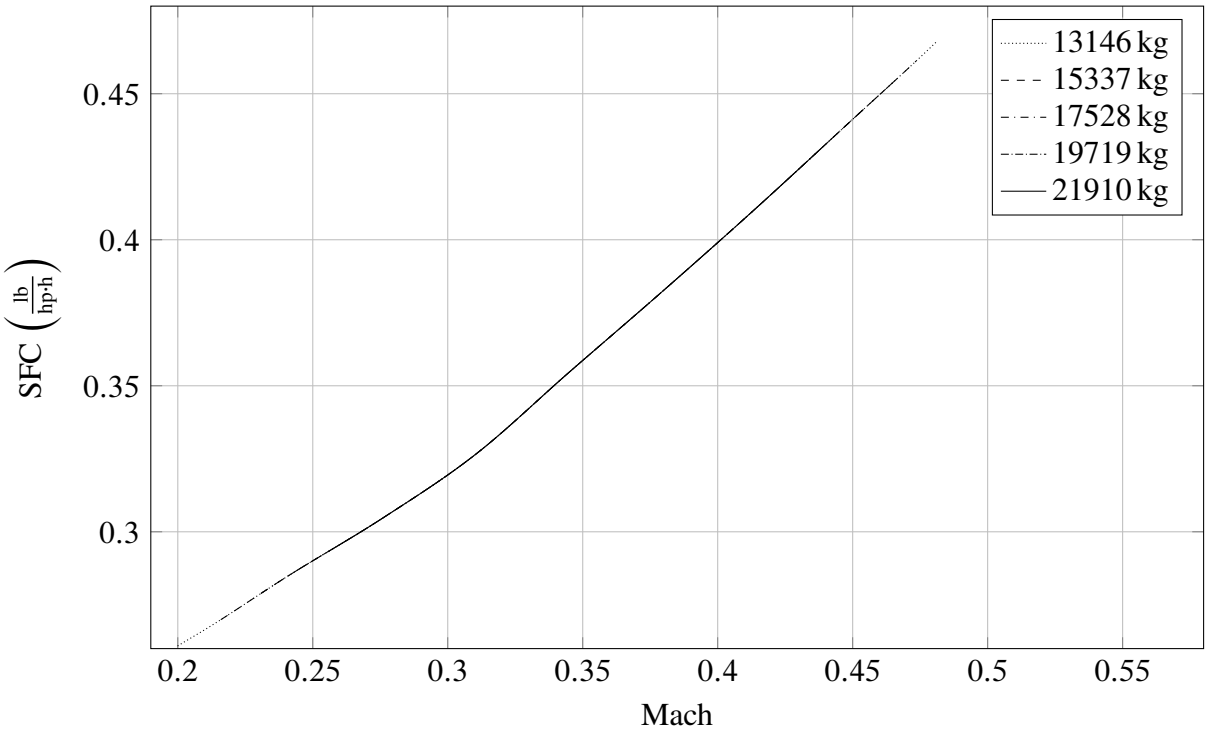

Figure 3.10 ATR-72 specific fuel consumption against Mach number at 6000m

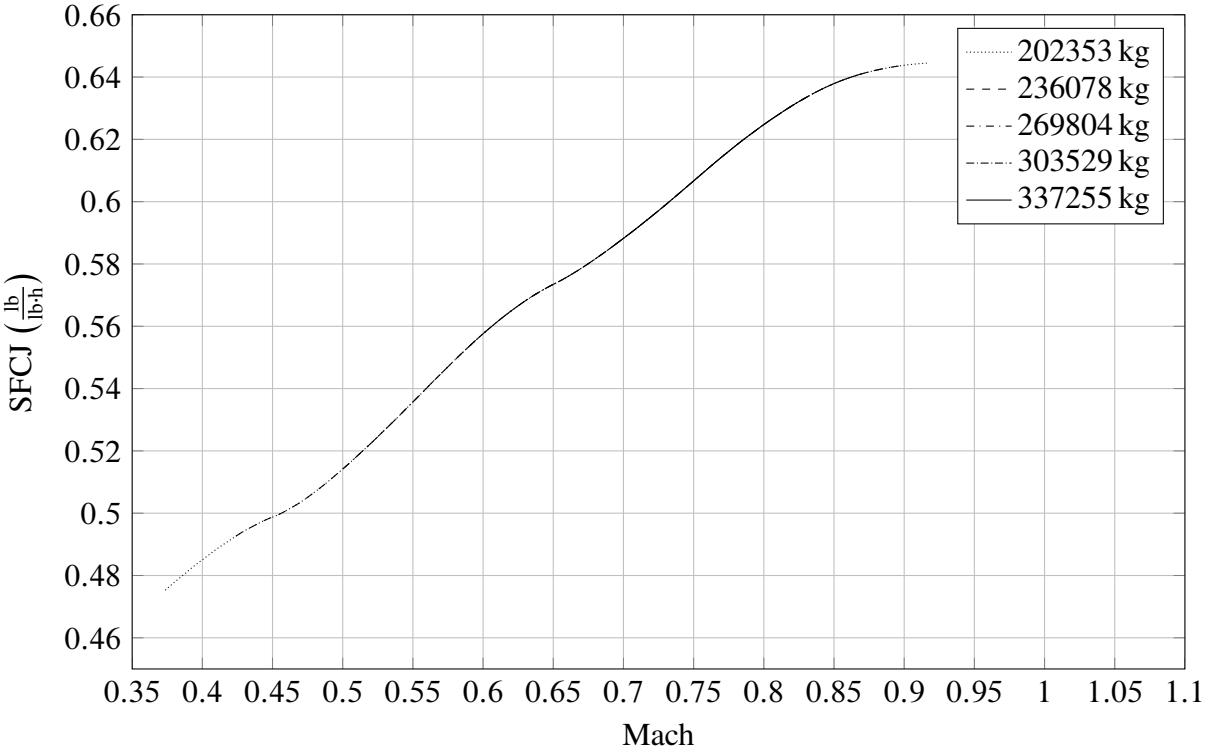

Figure 3.11 B747-100B specific fuel consumption against Mach number at 10000m

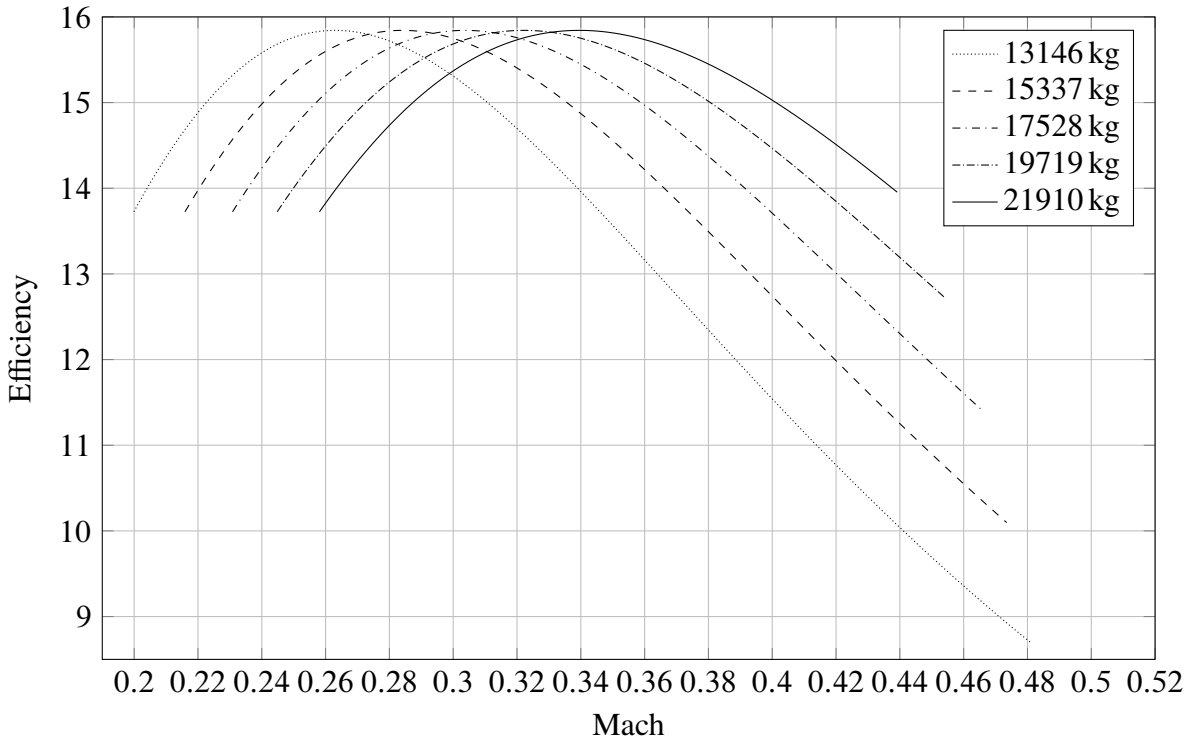

Figure 3.12 ATR-72 efficiency against Mach number at 6000m

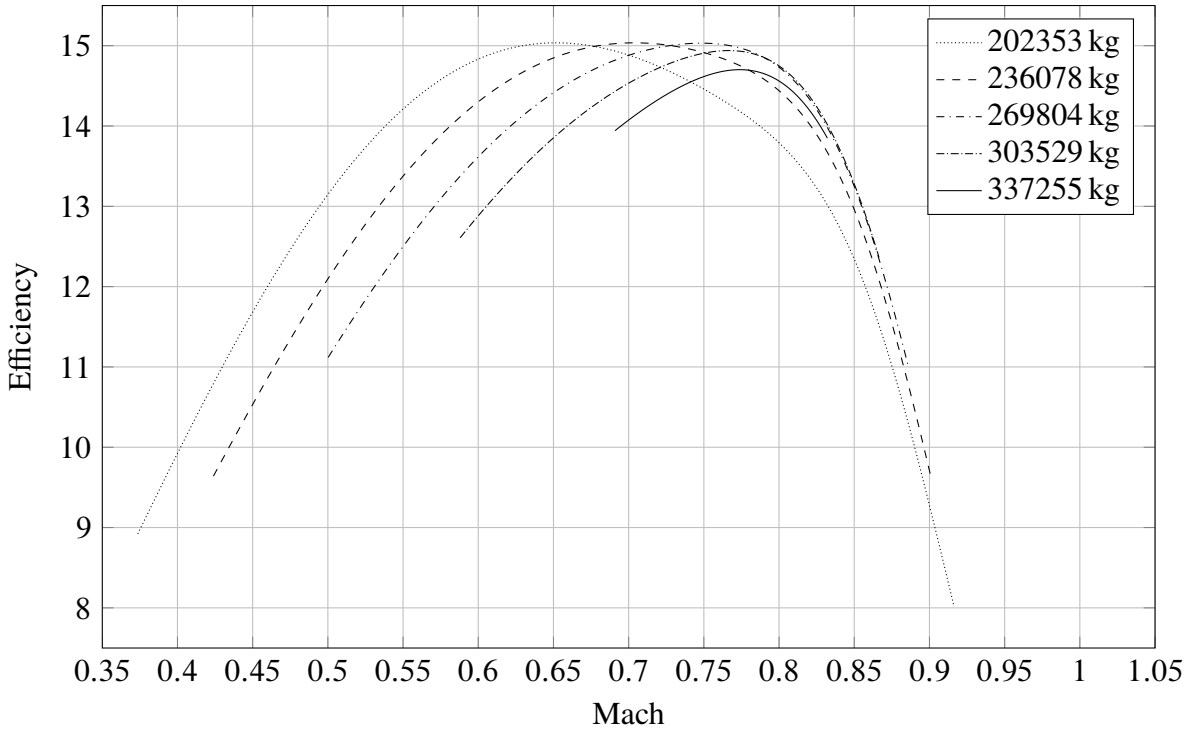

Figure 3.13 B747-100B efficiency against Mach number at 10000m

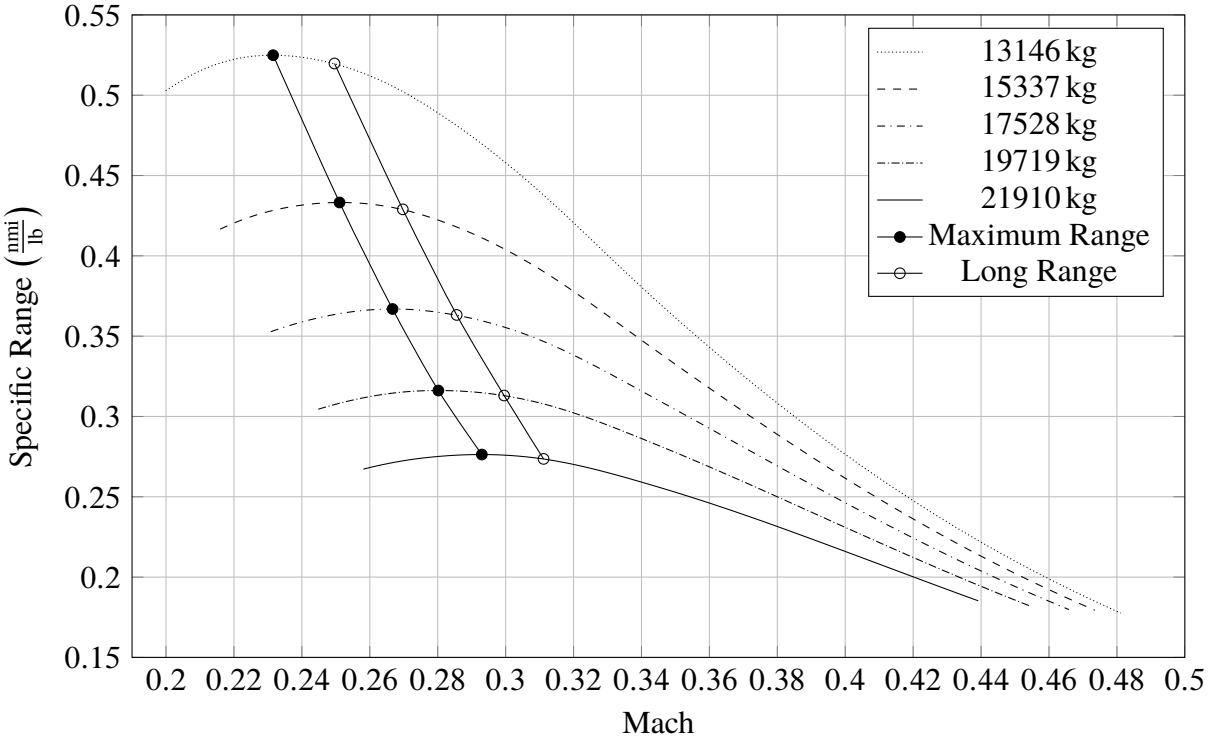

Figure 3.14 ATR-72 cruise grid chart at 6000m

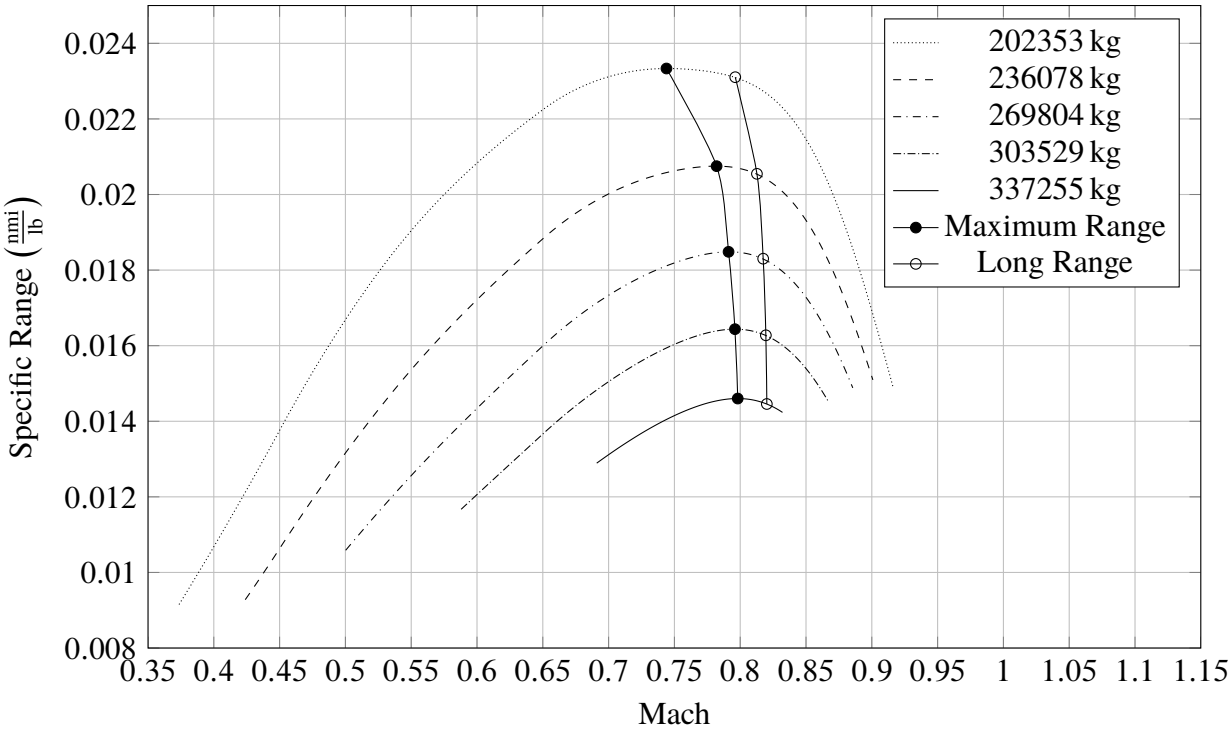

Figure 3.15 B747-100B cruise grid chart at 10000m

Chapter 4

High lift devices effects

There are two critical points in every aerial flight—its beginning and its end.

– Alexander Graham Bell, 1906

In the preliminary design of the wing a large number of requirements have to be fulfilled as exemplified in table 4.1; but, as many of them are in conflict, it is hardly ever possible to check them all, so that a compromise has to be found.

High lift to drag ratio L/D
Satisfactory maximum lift coefficient
Satisfactory stall quality
High value of the critical Mach number
Low weight
Ensure satisfactory performance in all flight phases

Table 4.1 Some wing design requirements

With respect of the last shown requirement, the wing is usually equipped with high-lift devices, which change it's shape, in order to make it performant both in cruise both in take-off and landing phases. In fact, as can be seen from table 4.2, these phases show conflicting objectives which can only be mediated through the introduction of these devices.

Cruise requirements	**Take-Off and Landing requirements**
Small wing surface and high wing loading W/S	Big wing surface, or high C_{Lmax}, in order to have an high equivalent wing loading $m = W/(S \cdot C_{Lmax})$
Small camber	
Low drag	High drag value in landing
High speed	Lift generated using high C_L due to the low speed
Lift generated using low C_L	

Table 4.2 Comparison between cruise and take-off/landing design requirements

4.1 Theoretical background

In this paragraph, a general overview of the different type of high-lift devices is provided as well as the semi-empirical steps used to predict their effects on aerodynamic perofmance of the wing.

The disigner may choose from a large collection of feasible high-lift systems, although in the case of a specific project this freedom will be limited, since incremental drag, mechanical complexity, development and maintenance costs and structural weight are all factors to be considered [31].

All high-lift devices can be divided in two main groups of which only the first one will be analyzed in this discussion:

- Systems for passive lift increase, such as *leading-edge devices* or *trailing-edge devices*, which modify the wing shape.
- Systems for active lift increase, such as *blown flaps* or *jet flaps*, which acts directly on the flow in order to control it.

Genrally speaking, *trailing-edge devices* are used to increase the wing maximum lift coefficient, while *leading-edge devices* are used to increase the stalling angle of attack. A more in depth analysis shows that *trailing-edge devices* increase the camber and improve the flow at the trailing edge, but tend to promote leading edge stall on thin sections and may cause a reduction in the stalling angle of attack; on the other hand *leading-edge devices* postpone or eliminate leading edge stall, but they have little effects on the airfoil camber as a whole, although locally the camber is increased [31].

About *trailing-edge devices*, their effects can be resumed in:

- Higher C_L at a given angle of attack and higher C_{Lmax}
- Lower stalling angle of attack
- Lower zero-lift angle due to increasing camber

while *leading-edge devices* provide the followings:

- Extension of the linear trait of the lift curve with an increase of the maximum angle of attack and of the C_{Lmax}
- Higher zero-lift angle due to translation of the lift curve on the right caused by leading edge deflection which reduce the actual angle of attack
- Higher slope of the linear trait of the lift curve, for those devices which extend airfoils chords with the effect of increase the wing surface and, with constant wing span, the aspect ratio

Among the variety of different devices, only the followings will be taken into account as they represents the most used ones.

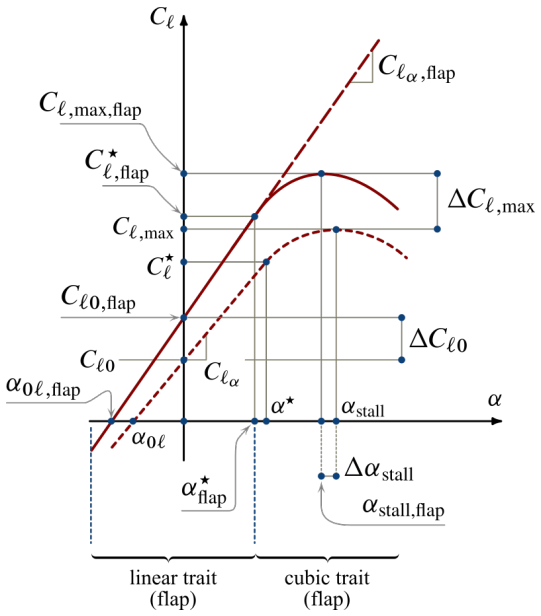

Figure 4.1 Trailing-edge devices effects

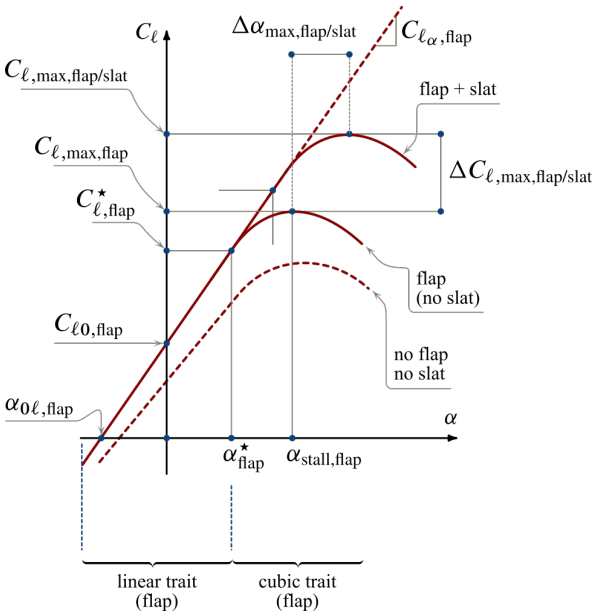

Figure 4.2 Leading-edge devices effects

Plain flap This device is most used on small aircraft or ones equipped of a thin wing because it doesn't support a complex mechanism of retraction. Typical deflections are about 20° for take-off and 60° for landing.

Single slotted flap It can be seen as a plain flap with a gap between the two elements composing the airfoil. The single slotted flap has very little flap overlap with the fixed trailing edge and hence develops only little Fowler motion, that is the aft travel of the flap that increases the section chord. Its typical deflections are about 20° for take-off and 45° for landing. The effects of a single slotted flap show an increment in all the aerodynamic coefficients, but it must be said that the increment in drag is lower than that for plain flaps. The slotted flap chord usually ranges from the 25% up to the 30% of the section chord. Moreover the slot influences boundary layer control, in fact it introduces a blowing that energizes the boundary layer delaying separation, so an increase in lift is generated.

Double slotted flap This device is superior to the previous type at large deflections, because separation of the flow over the flap is postponed by the more favourable pressure distribution. Its typical deflections are about 20° for take-off and 50° for landing; in particular, in order to avoid an increasing twisting moment due the deflection, this devices are usually combined with leading edge slats deflected of the same quantity.

Triple slotted flap This device is used on several transport aircraft with very high wing loadings. In combination with leading edge devices, this system represents almost the ultimate achievement in passive high-lift technology, but its shape shows that complicated flap supports and controls are required. Its typical deflections are about 20° for take-off and 50° for landing.

Fowler flap It is theoretically a single slotted flap that adds to the downward deflection also a backward motion that allows the increment of the effective chord and camber. Due to the

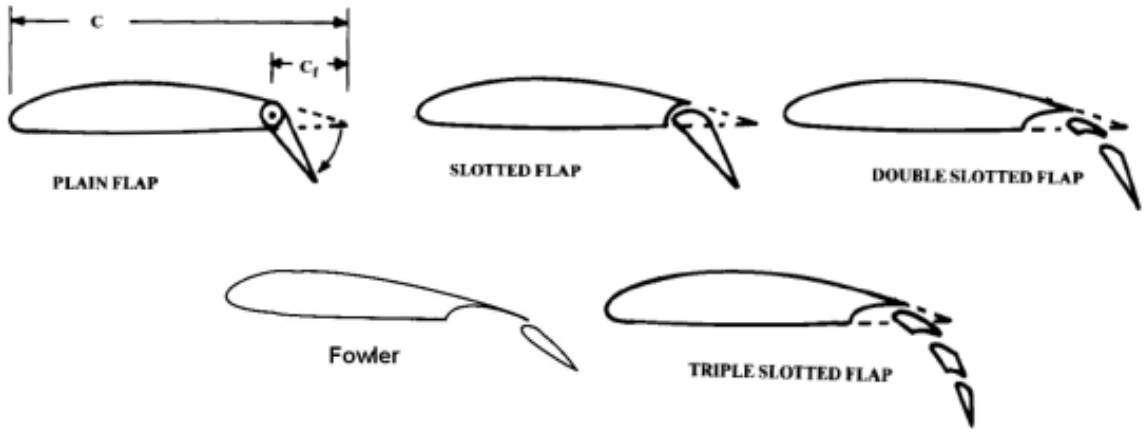

Figure 4.3 Analyzed trailing edge devices types

necessity of keeping the rear part of the wing section extended out the main element its implementation system is usually more complicated than the single slotted flaps but its weight and costs are largely justified by its high lift effectiveness. Its typical deflection are about 40° for landing and 15° for take-off, smaller than other types because the chord extension provides a bigger lift increment due to the bigger wing surface; this also reduces the drag in take-off wth benefit an the required field length.

Slat It's the most efficient leading edge device; thanks to combined deflection and forward motion, it acts in order to increase airfoil camber, and so the maximum lift coefficient, as well as increase the airfoil chord with the result of a bigger surface which provides a bigger aspect ratio with the effect of increase the lift curve slope of the linear trait. Furthermore, thanks to the slot which provides a boundary layer energization, it also increases the stalling angle of attack.

Krueger flap It acts in the same way as the slat, but it is thinner and more suitable for installation on thin wings. Krueger flaps are very common because of their simple architecture.

Plain leading edge flap Is less effective than slat since it has no slot, it is mechanically simple and rigid and particularly suitable for thin airfoil sections. The leading edge can be hinged in order to move it backward (droop nose) or it has a mechanism inside that changes the curvature of the nose (variable camber flap).

Leading edge fixed slot It has a fixed slot at the leading edge that, at high angle of attack, allows the airflow to pass through energizing the boundary layer; this helps to increase the stalling angle of attack. During the criuse phase, in which the angle of attack is small, the gap is usually seald.

In order to predict, from the preliminary design phase, the aerodynamic characteristics of the high-lift devices, some useful semi-empirical methods are available; in this particular case the followings formulas and charts are taken from [31] and [26].

The guideline that will be followed provides to analyze separately the trailing edge and the leading edge devices effects; moreover it will start by evaluating the changes in aerodynamic

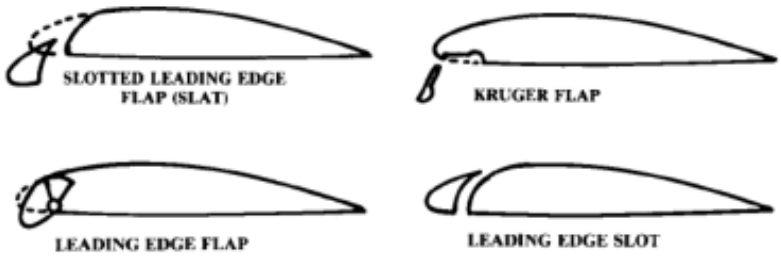

Figure 4.4 Analyzed leading edge devices types

characteristics of airfoils for then extends these to entire wing. From figures 4.1 and 4.2, it's possible to understand that the main changes introduced by trailing edge, or leading edge, devices are related to the evaluation of four quantities:

- ΔC_{l0}
- $\Delta C_{l\max}$
- $C_{l\alpha,\text{flap}}$
- $\Delta\alpha_{\text{stall}}$

4.1.1 ΔC_{l0} and ΔC_{L0} calculation

An empirical method for predicting airfoil lift increments at zero angle of attack for high-lift systems (ΔC_{l0}) comes from the Glauert's linearized theory for thin airfoils with flaps. A result obtained from this theory for the lift due to flap deflection is the following.

$$\alpha_\delta = 1 - \frac{\theta_f - \sin(\theta_f)}{\pi} \tag{4.1}$$

In particular θ_f can be calculated as follows.

$$\theta_f = \cos^{-1}\left(2\,\frac{c_f}{c} - 1\right) \tag{4.2}$$

Known this value, it's possible to evaluate the theoretical Δc_{l0} which can be calculated as proposed in (4.3).

$$\Delta C_{l0} = \alpha_\delta\, C_{l\alpha}\, \delta_f \tag{4.3}$$

with $C_{l\alpha}$ equals to the linear slope of the llift curve of the airfoil, and δ_f the flap angular deflection. For large flap deflections and for the separation at large flap angles due to viscosity, linear theory is in error when compared with exact one, for this reason we assume the effectiveness factor η_δ, so the formulation becomes the following.

$$\Delta C_{l0} = \alpha_\delta\, C_{l\alpha}\, \delta_f\, \eta_\delta \tag{4.4}$$

More in detail, η_δ can be evaluated from the charts provided in figures 4.5 and 4.6. In case of flaps which extend the airfoil chord, this effects also contributes to the lift increase and can be taken into account by referring the section lift to the extended chord and then converting the

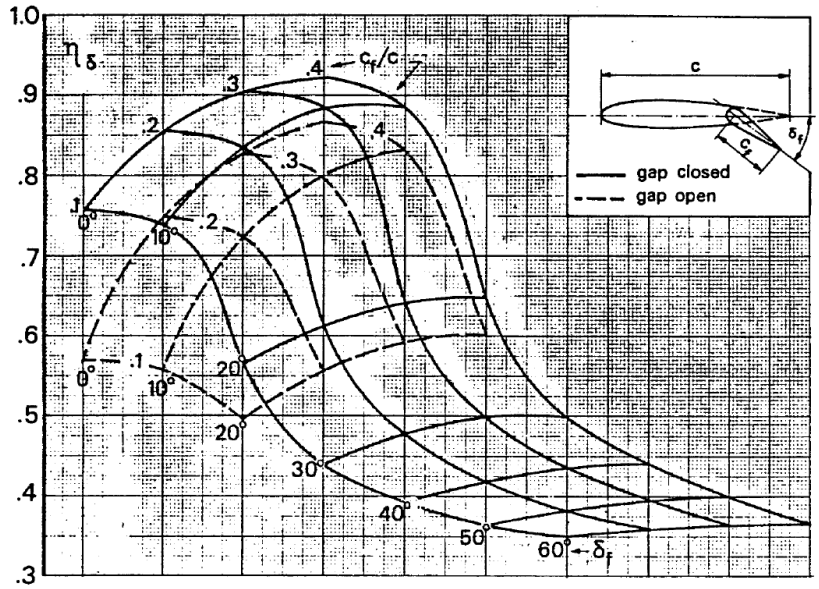

Figure 4.5 η_δ for plain flap

result to the original chord. As a results of this, the section lift coefficient can be evaluated as shown in (4.5).

$$C_l = \left(C'_{l0} + \Delta C'_{l0}\right) \frac{c'}{c} \tag{4.5}$$

Here the variables with superscript are referred to the extended chord. Assuming that for the basic section $C'_{l0} = C_{l0}$, it's possible to derive the ΔC_{l0} as folllows.

$$\Delta C_{l0} = \Delta C'_{l0} \frac{c'}{c} + C_{l0} \left(\frac{c'}{c} - 1\right) \tag{4.6}$$

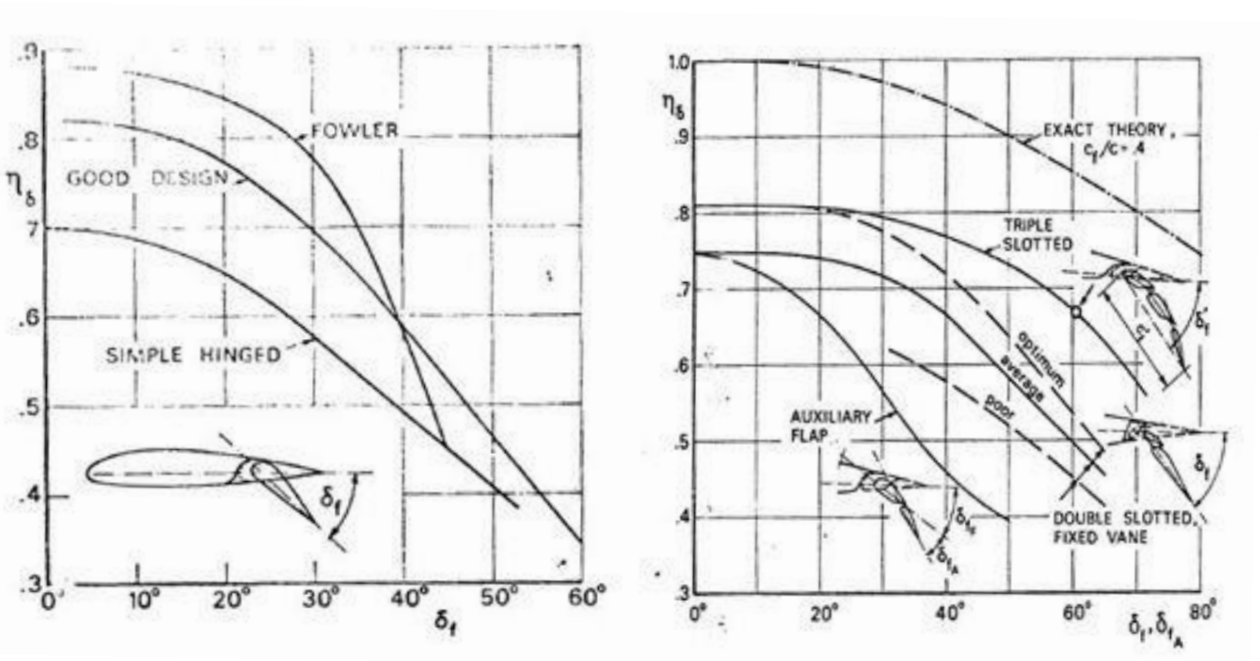

Figure 4.6 η_δ for other type of flaps

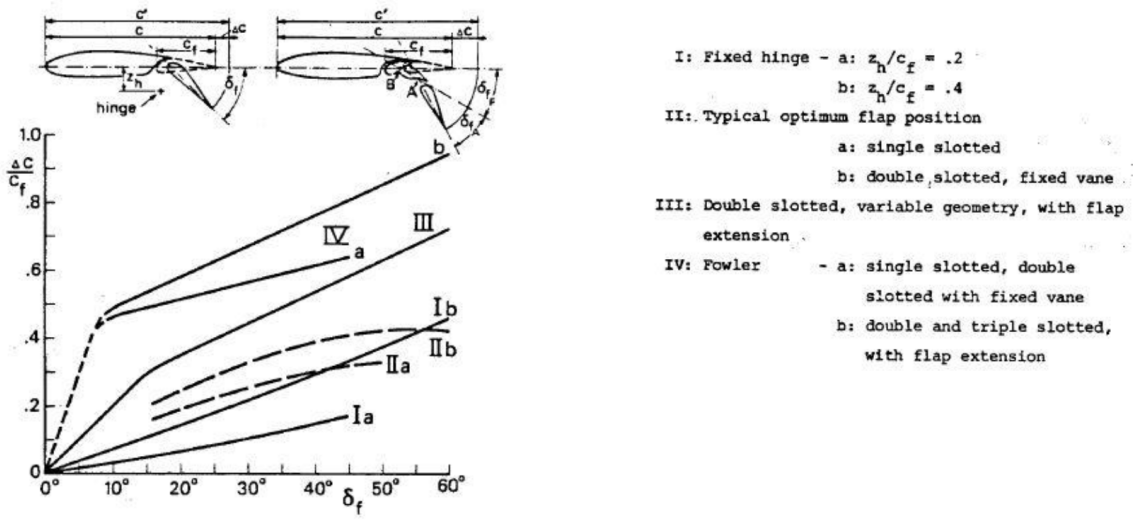

Figure 4.7 $\frac{\Delta c}{c_f}$ for different type of flaps as function of flap deflection

In particular C_{l0} is known, $\Delta C'_{l0}$ is calculated as in (4.4) and $\frac{c'}{c}$ is equal to the following expression.

$$\frac{c'}{c} = 1 + \frac{\Delta c}{c_f}\frac{c_f}{c} \tag{4.7}$$

In particular $\frac{\Delta c}{c_f}$ can be derived from the charts of figure 4.7. The ΔC_{l0} so calculated has now to be extended to the entire wing; this can be through the following forumla.

$$\Delta C_{L0} = \Delta C_{l0}\left(\frac{C_{L\alpha}}{C_{l\alpha}}\right)\left[\frac{(\alpha_\delta)_{C_L}}{(\alpha_\delta)_{C_l}}\right]K_b \tag{4.8}$$

where $C_{L\alpha}$ and $C_{l\alpha}$ are respectively the lift curve slopes of the wing and the airfoil, $\left[(\alpha_\delta)_{C_L}/(\alpha_\delta)_{C_l}\right] = K_c$ is the ratio of the three-dimensional flap effectiveness parameter to the two dimensional flap effectiveness one, which can be derived from the figure 4.8, and K_b is a flap span effectiveness factor, which is function of the flap span-wise extension, that can be read from 4.9.

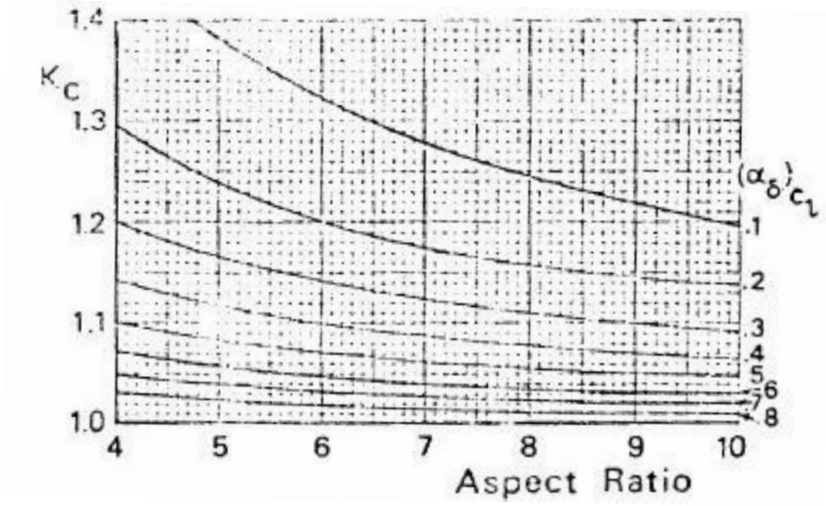

Figure 4.8 K_c for different $(\alpha_\delta)_{C_L}$ as function of wing aspect ratio

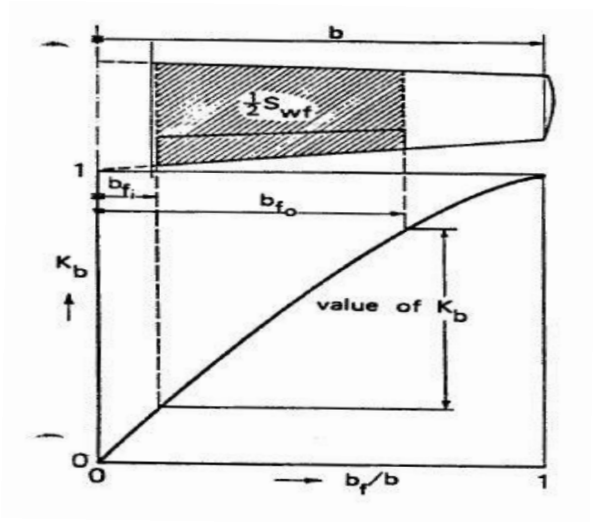

Figure 4.9 K_b as function of flap span to wing span ratio ($\frac{b_f}{b}$)

4.1.2 $\Delta C_{l\max}$ and $\Delta C_{L\max}$ calculation for trailing edge and leading edge devices

An empirical method for predicting airfoil maximum lift increments for plain and slotted flaps is presented in DATCOM and will be followed from [26]. The maximum lift increment provided to an airfoil by the deflection of a trailing edge flap is given by the (4.9).

$$\Delta C_{l\max} = k_1\, k_2\, k_3\ \ (\Delta C_{l\max})_{\text{base}} \tag{4.9}$$

Here $(\Delta C_{l\max})_{\text{base}}$ is the section maximum lift increment for 25 percent-chord flaps at the reference flap-deflection angle and is shown in figure 4.10 for different flap systems. The quantity k_1 is a factor accounting for flap-chord-to-airfoilchord ratios, $\frac{c_f}{c}$, other than 0.25 and is shown in figure 4.11. The quantity k_2 is a factor accounting for flap deflections other than the reference value and is shown in figure 4.12. Finally, k_3 is a factor accounting for flap motion as a function of flap deflection and is shown in figure 4.13.

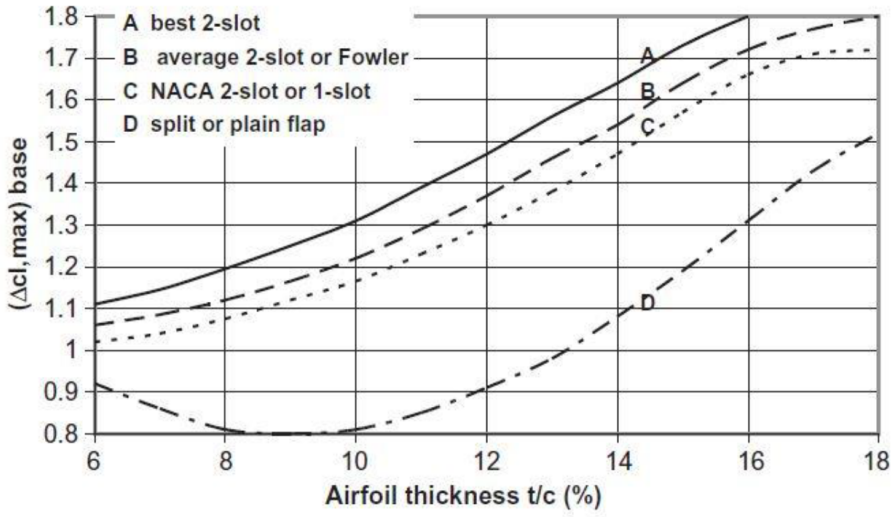

Figure 4.10 $(\Delta C_{l\max})_{\text{base}}$ DATCOM chart

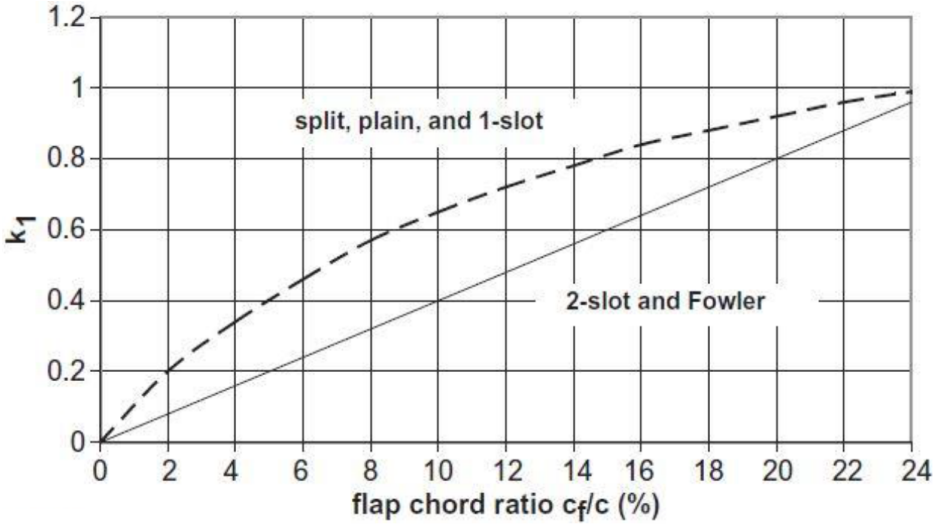

Figure 4.11 k_1 correction factor for trailing edge flap chord to airfoil-chord ratios, $\frac{c_f}{c}$, other than 0.25

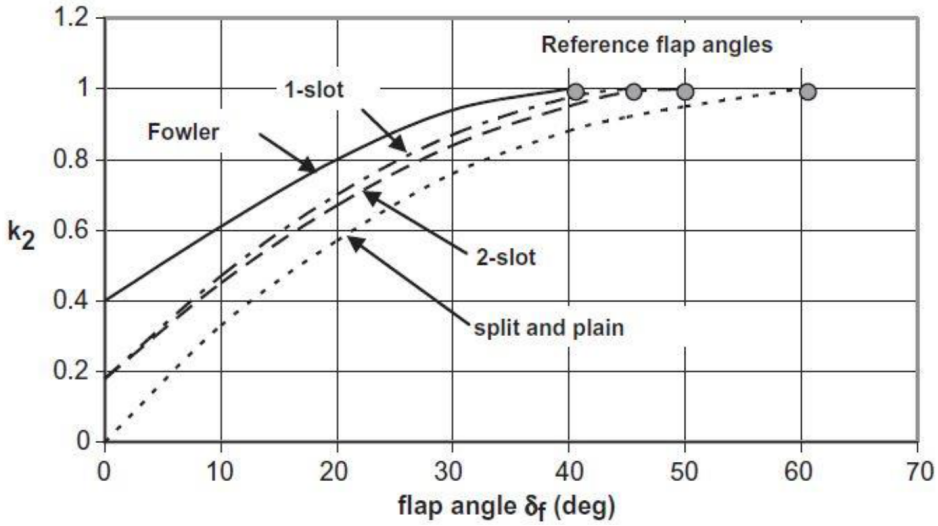

Figure 4.12 Flap angle correction factor k_2. The reference flap angle for each type of flap is shown as a solid symbol at $k_2 = 1$

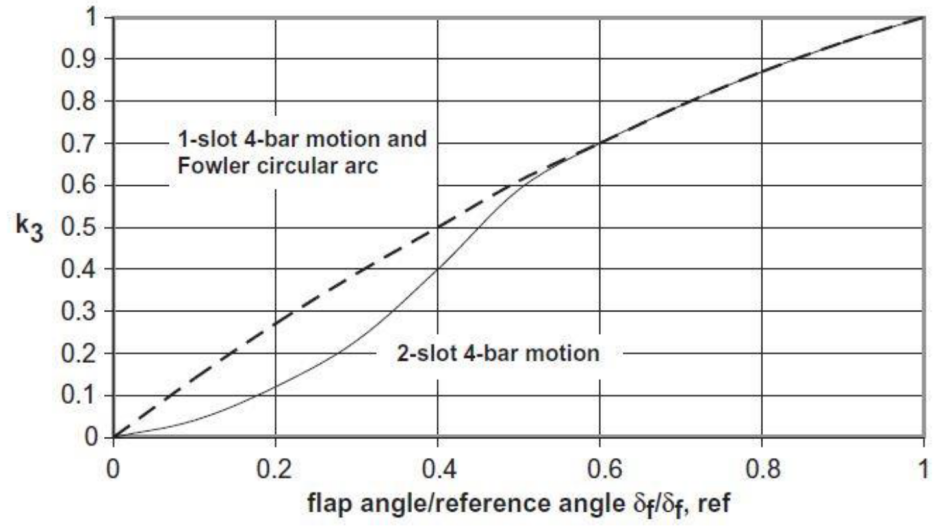

Figure 4.13 Flap motion correction factor k_3

Regarding leading edge devices, the leading edge slats are the most common ones; as explained previously, the latter increase the maximum lift coefficient of the airfoil along with the stall angle. As a result, they are commonly used, particularly in landing, though they are also useful on takeoff because the lift increment they develop comes with little drag penalty. The DATCOM method for leading edge flaps and slats proposes that the maximum lift increment for leading edge flaps or slats may be approximated by the following empirical relation.

$$\Delta C_{l\max} = \left(\frac{\partial C_l}{\partial \delta}\right)_{\max} \eta_{\max}\ \eta_\delta\ \delta_s\ \frac{c'}{c} \tag{4.10}$$

The first term in the previous equation is the theoretical lift effectiveness which gives the rate of change of the lift coefficient with change in deflection angle; it is shown in figure 4.14 as a function of the leading edge flap, or slat, chord to airfoil-chord ratio $\frac{c_s}{c}$. The second term, $\eta_{\max}$, is an empirical factor which accounts for the effects of airfoil leading edge radius and maximum thickness. A graph of this factor is presented in figure 4.15; the discontinuity in the curve for slats is said to be due to a lack of data in the region of the discontinuity, but an *ad hoc* correction is also proposed in order to provide more accurate results. The third term, η_δ, is another empirical factor which corrects for flap, or slat, deflections different from the optimum flap angle. This parameter is shown in figure 4.16 as a function of the flap, or slat, deflection angle δ_s. To understand the angle δ_s one may first imagine drawing a chord line on the slat-airfoil combination when the slat is stowed; then when the slat is deflected, the segment of the chord line that was drawn on the slat in the stowed position has now rotated through the deflection angle δ_s. This is the standard used in the DATCOM method and is not necessarily used throughout the literature as a definition of slat deflection angle. Finally, the ratio $\frac{c'}{c}$ accounts for the apparent increase in chord length when the slat is deflected and a slot is formed between the two airfoil elements; this dimension, along with the deflection angle δ_s, is illustrated in figure 4.17.

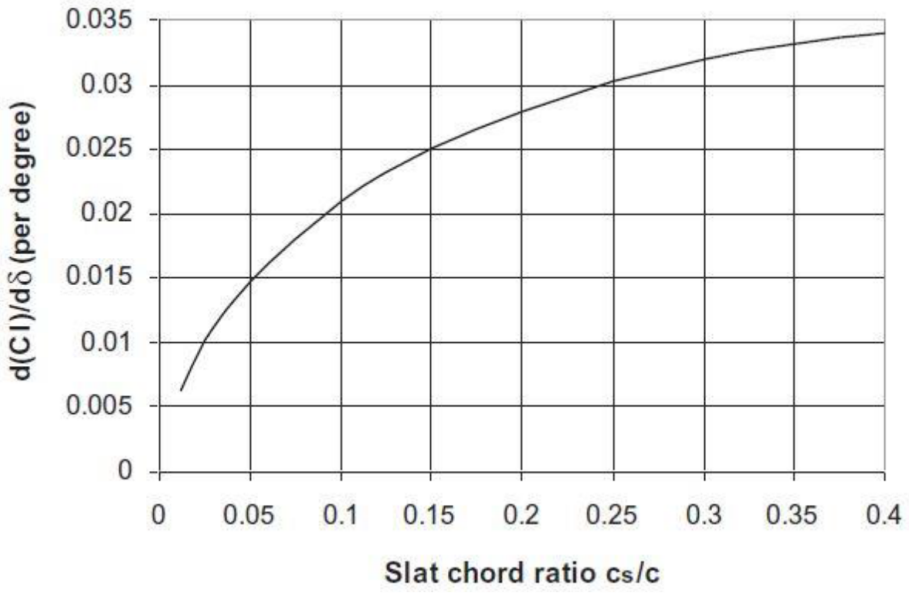

Figure 4.14 Rate of change of airfoil lift coefficient with slat deflection (per degree)

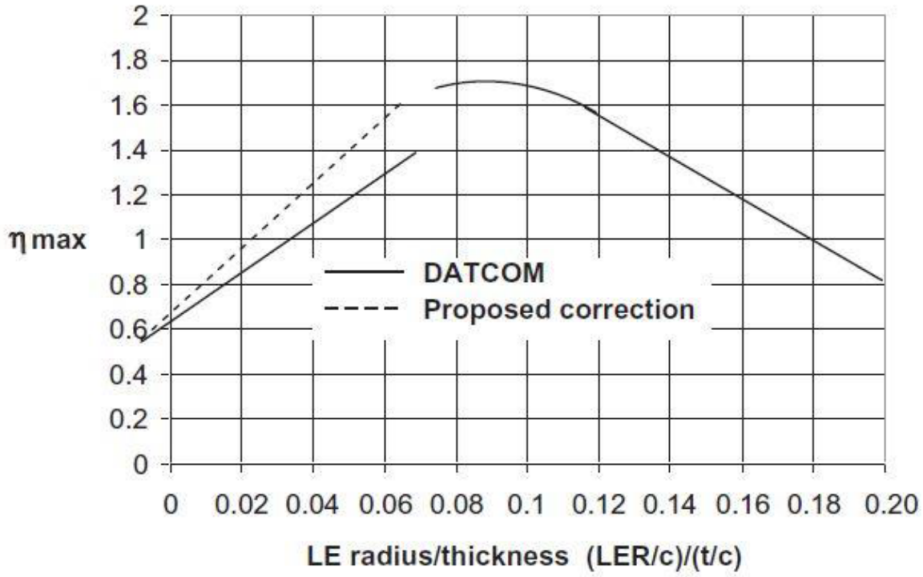

Figure 4.15 Correction factor for leading edge radius and airfoil thickness ratio

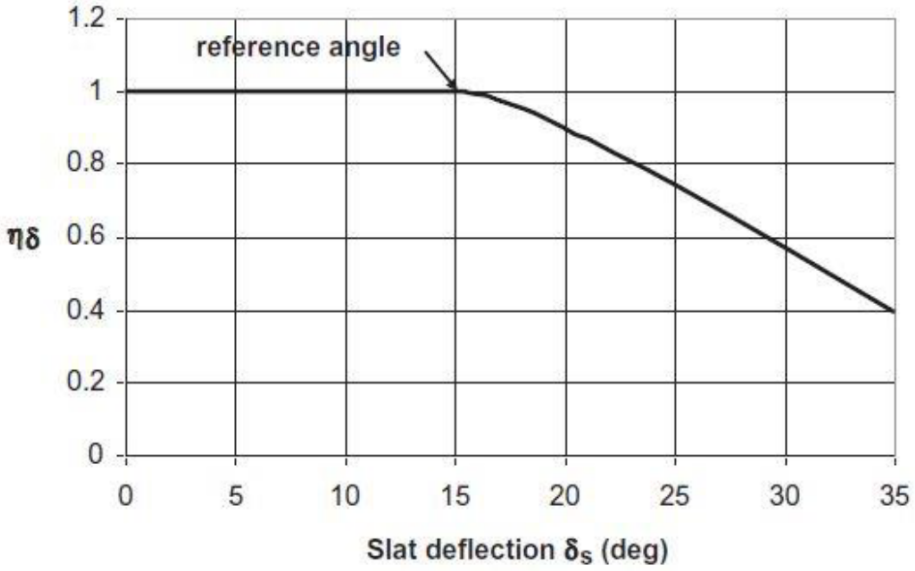

Figure 4.16 Slat deflection correction factor as a function of deflection angle

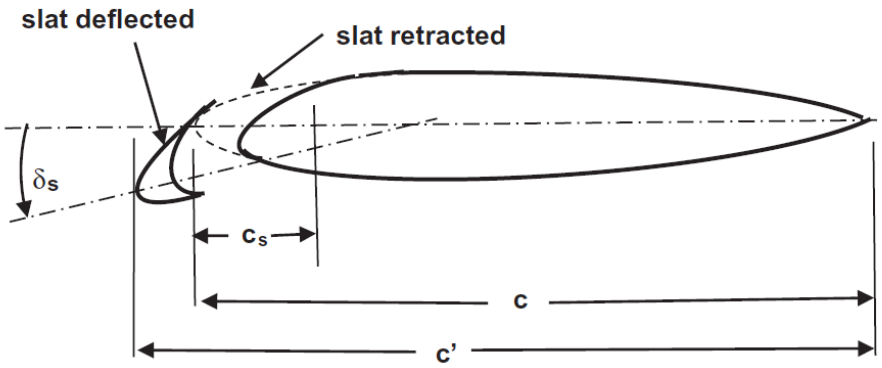

Figure 4.17 Geometry of the leading edge as used in the DATCOM method

Now that all increments related to the airfoil have been explained, the next step is to extend these two-dimensional quantities to the entire three-dimensional wing. Regarding trailing edge devices, the DATCOM method provides the following equation to compute the increment in maximum lift coefficient for the wing.

$$\Delta C_{L\max} = \Delta C_{l\max} \frac{S_{w,f}}{S} K_\Lambda \tag{4.11}$$

Here $\Delta C_{l\max}$ is the increment in lift coefficient due to flap deflection as defined in (4.9) while the quantity $\frac{S_{w,f}}{S}$ is the ratio of wing area affected by the trailing edge flap deflection (including both port and starboard wings) to the total wing area, as shown in figure 4.18; the wing area affected by the flap may be written as in (4.12).

$$S_{w,f} = \left(\frac{b}{2}\right) c_r \ [2 - (1 - \lambda) \ (\eta_i - \eta_o)] \ (\eta_i - \eta_o) \tag{4.12}$$

In the (4.12) the quantity $\lambda = \frac{c_t}{c_r}$ is the taper ratio, b is the wingspan while η_i and η_o are, respectively, the non-dimensional inboard location and the outboard location of the flap. If the flaps do not extend continuously along the trailing edge then the affected area for each may be calculated independently and added together.

Finally the correction factor for sweepback wings is given by the following equation.

$$K_\Lambda = \left(1 - 0.08 \ \cos^2 \Lambda_{c/4}\right) \cos^{3/4} \Lambda_{c/4} \tag{4.13}$$

The same calculations provided for trailing edge devices can be used for leading edge deivces as well with the difference that the $\Delta C_{l\max}$ has to be calculated with the (4.10).

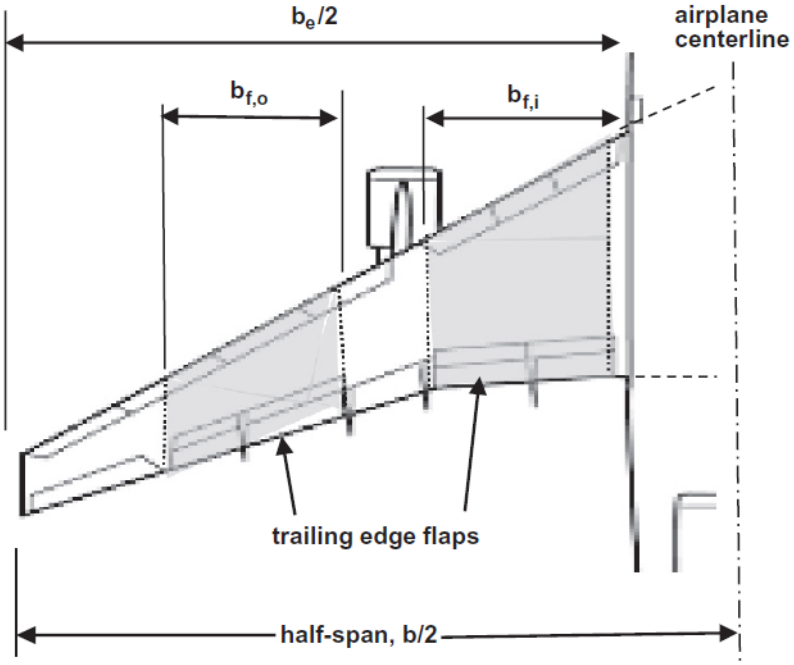

Figure 4.18 Plan view of wing where shaded area, including both the starboard and port wing, is the rated area for the flaps, that is, the portion of the wing planform area affected by the flaps

4.1.3 $C_{l\alpha,\text{flap}}$ and $C_{L\alpha,\text{flap}}$ calculation

The lift gradient is affected by flap deflection in three ways:

- The chord extension increases $C_{l\alpha}$. This effect depends on the multiplicative factor $\frac{c'}{c}$
- The potential flow effect of flap deflection on the lift curve slope is reduced with increasing α. The result is a nonlinearity in the lift curve which is particularly pronounced for large flap angles.
- The effect of viscosity on the lift effectiveness of a flap increases with the angle of attack, thus reducing $(\Delta C_l)_{\text{flap}}$ with increasing α

In order to evaluate the variation in $C_{l\alpha}$ due to flap deflection, references to [31] have been made. In particular, Torenbeek provides the following equation which approximates the results of the exact theory fairly accurately and is in qualitative agreement with experimental data.

$$C_{l\alpha_{(\text{flap down})}} = C_{l\alpha}\left[\frac{c'}{c}\left(1-\frac{c_f}{c'}\sin^2\delta_f\right)\right] \tag{4.14}$$

This equation is then corrected for the three-dimensional wing as follows.

$$C_{L\alpha_{(\text{flap down})}} = C_{L\alpha}\left\{1+\frac{\Delta C_{L0}}{\Delta C_{l0}}\left[\left(\frac{c'}{c}\left(1-\frac{c_f}{c'}\sin^2\delta_f\right)-1\right)\right]\right\} \tag{4.15}$$

4.1.4 $\Delta\alpha_{\text{stall}}$ calculation for trailing edge and leading edge devices

As explained previously, trailing edge devices contribute to lower the stalling angle of attack, while leading edge devices do the opposite providing an higher α_{stall}. In order to compute these two effects, a reference to [23] has been done. In particular, for trailing edge devices, data from [2] suggest the following curve where the $\Delta\alpha_{\text{stall}}$ is negative and is function of flap delflection.

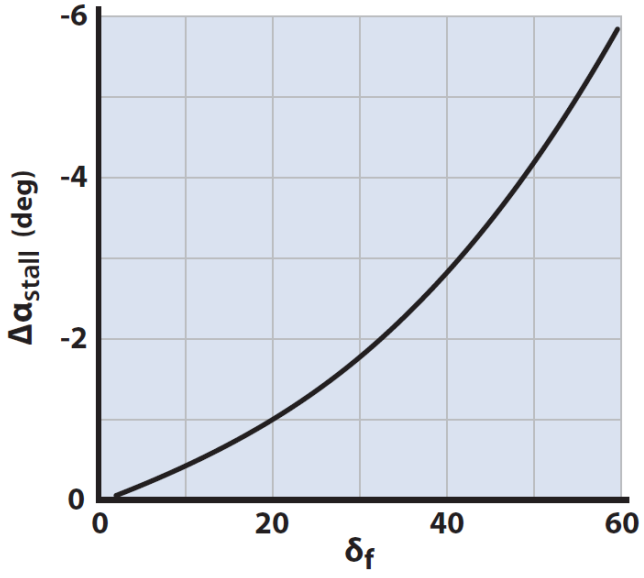

Figure 4.19 Decrease in stall angle with fl ap defl ection

Since there are no precise methods for predicting leading edge devices effects on α_{stall}, references to experimental data are the only way to follow. In order to build the lift curve with flaps and slats effects, this value of $\Delta\alpha_{\text{stall}}$ is taken as the same used in the construction of the clean wing lift curve; the latter shown in figure A.2.

4.1.5 Further effects calculations

High-lift devices influence not only the lift curve but also the wing drag polar, in particular the C_{D0}, and the wing pitching moment. Further calculations are, therefore, necessary to predict these latter.

Regarding the ΔC_{D0} provided by high-lift devices, the reader can refer to [33]. Proceeding on much the same lines as in the analysls of lift coefficient increments, Young and Hufton assumed that $\Delta C_D 0$ can be calculated as follow for a full-span flap.

$$\Delta C_{D0} = \delta_1 \left(c_f/c\right) \cdot \delta_2 \left(\delta_f\right) \tag{4.16}$$

where δ_1 and δ_2 are functions that were determined from experimental data. Their related curves are shown in figure 4.20 and 4.22, for plain flap, and, in figure 4.21 and 4.23, for slotted flaps.

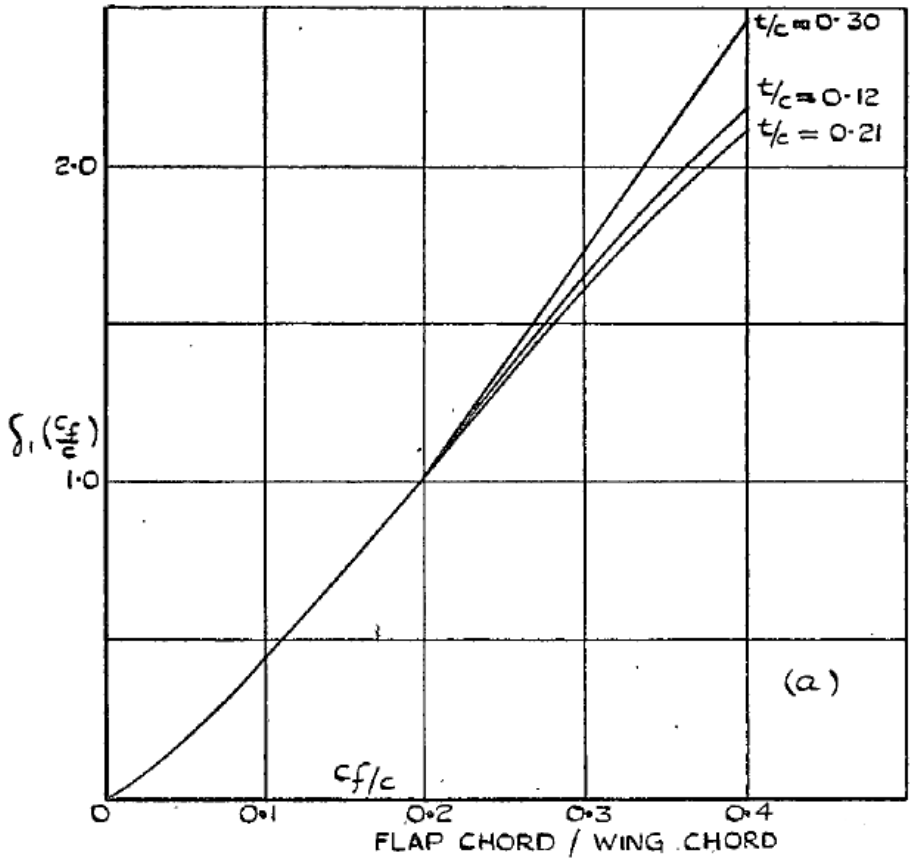

Figure 4.20 The functions $\delta_1 \left(c_f/c\right)$ for split and plain flaps

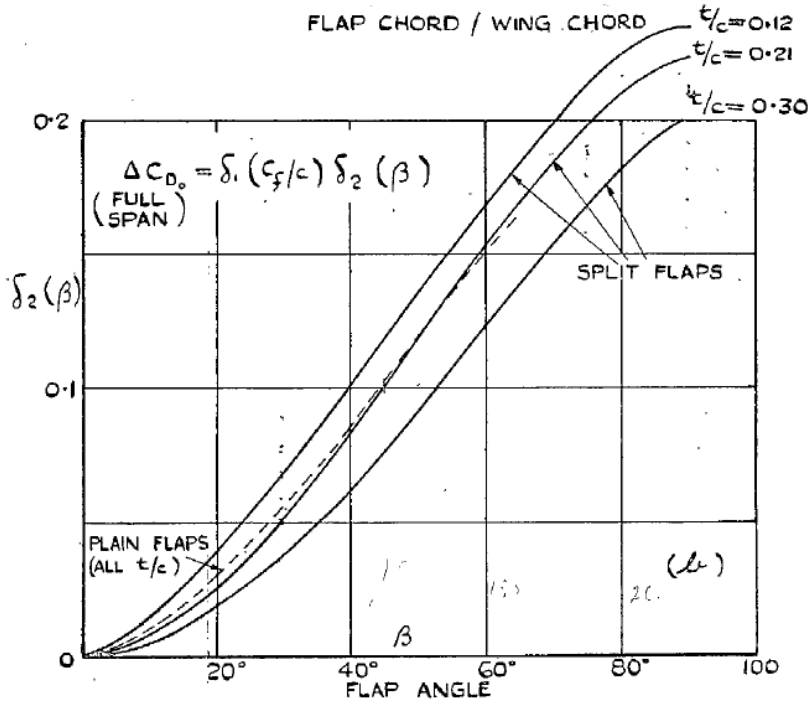

Figure 4.21 The functions $\delta_2\left(\delta_f\right)$ for plain flaps

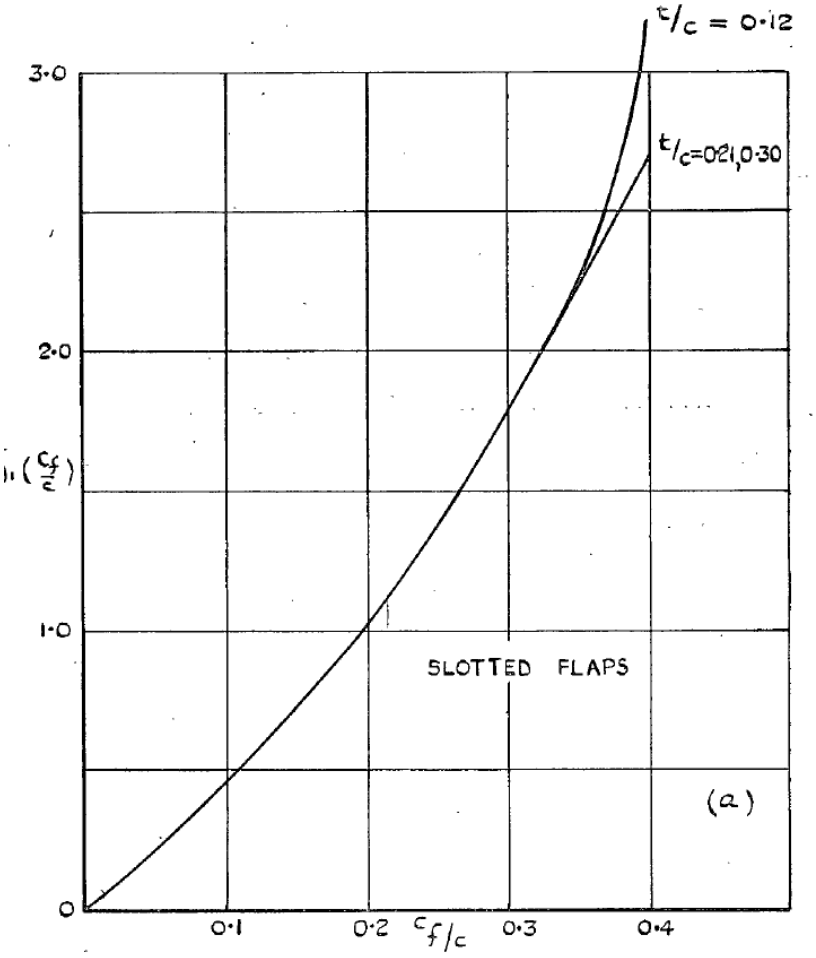

Figure 4.22 The functions $\delta_1\left(c_f/c\right)$ for split and slotted flaps

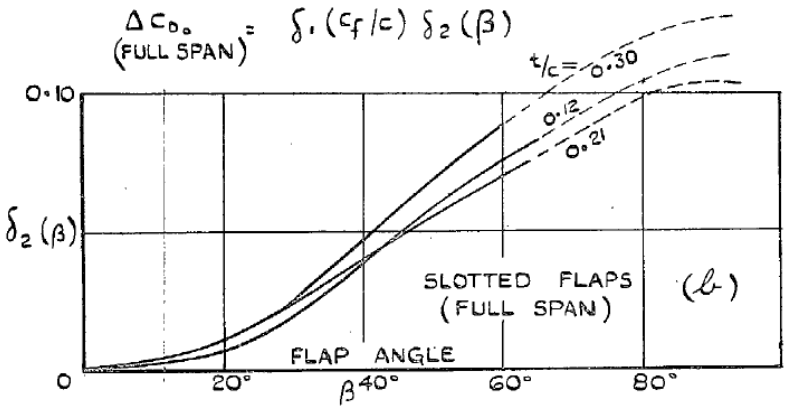

Figure 4.23 The functions $\delta_2\left(\delta_f\right)$ for slotted flaps

General considerations, confirmed by experimental data, led Young and Hufton to conclude that the drag increment of a part-span flap of any type is proportional to the area of the flapped part of the wing. Hence, to determine the increment for a part-span flap it's necessary to multiply the increment for a full-span flap by the ratio of the flapped wing area to the total wing area. The latter ratio, denoted by δ_3, is shown in figure 4.24 as a function of flap span for wings of various taper ratios. Thus the ΔC_{D0} equation becomes the following.

$$\Delta C_{D0} = \delta_1\left(c_f/c\right) \cdot \delta_2\left(\delta_f\right) \cdot \delta_3\left(b_f/b\right) \tag{4.17}$$

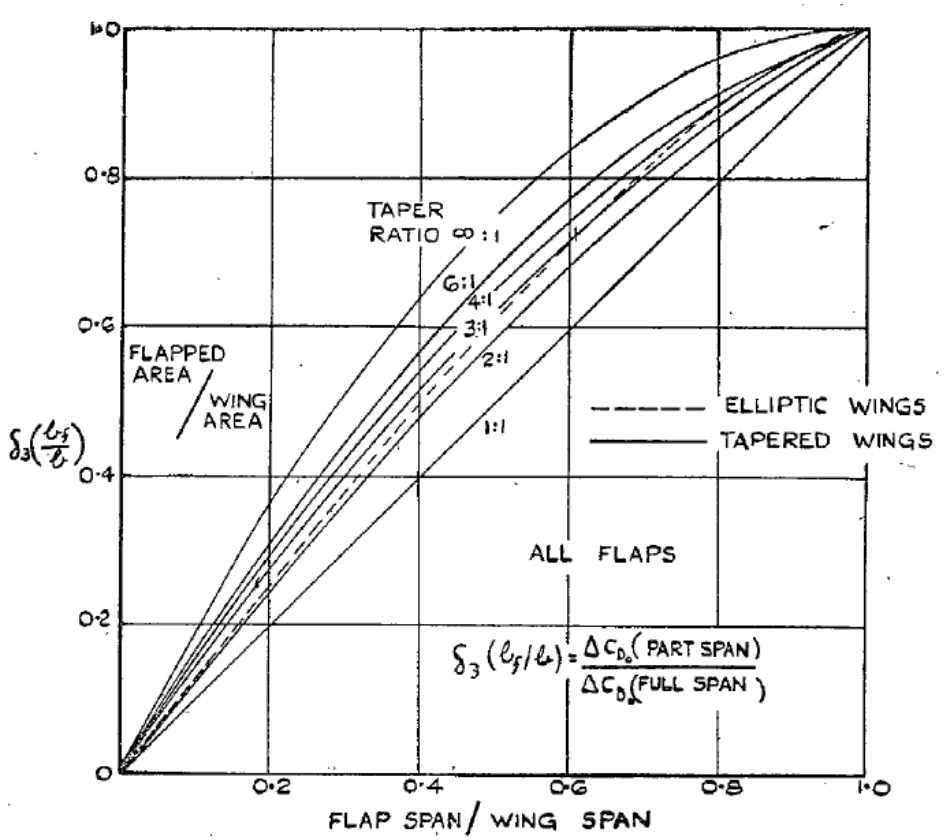

Figure 4.24 The functions $\delta_3\left(b_f/b\right)$ for slotted flaps

The last, but not least, effect to be predict refers to the pitching moment of the airfoil and of the wing. Following the guideline proposed in [31], the generalized expression, for airfoils, in (4.18) represents a useful starting point when experimental data are not available.

$$\Delta C_{m_{\frac{c}{4}}} = -\mu_1 \, \Delta C_{l\max} \left(\frac{c'}{c}\right)^2 - \frac{C_l}{4} \frac{c'}{c} \left(\frac{c'}{c} - 1\right) + \left(C_{m_{\frac{c}{4}}}\right)_{\delta_f = 0} \left[\left(\frac{c'}{c}\right)^2 - 1\right] \tag{4.18}$$

where extended chord c', shown in figure 4.7, allows for the effects of the backward movement of the flap while it is being extended. In equation this equation, the first contribution is due to the increased section camber and the factor μ_1 is defined as follows according to Glauert's linear theory for small flap deflections.

$$\mu_1 = \frac{1}{2}\left(1 - \frac{c_f}{c}\right) \frac{\sin\theta_f}{\pi - \left(\theta_f - \sin\theta_f\right)} \tag{4.19}$$

where θ_f is the one from equation (4.2). This theoretical value of μ_1 generally underpredicts the pitching moment coefficient; in fact it has been found that, for slotted flaps with, or without, Fowler movement, most data are on a single line, provided the second term of the equation (4.19), representing the theoretical rearward shift of the airfoil aerodynamic center, is halved. Furthermore, in the case of split and plain flaps, the flap angle is observed to exert a pronounced influence on μ_1 as shown in figure 4.25. The last term of the (4.19) is generally of a low order and can be ignored. This, together with the halving of the second term of the previous equation, leads to the following practical expression of $\Delta C_{m_{\frac{c}{4}}}$.

$$\Delta C_{m_{\frac{c}{4}}} = -\mu_1 \, \Delta C_{l\max} \left(\frac{c'}{c}\right) - \frac{C_l}{8} \frac{c'}{c} \left(\frac{c'}{c} - 1\right) \tag{4.20}$$

The obtained two-dimensional equation can, then, be converted into a three-dimensional one,

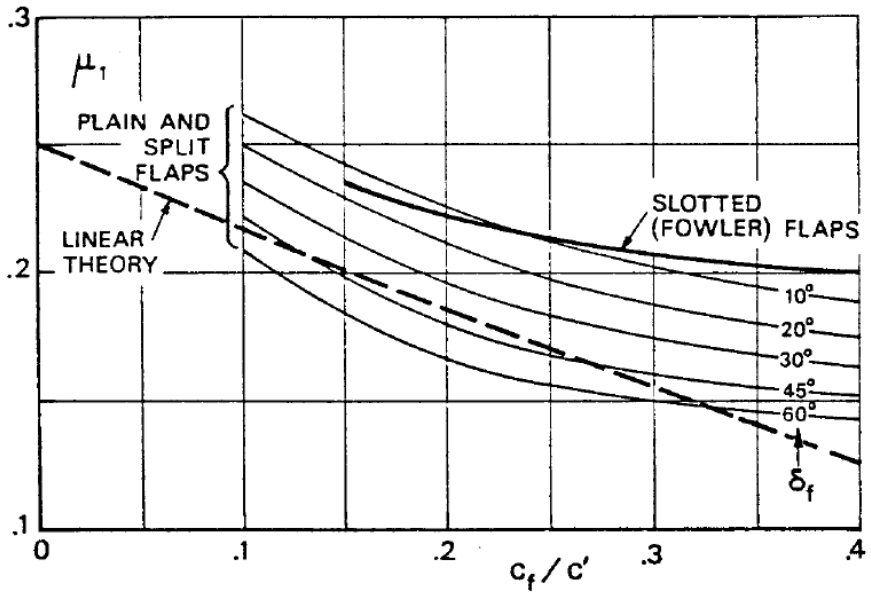

Figure 4.25 The pitching moment function μ_1

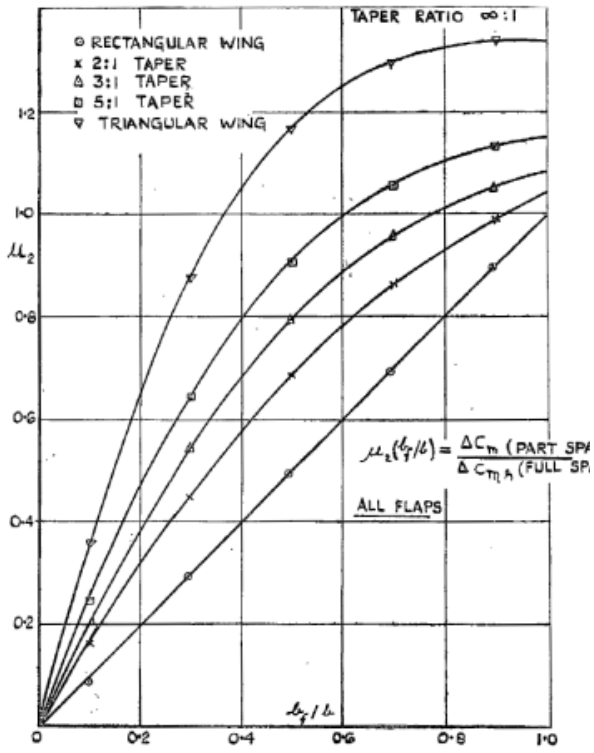

Figure 4.26 The pitching moment function μ_2

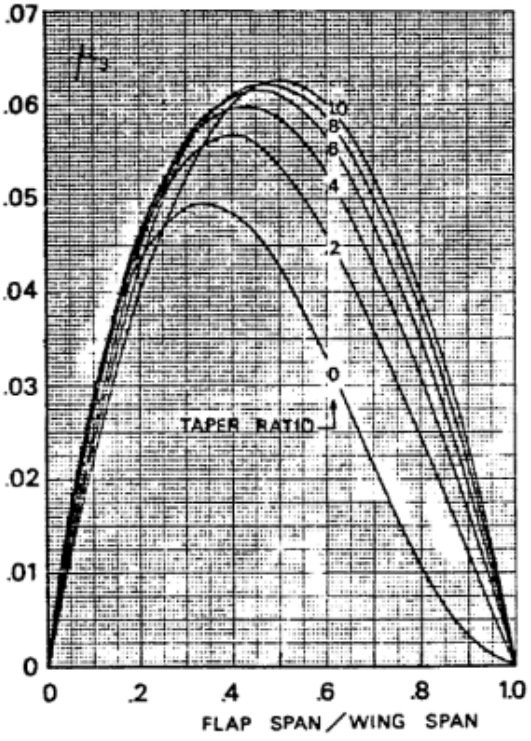

Figure 4.27 The pitching moment function μ_3

which computes the pitching moment change on the entire wing, as follows.

$$\Delta C_{M_{\frac{c}{4}}} = \mu_2 \, \Delta C_{m_{\frac{c}{4}}} + 0.7 \, \frac{\mathcal{AR}}{1 + \frac{2}{\mathcal{AR}}} \, \mu_3 \, \Delta C_{l_{\max}} \, \tan \Lambda_{\frac{c}{4}} \tag{4.21}$$

where $\Delta C_{m_{\frac{c}{4}}}$ is the one from (4.20), provide that $C_l = C_L + \Delta C_{l\max} \left(1 - \frac{S_{w,f}}{S}\right)$, and the correction factors μ_2 and μ_3 are the one shown in figures 4.26 and 4.27. Making the appropriate substitutions, the following operational formula is, finally, obtained.

$$\begin{aligned} \Delta C_{M_{\frac{c}{4}}} = \mu_2 \, & \left\{ -\mu_1 \, \Delta C_{L\max} \, \frac{c'}{c} - \left[C_L + \Delta C_{l\max} \left(1 - \frac{S_{w,f}}{S}\right) \right] \frac{1}{8} \, \frac{c'}{c} \left(\frac{c'}{c} - 1 \right) \right\} \\ & + 0.7 \, \frac{\mathcal{AR}}{1 + \frac{\mathcal{AR}}{2}} \, \mu_3 \, \Delta C_{l\max} \, \tan \Lambda_{\frac{c}{4}} \end{aligned} \tag{4.22}$$

4.2 Java class architecture

The main purpose of this paragraph is to provide a comprehensive description of how all semi-empirical methods, presented previously, are implemented and used inside the Java class created to manage high-lift devices effects upon wings. Because of the great number of charts from which some important data have to be derived, an external database, containing all the digitized curves shown in previous paragraph figures, has been created following the procedure explained in appendix A.1.

This database, named `HighLiftDatabase`, is then read by a dedicated class created inside the database package of `JPADCore` which name is `HighLiftDatabaseReader`; the latter, in particular,

Available Flaps	Typical maximum deflection	Database index
Single slotted	45°	1.0
Double slotted	50°	2.0
Plain	60°	3.0
Fowler	40°	4.0
Triple slotted	50°	5.0

Table 4.3 Summary of available flaps types with thier related properties

has been built up following the giudeline of appendix A.2.

Having now a powerful tool to bypass the charts reading, it's now possible to create the Java class dedicated to the management of high-lift devices effects. Since this class, named `CalcHighLiftDevices`, has to evaluate these effects on a wing, the philosophy followed in its creation suggests to relate it to the class in charge of all analysis upon a lifting surface; in particular, this class is named `LSAerodynamicsManager` and `CalcHighLiftDevices` is an inner class of it.

The first component of `CalcHighLiftDevices` that has to analyzed is the constructor; the latter, used whenever an object of this class has to be created, has three main purposes:

1. Assign user's inputs to their related class fields
2. Recognize flaps types from a List of `FlapTypeEnum` objects and link them to their typical maximum deflection value; moreover it gives them an appropriate index to be used inside `HighLiftDatabaseReader` in order to access to the specific flap type curve of the database when needed
3. Perform the preliminary calculation of the wing clean maximum lift coefficient, and of the related stalling angle of attack, using the method named `calcAlphaAndCLMax`, of the `LSAerodynamicsManager`, which accepts as input the wing mean airfoil calculated by the constructor using the method `calculateMeanAirfoil`, of a `LSAerodynamicsManager` inner class named `MeanAirfoil`, which performs the calculation of the required characteristics using influence areas method based on the three main wing airfoils (Root, Kink, Tip).

In particular `FlapTypeEnum` is an Enumeration[8] of possible flaps types created inside the `JPADConfigs` package to manage in a smart and intuitive way this parameter. Furthermore it has to be noted that having the clean maximum lift coefficient, and its stalling angle of attack, calculated in the constructor allows to reduce computational cost because these heavy calculations are performed only once when the object is created; this is very useful when the user wants to calculate the lift coefficient, at a specific angle of attack on the flapped lift curve, many time, for example, during iterations.

`CalcHighLiftDevices` class is also made up of three main methods each of which in charge of a specific task. The first one is `calculateHighLiftDevicesEffects` which implements all formulas explained in paragraph 4.1 using input data derived from the constructor, and shown in table 4.4. This method is defined `void` so that it returns nothing as output; in fact, the latter

`theWing`	A `LiftingSurface` object representing an aircraft wing
`theConditons`	An `OperatingConditions` object representing aircraft flight conditions
`deltaFlap`	A `List` of `Double` arrays containing each flap deflections
`flapType`	A `List` of `FlapTypeEnum` values for each flap
`deltaSlat`	A `List` of `Double` values containing each slat deflections; to be set to null if there aren't slats
`etaInFlap`	A `List` of `Double` values containing each flap inner adimensional position
`etaOutFlap`	A `List` of `Double` values containing each flap outer adimensional position
`etaInSlat`	A `List` of `Double` values containing each slat inner adimensional position, to be set to null if there aren't slats
`etaInSlat`	A `List` of `Double` values containing each flap outer adimensional position, to be set to null if there aren't slats
`cfc`	A `List` of `Double` values containing flap chord to airfoil chord ratios for each flap
`csc`	A `List` of `Double` values containing slat chord to airfoil chord ratios for each slat, to be set to null if there aren't slats
`leRadiusRatioSlat`	A `List` of `Double` values containing Leading Edge Radius (LER) to airfoil thickness ratios ($\mathrm{LER}/t = \mathrm{LER}/c \cdot t/c$) for each slat, to be set to null if there aren't slats
`cExtcSlat`	A `List` of `Double` arrays containing extended chord to airfoil chord ratios for each slat

Table 4.4 `CalcHighLiftDevices` constructor input

has only to provide the required calculations leaving to the user the output data retrieval using the appropriate *getter* methods of which the class is provided.

The second one is `calcCLatAlphaHighLiftDevice`, which requires firstly to perform the analysis of `calculateHighLiftDevicesEffects`; it calculates the lift coefficent at a given angle of attack for a wing with high-lift devices deflected. This method calculates both linear trait and non-linear trait of the curve using the NASA Blackwell method for the slope of the linear trait and a cubic interpolation for the non-linear one. In particular it sums up the effect of ΔC_{L0} to the clean wing C_{L0} in order to obtain the new value with deflected high-lift devices, then it replaces the clean wing $C_{L\alpha}$ with the new one calculated with the previous method; assuming also that the angle of attack at the end of the linear trait is taken as the mean of the old one, realted to the mean airfoil, and the one calculated at the same C_L^* but on the new curve, the new C_L^* can be evaluated completing the calculations of the linear trait parameters. From this point on, $\Delta C_{L\max}$ and $\Delta\alpha_{\text{stall}}$ are summed to the clean curve $C_{L\max}$ and α_{stall} calculated into the constructor; then a cubic interpolation, starting from the end of linear trait, allows to build the non-linear one. The method accept as input an angle of attack which is then compared with α^* calculated previously for the flapped curve; if this is lower the methods calculates the required lift coefficient using the linear trait, otherwise it uses the non-linear one.

The third, and last, method is `plotHighLiftCurve`. It is used to plot both the clean wing lift curve, both the flapped one; the first task is done by an `LSAerodynamicsManager` method named `PlotCLvsAlphaCurve`, while the second one follows the same philosophy of the previous method to calculate the lift coefficent for 30 angles of attack starting from -10° and ending at $(\alpha_{stall})_{flap}$+2°.

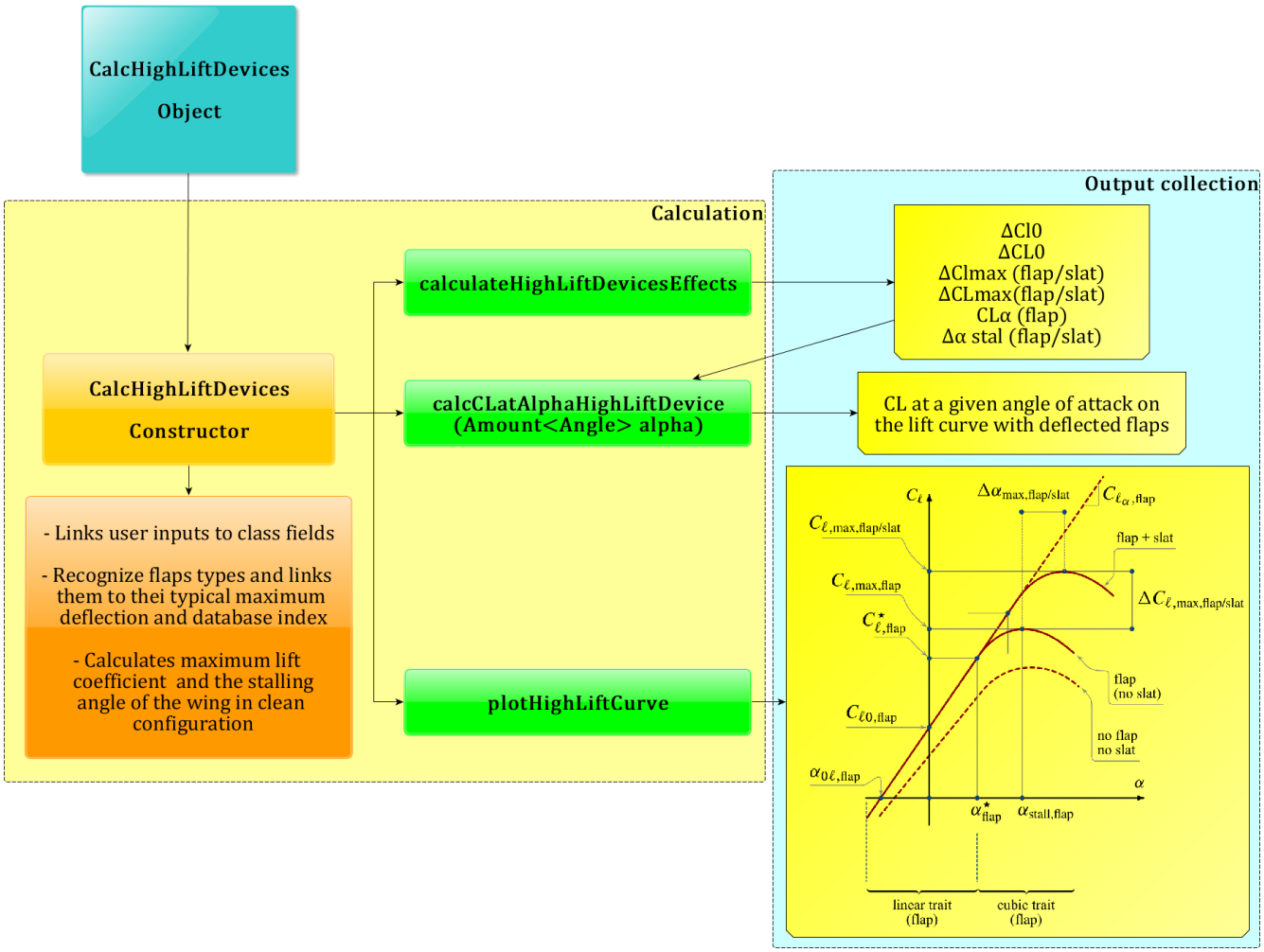

Figure 4.28 `CalcHighLiftDevices` class flowchart

4.3 Case study: ATR-72 and B747-100B

The last task to be done is to show an application of the `CalcHighLiftDevices` class in order to validate the calculation performed as well as to give a useful example to a potential user developer. As in previous chapters the test will be performed using ATR-72 and B747-100B aircraft models, built up as explained in paragraph 1.4. These aircraft models dosen't have high-lift devices input data embedded, so that they have to be assigned from an external XML file; this choice has been taken because the user may want to try different high-lift devices setups without changing the aircraft model on which these latter will be applied. The input XML file[6] has to provide all data from 4.4, except for the lifting surface model and the operating condition, and it is modeled as follows.

```
<input>
      <B747>
            <Flap_Number unit="">2</Flap_Number>
            <Flap1>
                  <FlapType unit="">DOUBLE_SLOTTED</FlapType>
                  <Cf_C unit="">0.1264</Cf_C>
                  <Delta_Flap1 unit="deg">15</Delta_Flap1>
                  <Delta_Flap1 unit="deg">5</Delta_Flap1>
                  <Flap_inboard>0.0995</Flap_inboard>
                  <Flap_outboard>0.3614</Flap_outboard>
            </Flap1>
            <Flap2>
                  <FlapType unit="">DOUBLE_SLOTTED</FlapType>
                  <Cf_C unit="">0.2806</Cf_C>
                  <Delta_Flap2 unit="deg">15</Delta_Flap2>
                  <Delta_Flap2 unit="deg">5</Delta_Flap2>
                  <Flap_inboard unit="">0.4123</Flap_inboard>
                  <Flap_outboard unit="">0.6624</Flap_outboard>
            </Flap2>
            <Slat1>
                  <Delta_Slat unit="deg">15</Delta_Slat>
                  <Cs_C unit="">0.15</Cs_C>
                  <cExt_c unit="">1.1</cExt_c>
                  <LEradius_c_ratio unit="">0.0097</LEradius_c_ratio>
                  <Slat_inboard unit="">0.14</Slat_inboard>
                  <Slat_outboard unit="">0.37</Slat_outboard>
            </Slat1>
            <Slat2>
                  <Delta_Slat unit="deg">15</Delta_Slat>
                  <Cs_C unit="">0.193</Cs_C>
                  <cExt_c unit="">1.1</cExt_c>
                  <LEradius_c_ratio unit="">0.0097</LEradius_c_ratio>
                  <Slat_inboard unit="">0.42</Slat_inboard>
                  <Slat_outboard unit="">0.9</Slat_outboard>
            </Slat2>
      </B747>
</input>
```

Listing 4.1 Example of high-lift devices XML input file for the B747-100B in take-off configuration

```
<input>
      <ATR72>
            <Flap_Number unit="">2</Flap_Number>
            <Flap1>
                  <FlapType unit="">SINGLE_SLOTTED</FlapType>
                  <Cf_c unit="">0.35</Cf_c>
                  <Delta_Flap1 unit="deg">20</Delta_Flap1>
                  <Flap_inboard>0.0884</Flap_inboard>
                  <Flap_outboard>0.34575</Flap_outboard>
            </Flap1>
            <Flap2>
                  <FlapType unit="">SINGLE_SLOTTED</FlapType>
```

```
                <Cf_c unit="">0.35</Cf_c>
                <Delta_Flap2 unit="deg">20</Delta_Flap2>
                <Flap_inboard>0.34575</Flap_inboard>
                <Flap_outboard>0.82063</Flap_outboard>
            </Flap2>
        </ATR72>
</input>
```

Listing 4.2 Example of high-lift devices XML input file for the ATR-72 in take-off configuration

The first thing to do in order to read these data is use the Java library `args4j`; the latter allows the user to make the parsing of the input file in order to make it available for reading inside the test class. For more information regarding this library the reader can consult [20]. After the input file has been parsed correctly, the reading process can start. The user now has to create a List for each input data of table 4.4 and then an object of the class `JPADXmlReader`, of the `JPADCore standaloneutils` package, in order to manage the scan of the XML file through its methods.

```
// High Lift Devices Input
List<Double[]> deltaFlap = new ArrayList<Double[]>();
List<FlapTypeEnum> flapType = new ArrayList<FlapTypeEnum>();
List<Double> etaInFlap = new ArrayList<Double>();
List<Double> etaOutFlap = new ArrayList<Double>();
List<Double> cfc = new ArrayList<Double>();
List<Double> deltaSlat = new ArrayList<Double>();
List<Double> etaInSlat = new ArrayList<Double>();
List<Double> etaOutSlat = new ArrayList<Double>();
List<Double> csc = new ArrayList<Double>();
List<Double> cExtcSlat = new ArrayList<Double>();
List<Double> leRadiusSlatRatio = new ArrayList<Double>();
// XML reading phase:
// Arguments check
if (args.length == 0){
  System.err.println("NO INPUT FILE GIVEN --> TERMINATING");
  return;}
// Input file parsing
main.theCmdLineParser.parseArgument(args);
// Creation of the reader object
String path = main.get_inputFile().getAbsolutePath();
JPADXmlReader reader = new JPADXmlReader(path);
// XML file scan through JPADXmlReader methods
List<String> flapNumberProperty = reader.getXMLPropertiesByPath("//Flap_Number");
int flapNumber = Integer.valueOf(flapNumberProperty.get(0));
List<String> flapTypeProperty = reader.getXMLPropertiesByPath("//FlapType");
List<String> cfcProperty = reader.getXMLPropertiesByPath("//Cf_C");
List<String> deltaFlap1Property = reader.getXMLPropertiesByPath("//Delta_Flap1");
List<String> deltaFlap2Property = reader.getXMLPropertiesByPath("//Delta_Flap2");
List<String> etaInFlaProperty = reader.getXMLPropertiesByPath("//Flap_inboard");
List<String> etaOutFlapProperty =
        reader.getXMLPropertiesByPath("//Flap_outboard");
List<String> deltaSlatProperty = reader.getXMLPropertiesByPath("//Delta_Slat");
```

```
List<String> cscProperty = reader.getXMLPropertiesByPath("//Cs_C");
List<String> cExtcSlatProperty = reader.getXMLPropertiesByPath("//cExt_c");
List<String> leRadiusSlatRatioProperty =
        reader.getXMLPropertiesByPath("//LEradius_c_ratio");
List<String> etaInSlatProperty = reader.getXMLPropertiesByPath("//Slat_inboard");
List<String> etaOutSlatProperty =
        reader.getXMLPropertiesByPath("//Slat_outboard");
// Management of the Lists of String in order to populate the previous Lists
// Recognizing flap type
for(int i=0; i<flapTypeProperty.size(); i++) {
  if(flapTypeProperty.get(i).equals("SINGLE_SLOTTED"))
    flapType.add(FlapTypeEnum.SINGLE_SLOTTED);
  else if(flapTypeProperty.get(i).equals("DOUBLE_SLOTTED"))
    flapType.add(FlapTypeEnum.DOUBLE_SLOTTED);
  else if(flapTypeProperty.get(i).equals("PLAIN"))
    flapType.add(FlapTypeEnum.PLAIN);
  else if(flapTypeProperty.get(i).equals("FOWLER"))
    flapType.add(FlapTypeEnum.FOWLER);
  else if(flapTypeProperty.get(i).equals("TRIPLE_SLOTTED"))
    flapType.add(FlapTypeEnum.TRIPLE_SLOTTED);
  else {
    System.err.println("NO VALID FLAP TYPE!!");
    return;
  }
}
Double[] deltaFlap1Array = new Double[delta_flap1_property.size()];
for(int i=0; i<deltaFlap1Array.length; i++)
  deltaFlap1Array[i] = Double.valueOf(deltaFlap1Property.get(i));
  Double[] deltaFlap2Array = new Double[deltaFlap2Property.size()];
for(int i=0; i<deltaFlap1Array.length; i++)
  deltaFlap2Array[i] = Double.valueOf(deltaFlap2Property.get(i));
  deltaFlap.add(deltaFlap1Array);
deltaFlap.add(deltaFlap2Array);
  for(int i=0; i<cfcProperty.size(); i++)
  cfc.add(Double.valueOf(cfcProperty.get(i)));
for(int i=0; i<etaInFlapProperty.size(); i++)
  etaInFlap.add(Double.valueOf(etaInFlapProperty.get(i)));
for(int i=0; i<etaOutFlapProperty.size(); i++)
  etaOutFlap.add(Double.valueOf(etaOutFlapProperty.get(i)));
for(int i=0; i<deltaSlatProperty.size(); i++)
  deltaSlat.add(Double.valueOf(deltaSlatProperty.get(i)));
for(int i=0; i<cscProperty.size(); i++)
  csc.add(Double.valueOf(cscProperty.get(i)));
for(int i=0; i<cExtcSlatProperty.size(); i++)
  cExtcSlat.add(Double.valueOf(cExtcSlatProperty.get(i)));
for(int i=0; i<leRadiusSlatRatioProperty.size(); i++)
  leRadiusSlatRatio.add(Double.valueOf(leRadiusSlatRatioProperty.get(i)));
for(int i=0; i<etaInSlatProperty.size(); i++)
  etaInSlat.add(Double.valueOf(etaInSlatProperty.get(i)));
for(int i=0; i<etaOutSlatProperty.size(); i++)
  etaOutSlat.add(Double.valueOf(etaOutSlatProperty.get(i)));
```

Listing 4.3 Excerpt of B747-100B test - Input data reading

It's important to highlight that `JPADXmlReader` methods can return an `Amount`[10] or a `String` for each data read this way so that a post-reading management is necessary in order to have data in correct types.

At this point all the required data are ready for use; now the user just have to create the object of the class `CalcHighLiftDevices` and call its methods as described in figure 4.28. It has to be noted that, since `CalcHighLiftDevices` is a class nested into `LSAerodynamicsManager`, the creation of the required object is a little bit different from the usual; in particular an object of `LSAerodynamicsManager`, named `theLSAnalysis` in the example below, has to be created first and then the `CalcHighLiftDevices` one can be created as follows.

```
1  LSAerodynamicsManager.CalcHighLiftDevices highLiftCalculator = theLSAnalysis
2          .new CalcHighLiftDevices(
3              aircraft.get_wing(),
4              theCondition,
5              deltaFlap,
6              flapType,
7              deltaSlat,
8              eta_in_flap,
9              eta_out_flap,
10             eta_in_slat,
11             eta_out_slat,
12             cf_c,
13             cs_c,
14             leRadius_c_slat,
15             cExt_c_slat
16             );
```

Listing 4.4 Excerpt of B747-100B test - calculator object creation

In conclusion, the following pages show the numerical and graphical results of the described tests, applied to the two aircraft, mentioned at the beginning of paragraph, in take-off configuration.

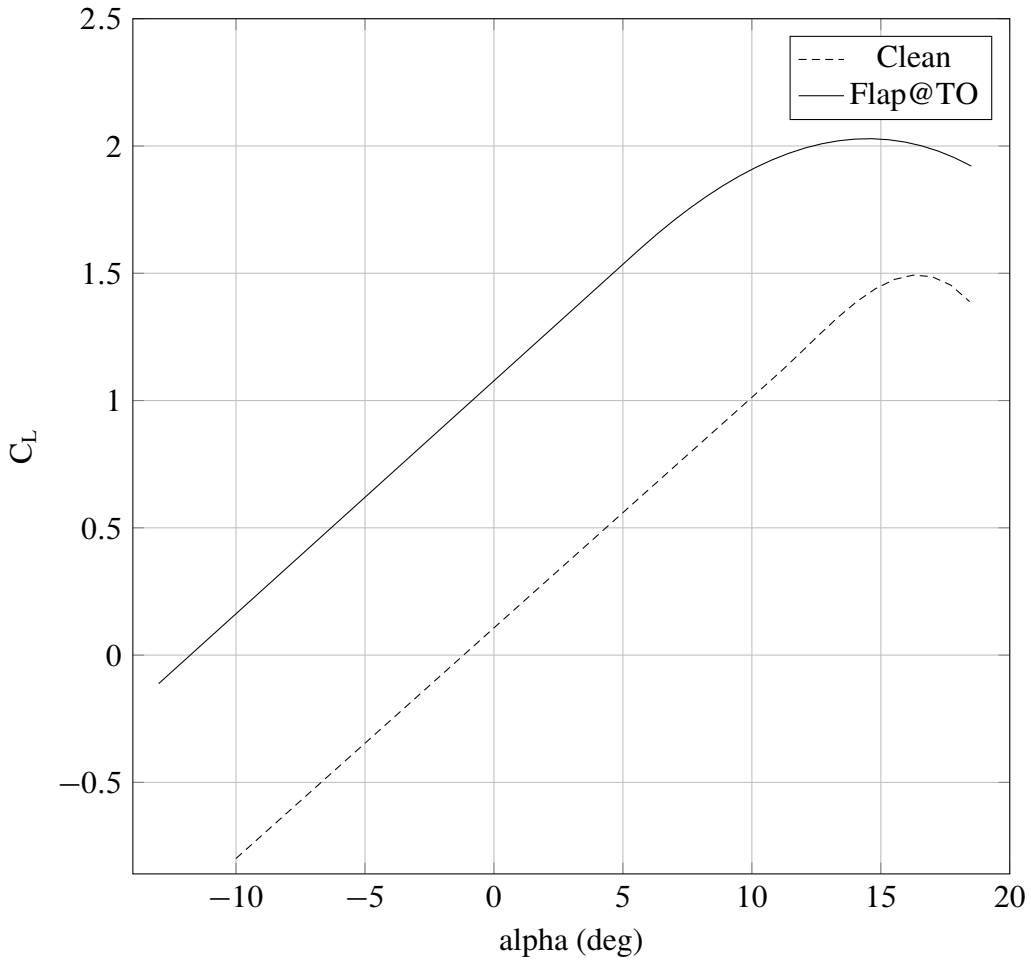

Figure 4.29 ATR-72 lift curve with and without flaps deflected

```
deltaCl0_flap = 2.3939974237661317
deltaCL0_flap = 0.9707980864918362
deltaClmax_flap = 1.2122379635109801
deltaCLmax_flap = 0.5352640320770878
cLalpha_new = 0.09150912423296985 (1/deg)
deltaAlphaMax = -1.9369466185196893 (deg)
deltaCD = 0.01518722606307197
deltaCMc_4 = -0.12338582743821447

 -----------CLEAN--------------
 alpha max 16.44529532754857 (deg)
 alpha star 10.918671750781748 (deg)
 cL max 1.4934954072726092
 cL star 1.0956477082461238
 cL alpha 5.189644003702159 (1/rad)
 -----------HIGH LIFT--------------
 alpha max 14.50713741489647 (deg)
 alpha star 5.55824828132392 (deg)
 cl max 2.028759439349697
 cl star 1.5861753654540678
 cl alpha 5.243472818549172 (1/rad)
```

Listing 4.5 ATR-72 test results

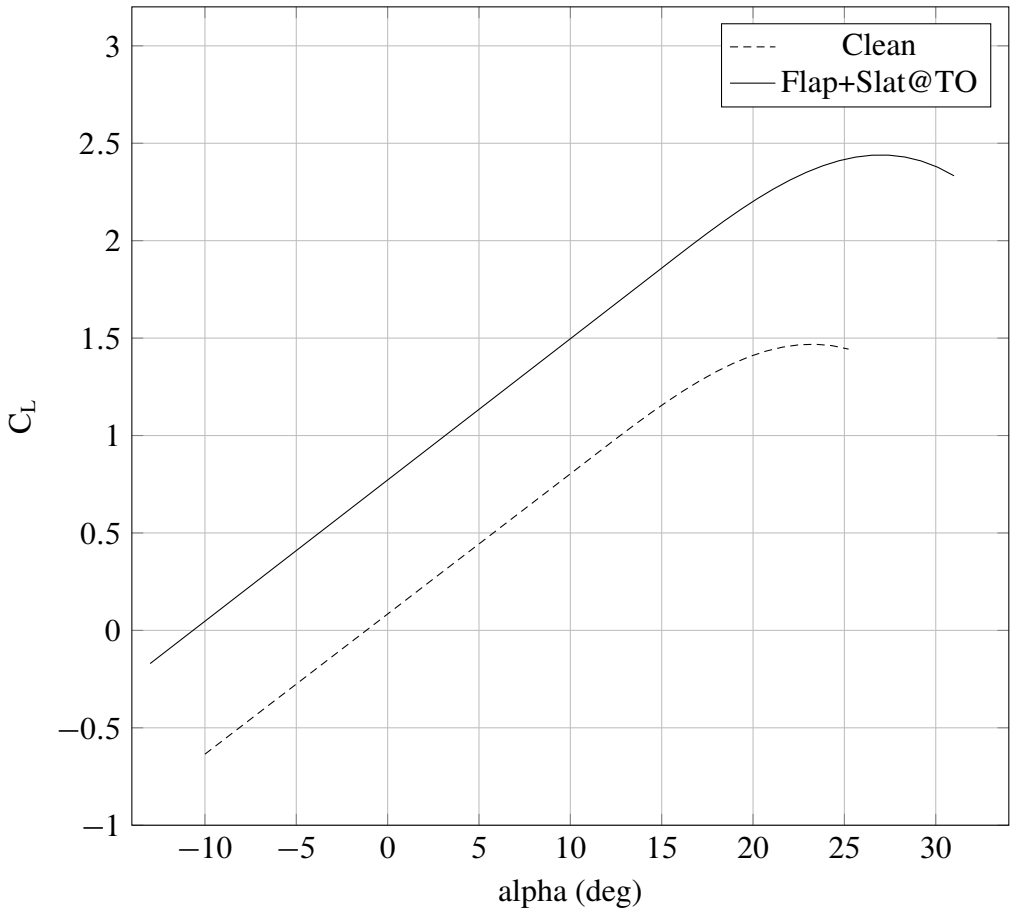

Figure 4.30 B747-100B lift curve with and without flaps and slats deflected

```
deltaCl0_flap = 2.275544332733731
deltaCL0_flap = 0.687810914469225
deltaClmax_flap = 0.6867057564174156
deltaCLmax_flap = 0.24074371285515445
deltaClmax_slat = 1.3986165535139938
deltaCLmax_slat = 0.7307256446849437
cLalpha_new = 0.07247171535879046 (1/deg)
deltaAlphaMaxFlap = -1.9369466185196893 (deg)
deltaCD = 0.006456897617521261

 -----------CLEAN--------------
 alpha max 23.232964070877248 (deg)
 alpha star 10.32038751072542 (deg)
 cL max 1.4686492856382116
 cL star 0.8263686942120971
 cL alpha 4.121408852511741 (1/rad)
 -----------HIGH LIFT--------------
 alpha max 27.00200178515614 (deg)
 alpha star 14.091414230062869 (deg)
 cl max 2.4401186431783097
 cl star 1.7930955427485755
 cl alpha 4.152629290058694 (1/rad)
```

Listing 4.6 B747-100B test results

Chapter 5

Take-off and Landing performance

When everything seems to be going against you,
remember that the airplane takes off against the wind, not with it.
– Henry Ford

Although the take-off field length may seem like a performance characteristic of secondary importance, it is very often one of the critical design constraints. If the required runway length is too long, the aircraft cannot take-off with full fuel or full payload and its economics are compromised. So take-off performance play a significant role in both the conceptual and the preliminary design phases of an aircraft because it is both design requirement, specified by the Federal Aviation Regulations (FAR) and by the customer, to be fulfilled, both a driving parameter in the definition of the design point. If the take-off performance is important, as it provides a limitation both in the wing loading $\frac{W}{S}$, both in the $\frac{T}{W}$ (or the $\frac{W}{P}$) ratio, also the landing performance is crucial as it imposes another limitation on the wing loading inside the design point definition.

5.1 Theoretical background

The take-off may be considered as made up of two parts: a ground run and an air run, as shown schematically in figure 5.1. The simplest description of the take-off process is that the engine thrust is increased to the take-off level at x = 0 and the brakes are released to begin acceleration down the runway. At some point, the pilot commands rotation of the aircraft which lifts the nose wheel from the ground and allows to achieve the take-off angle of attack; in this way the aircraft lift can grows faster and, when it is equal to the aircraft maximum take-off weight, the aircraft can lifts completely from the ground and begins climbing. The point at which it reaches an altitude of 35 ft (10.7 m), is considered, for an aircraft which refers to the FAR-25, the end of the take-off run. This is the usual situation for take-off; subsequently, the modifications to safely deal with a take-off emergency, such as an engine failure, will be discussed.

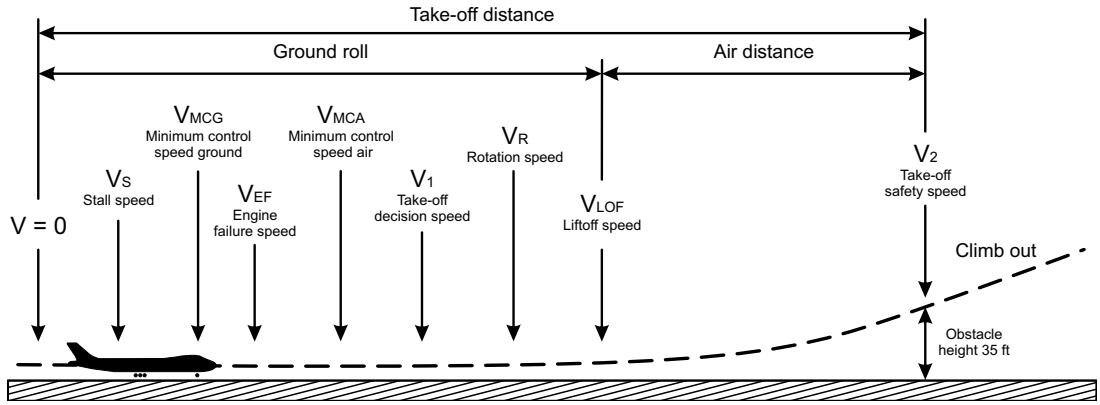

Figure 5.1 Scheme of an aircraft take-off run

5.1.1 AOE take-off run

In order to deal with the calculation of the take-off run distance, a smart strategy is to find out all the foundamentals variables, which describes completely the aircraft state in this phase, and so, to study the dynamic system in exam in a state-space representation. To find out these state variables it is necessary to analyze aircraft equations of motion during take-off phases, these latter described as follows.

- **Ground roll phase**: starting from standstill with brakes released and at maximum power output, the aircraft accelerates on the runway, with constant angle of attack, until it reaches a speed equals to the rotation speed V_{Rot}; after that the following subphase begin.
  - **Rotation phase**: a short phase in which the pilot gives an assigned pitching law to lift the aircraft nose and, as a result, increasing the angle of attack. This phase ends when the load factor is equal to 1, meaning that the lift has reached the value of the maximum take-off weight, and the relative lift-off speed is indicated with V_{LO}.
- **Airborne phase**: is the phases in which the aircraft, once it has lifted from the ground, gains altitude until it reaches the obstacle height of 35 ft (10.7 m) imposed by the FAR-25. This phase begin at V_{LO} and ends at the speed related to the obstacle overcoming, indicated with V_2; furthermore it can be divided into the followngs two subphases:
  - **Transition phase**: in which the aircraft rotates in order to increase the climb angle (γ) with the result of increasing the angle of attack and the relative lift coefficient, which should not surpass a safety value of the 90% of the maximumlift coefficient in take-off configuration. This sub-phase ends when the desidered climb speed is reached.
  - **Climb-out to the obstacle phase**: in which the aircraft climbs at constant climb angle until the obstacle is surpassed.

For more information regarding take-off equations of motion during each of the previously described phases, the reader can refer to [22].

The set of Ordinary Differential Equations (ODE) that models the take-off run is written in the following form:

$$\begin{Bmatrix} \dot{s} \\ \dot{V} \\ \dot{\gamma} \\ \dot{h} \end{Bmatrix} = \begin{Bmatrix} f_1(s,\, V,\, \gamma,\, h\,;\, \alpha) \\ f_2(s,\, V,\, \gamma,\, h\,;\, \alpha) \\ f_3(s,\, V,\, \gamma,\, h\,;\, \alpha) \\ f_4(s,\, V,\, \gamma,\, h\,;\, \alpha) \end{Bmatrix} \quad \text{with} \quad \begin{cases} x_1 = s \\ x_2 = V \\ x_3 = \gamma \\ x_4 = h \end{cases} \quad \text{and} \quad u = \alpha \tag{5.1}$$

These equations can be also written in a more concise way as shown below.

$$\dot{\boldsymbol{x}} = \boldsymbol{f}\left(\boldsymbol{x}\,;\, u\right) \tag{5.2}$$

The unknown $\boldsymbol{x} = [x_1,\, x_2,\, x_3,\, x_4]^{\mathrm{T}}$ is the vector of state variables. The input $u(t)$ is a given function of time, for $0 \leq t \leq t_{\text{final}}$, that corresponds to an assumed time history of the angle of attack during take-off. The right-hand sides of system (5.1) are defined by the following functions:

$$f_1\left(\boldsymbol{x}\,,\, u\right) = x_2 \tag{5.3a}$$

$$f_2\left(\boldsymbol{x}\,,\, u\right) = \frac{g}{W} \begin{cases} T(x_2) - D(x_2, u) - \mu\big[W - L(x_2, u)\big] & \text{if } \mathcal{S}(x_2, u) < 1 \\ T(x_2)\cos u - D(x_2, u) - W \sin x_3 & \text{if } \mathcal{S}(x_2, u) \geq 1 \end{cases} \tag{5.3b}$$

$$f_3\left(\boldsymbol{x}\,,\, u\right) = \frac{g}{W\, x_2} \begin{cases} 0 & \text{if } \mathcal{S}(x_2, u) < 1 \\ L(x_2, u) + T(x_2)\sin u - W\cos x_3 & \text{if } \mathcal{S}(x_2, u) \geq 1 \end{cases} \tag{5.3c}$$

$$f_4\left(\boldsymbol{x}\,,\, u\right) = x_2\, \sin x_3 \tag{5.3d}$$

The thrust $T(x_2)$ is calculated by means of the interpolating function $T_{\text{tab}}\left(V_{\text{a}}\right)$ based on a table lookup algorithm, where $V_{\text{a}} = V + V_{\text{w}}$ is the airspeed and V_{w} is the wind speed (horizontal component, positive if opposite to the aircraft motion). The drag D and lift L, as functions of airspeed V_{a} and angle of attack, are given by the following conventional formulas.

$$D(x_2, u) = \frac{1}{2}\,\rho\left(x_2 + V_{\text{w}}\cos x_3\right)^2 S\, C_D\left(u\right) \tag{5.3e}$$

$$L(x_2, u) = \frac{1}{2}\,\rho\left(x_2 + V_{\text{w}}\cos x_3\right)^2 S\, C_L\left(u\right) \tag{5.3f}$$

The switching function $\mathcal{S}$ of aircraft velocity and angle of attack is defined as follows:

$$\mathcal{S}(x_2, u) = \frac{L(x_2, u)}{W\cos x_3} \tag{5.3g}$$

The formulas (5.3) make the system (5.2) a closed set of ODE. When the function $u(t)$ is assigned and the system is associated to a set of initial conditions, in this particular case equal to $\boldsymbol{x}_0 = [0,\, 0,\, 0,\, 0]^{\mathrm{T}}$, a well-posed Initial Value Problem (IVP) is formed, which can be solved numerically. In table 5.1 are reported the take-off characteristic speeds and their corresponding requirements as defined by FAR-25.

Speed	Description	Requirement
V_S	aircraft stalling speed in take-off configuration	—
V_{MC}	minimum control speed with one engine inoperative (OEI)	—
V_1	OEI decision speed	$\geq V_{mc}$
V_{Rot}	rotation speed	$> 1.05\, V_{MC}$
V_{MU}	minimum unstick speed for safe flight	$\geq V_S$
V_{LO}	lift-off speed	$> 1.10\, V_{MU}$
		$> 1.05\, V_{MU}$ (OEI)
V_2	take-off climb speed at 35 ft	$> 1.20\, V_S$
		$> 1.10\, V_{MC}$

Table 5.1 Take-off speeds and FAR 25 requirements

It has to be highlithed that the drag coefficient C_D that appears in (5.3f) can be modelled as:

$$C_D = C_{D0} + (\Delta C_{D0})_{\text{flap+lg}} + K_g \frac{C_L^2}{\pi \text{Æ} e} \tag{5.4}$$

with $(\Delta C_{D0})_{\text{flap+lg}}$ due to flap, as shown in subparagraph 4.1.5, and landing gears, which contribution is usually about $0.010 \div 0.015$; moreover C_L is the one from the lift curve with flaps, and eventually slats, deflected. The term K_g in (5.4) incorporates the ground effect and it is calculated from [22] using the (5.5) which is a fifth order interpolating function of the graph in fugure 5.2, where the ratio h_W/b is obtained dividing the height of wing above the ground by the wing span, usually between 0.1 and 0.2 when the aircraft is on the ground and assumed as $h_W \approx h$ durign the airborne.

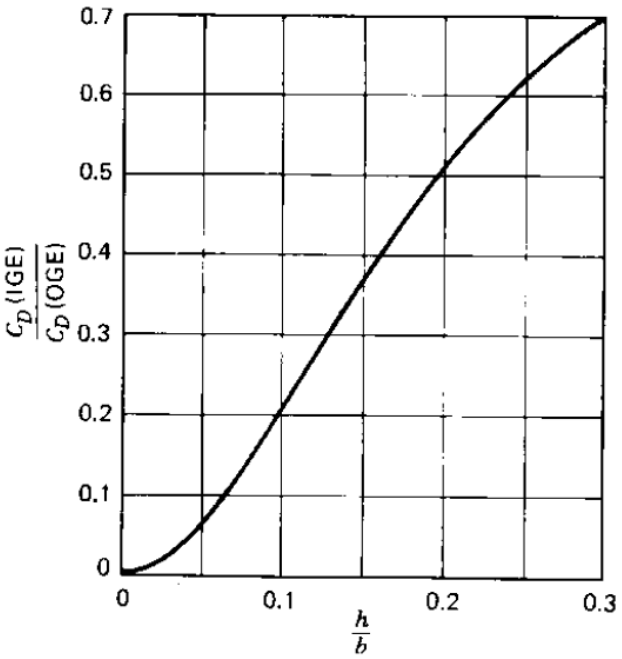

Figure 5.2 Gorund effect parameter K_g as function of the h_W/b ratio

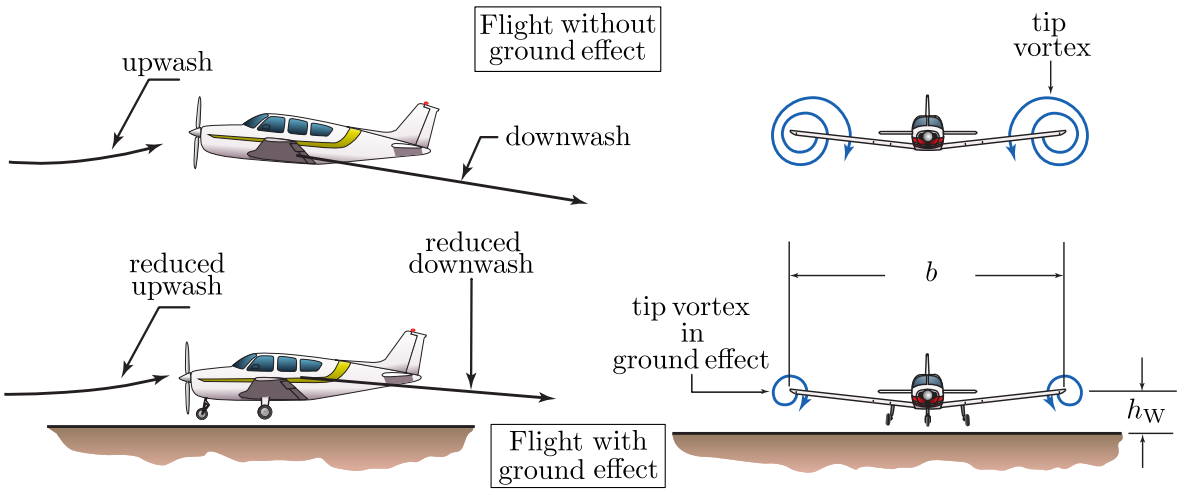

Figure 5.3 Comparison between flight with and without ground effect

$$K_g = -622.44x^5 + 624.46x^4 - 255.24x^3 + 47.105x^2 - 0.6378x + 0.0055 \qquad (5.5)$$

This polynomial equation has a coefficient of determination R^2 of 0.9999 which justifies the approximation.

In order to better understand the nature of the ground effect it is convenient to refer to [23], where the ground effect is explained as follows. As the aircraft flies close to the ground, the ground interferes with the horseshoe vortex system trailing behind the wing. Ground effect is often analyzed by putting an image horseshoe vortex system of equal but opposite strength at the same distance h_W below the ground. This image vortex system induces velocities at the wing aerodynamic center, which decreases the strength of the downwash at that point, thereby decreasing the induced angle of attack, α_i. Thus, the wing C_L is increased (or more correctly, the lift curve slope increases, giving an increase in C_L for the same geometric angle of attack, α) and the induced drag is decreased. This influence of the ground effect is a function of how close the aircraft is to the ground and of the size of the wing.

Speaking of the C_D, it has also to be noted that, at high C_L, the parabolic drag polar it's no longer accurate in describing the drag characteristics of the aircraft so that two correction factors have to be added to the (5.4). These latter triggers only when the C_L is higher than 1.2, as can be seen from the following equation, in which K_1 and K_2 values depend on the aircraft in exam.

$$C_D = C_{D0} + (\Delta C_{D0})_{\text{flap+lg}} + K_g \frac{C_L^2}{\pi \text{Æ} e} + K_1 \ (C_L - 1.2) + K_2 \ (C_L - 1.2)^2 \qquad (5.6)$$

Focusing, now, on the input law of the angle of attack, the function u can be constructed by picking the time t_{Rot} when the rotation speed V_{Rot} is reached along the ground roll; thus the $u(t)$ function can be defined as follows.

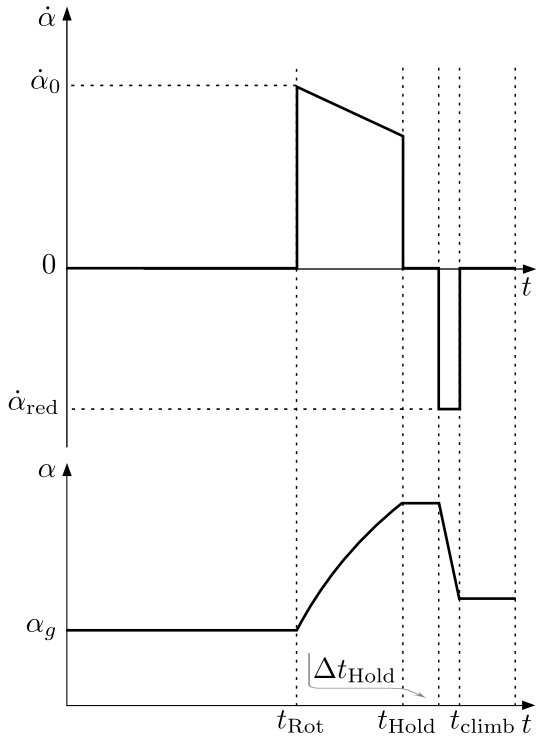

Figure 5.4 Qualitative representation of the angle of attack input law

$$u(t) = \begin{cases} \alpha_{\text{g}} & \text{if } t < t_{\text{Rot}} \\ \\ \alpha_1(t) & \text{if } t \geq t_{\text{Rot}} \end{cases} \tag{5.7}$$

with a constant α_{g} during the ground run up to the rotation speed, and a given non-zero law $\alpha_1(t)$ for the post-rotation angle of attack time history. Figure 5.4 shows a qualitative representation of the $\alpha_1(t)$ law. As can be seen, after t_{Rot} the pilot applies an initial angular velocity $\dot{\alpha}_0$, which decreases with time, according to the law written in (5.8) as function of α, until the time t_{Hold} has been reached; this particular instant is related to the achievement of the maximum admitted lift coefficient in take-off, which is set at 90% of the $C_{L\text{max,TO}}$.

$$\dot{\alpha} = \dot{\alpha}_0 \ (1 - k_\alpha \, \alpha) \tag{5.8}$$

In equation (5.8), the k_α slope is assigned (expressed in $1/°$ and dependent on the aircraft in exam), while the initial angular velocity $\dot{\alpha}_0$ is calculated as follows.

$$\dot{\alpha}_0 = \frac{\Delta\alpha}{\Delta t_{\text{Rot}}} = \frac{\alpha_{\text{LO}} - \alpha_g}{\Delta t_{\text{Rot}}} \tag{5.9}$$

where α_{LO} can be obtained from the lift curve of the wing, with flaps deflected in take-off configuration, by assigning the $C_{\text{L}_{\text{LO}}}$; this can be derived from the $C_{L\text{max,TO}}$ dividing it by the parameter K^2_{LO}, which represents the quantity that has to be multiplied by V_{S} in order to obtain V_{LO} (for example 1.1 with reference to table 5.1).

From this point on the pilot stops the pitching manouver and keeps the angle of attack constant for an assigned Δt_{Hold}. During this time interval, the lift coefficient is high and, as a result, also the induced drag is high so that aircraft acceleration will reduce. After this short time

interval the pilot has to reduce the angle of attack in order to avoid the acceleration to decrease too much and so an assigned negative angular velocity $\dot{\alpha}_{\text{red}}$ is applied; the latter assumed to be constant for simplicity. Finally, since the decrease of α determines also a reduction in C_L, the time t_{climb} will be reached when the load factor is reduced to 1; this means that a balance of the forces, perpendicular to the flight path, has been achieved and so the climb phase, at constant γ, can begin, leaving α constant and equal to last value reached. Moreover, from this time on, the lift value is constant and equal to $W \cdot \cos\gamma$, in order to maintain the load factor equal to 1; while the C_L is derived from the lift value using the (5.16).

5.1.2 OEI take-off run and balanced field lenght

A good description of the take-off with one engine failure is proposed in [26]. Here it is explained that in the event of an engine failure during the take-off roll the pilot must decide whether to continue the take-off or, instead, abort the take-off and decelerate to a stop on the runway. Obviously, if the engine failure occurs when the aircraft is traveling very slowly, the aircraft should be kept on the ground and brought to a stop at some safe location off the runway. Conversely, if the engine failure occurs when the aircraft is close to the take-off speed the take-off should be continued. The designer must provide a means for deciding whether it is safer to abort the take-off or continue it. The critical velocity, denoted as V_{act}, is the velocity at which action is taken, not that at which the decision to act is taken. The time between the recognition of an engine failure, which occurs at V_{ef}, and the critical velocity V_{act}, when action is taken is required to be more than one second. Generally this time period, which is set by the reaction time of the pilot, is taken to be about 3 s. If the pilot's decision is to continue the take-off with one engine inoperative, the distance to the lift-off speed V_{LO} and to the subsequent climb-out to 35 ft height above the runway, will obviously be longer than with all engines operating.

The calculation of the take-off distance in this situation is quite the same as the one explained previously, with the difference that now there is a discontinuity in thrust due to the broken engine. In particular, the thrust, $T(x_2)$, will still be read from the database but considering a number of engines reduced by one from the time t_{ef} at which the engine failure occurs.

On the other hand, in the case of the aborted take-off the pilot will apply the necessary braking procedures in order to get the maximum permissible deceleration while maintaining adequate control of the airplane's motion. The portion of the aborted take-off run up to the engine failure velocity V_{ef} is calculated in the same way as that for the continued take-off, so that the distance is the same in both cases. From this point on, until the pilot reacts by activating brakes, there is only a discontinuity in thrust due to the failed engine; while, after the time interval in which the pilot decides to abort the take-off, the thrust is set to minimum (ideally zero) and the brakes action provides an higher friction coefficient. During this last phase, the equation (5.15b) changes in the following.

$$f_2\big(\boldsymbol{x}\,,\,u\big) = \frac{g}{W}\left\{-D(x_2,u) - \mu_{\text{brakes}}\big[W - L(x_2,u)\big]\right\} \tag{5.10}$$

where μ_{brakes} is bigger than μ and it is usually about 0.3. Furthermore, it has to be noted that,

even if the aircraft in exam is supplied with a reverse thrust device, this effect has not to be taken into account for a more conservative result.

Instead of considering the limiting cases of aborting take-off at low V_{act} and continued take-off at high V_{act}, it is useful determine the critical velocity for which the distance required to continue the take-off is equal to the distance required to safely abort it. This velocity is the one from table 5.1 and it's called *decision speed* V_1, while the related distance is called the *balanced field length*. The latter, in particular, plays an important role in the sizing of the runway since is the maximal distance the aircraft can cover both in continued take-off, both in aborted take-off. In order to calculate this distance, and the related velocity, it's possible to evaluate, at different V_{act}, both the continued take-off distance with one inoperative engine, both the aborted take-off distance. Each couple of speed and distance can then be plotted with the result of building the curves of figure 5.5. The intersection of these latter, at which the two distances are the same, defines the *balanced field length* and the V_1.

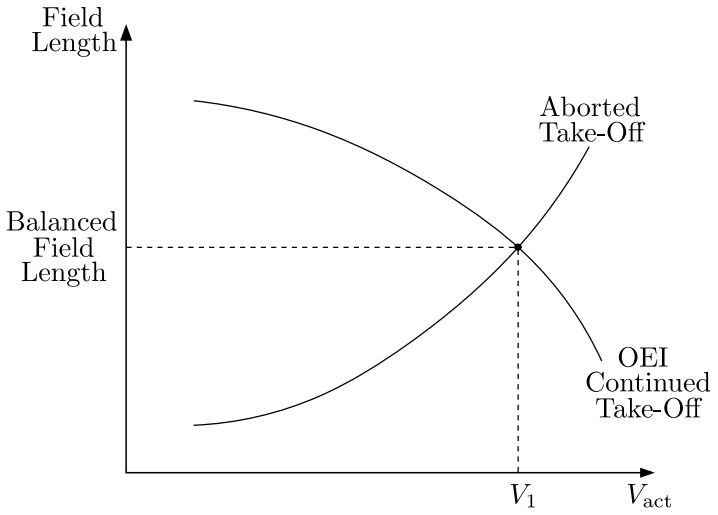

Figure 5.5 Qualitative representation of the field distances required to continue a takeoff or to abort it when one engine fails as a function of critical velocity

As expected, the take-off distance on the curve related to the continued take-off with one inoperative engine decreases with the failure speed, tending to the AOE condition; while the take-off distance on the other curve grows with the failure speed because the deceleration to stop the aircraft begins from an higher speed, requiring more distance to be dissipated.

5.1.3 Landing distance calculation

Similar to the take-off, the landing phase can be considered as made up of two main components: an air run and a ground run as shown in figure 5.6. A comprehensive description of these two phases is provided by [26] and it is reported below.

The air run may be considered to start at the an obstacle height of 50 ft, so that in an x-y coordinate system with x = 0 at y = 50 ft, the approach speed equals V_A, which, according to FAR, must be at least 1.3 V_s in landing configuration. The aircraft executes a gradual flare of large radius R, turning the pitch angle from θ around 2 ° or 3 ° down to zero at which time the aircraft has slowed to V_{Flare}, that is, to a value around $1.2 \div 1.25 V_s$ (in landing configuration)

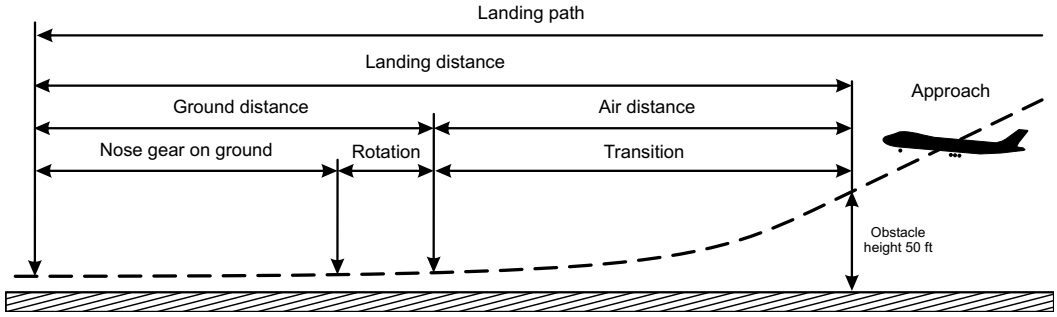

Figure 5.6 Scheme of an aircraft landing run

and settled onto the runway, ending the air run at x = $S_A + S_{\text{Flare}}$ as shown in figure 5.7. To calculate this distance the simpliest way is to assume that the flare rotation describes a circular trajectory and that θ_a, during approach is small and almost equal to θ_f. By using these hypotesis the two distances S_A and S_{Flare} can be calculated through a geometrical approach which follows the steps below.

- Assuming a load factor (n) of 1.2 and a V_{Flare} equal to the mean value between $V_A = 1.3\ V_s$ and $V_{\text{TD}} = 1.15\ V_s$ ($V_{\text{Flare}} = 1.23\ V_s$, the radius R can be calcuated as

$$R = \frac{V^2}{g\,(n-1)} = \frac{V_{\text{Flare}}^2}{0.2\ g} \tag{5.11}$$

  where g is the gravitational constant.

- With a θ_a around 2 ° or 3 °, it's possible to calculate the flare height from ground as follows.

$$h_{\text{Flare}} = R\,(1 - \cos\theta_a) \tag{5.12}$$

- At this point, known θ_a, R and h_{Flare}, the two distances can be calculated as follows.

$$S_A = \frac{50 - h_{\text{Flare}}}{\tan\theta_a}; \qquad S_{\text{Flare}} = R\sin\theta_a \tag{5.13}$$

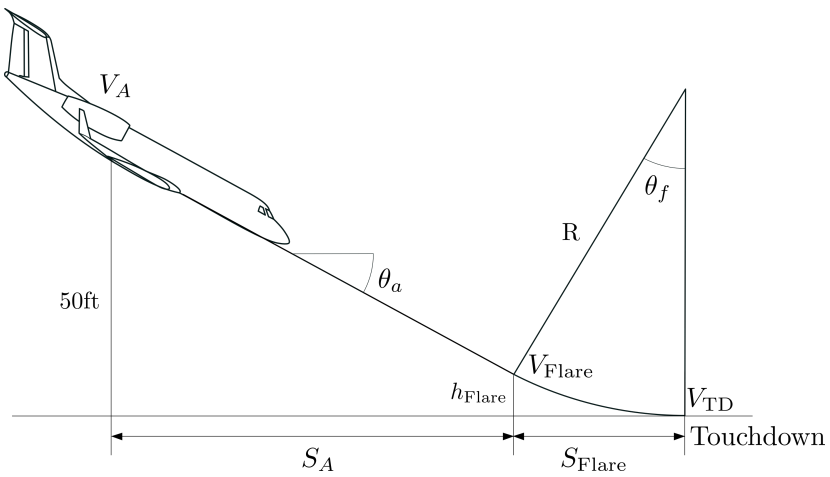

Figure 5.7 Scheme of the air run of the landing phase

The ground run begins when the aircraft touches the ground and ends when the speed is equal to zero; the first part of the ground run is named *free-roll distance*, which lasts around 3 s, and represents the distance covered while the pilot reduces the power to idle, retracts the flaps, deploys the spoilers, and applies the brakes. The second part of the ground run is almost the same described during the aborted take-off with the differece that now there is no engine failure and the focus is on the decelerated trait.

Also in this case the description of the aircraft motion can be easily carried out integrating an appropriate set of ODE, which are describe below.

$$\begin{Bmatrix} \dot{s} \\ \dot{V} \end{Bmatrix} = \begin{Bmatrix} f_1(s,\, V) \\ f_2(s,\, V) \end{Bmatrix} \qquad \text{with} \quad \boldsymbol{x} = \begin{Bmatrix} x_1 = s \\ x_2 = V \end{Bmatrix} \tag{5.14}$$

where the right-hand sides of system (5.14) are defined by the following functions:

$$f_1(\boldsymbol{x}) = x_2 \tag{5.15a}$$

$$f_2(\boldsymbol{x}) = \frac{g}{W} \begin{cases} -T_{\text{Rev}}(x_2) - D(x_2) - \mu\big[W - L(x_2)\big] & \text{if } t \leqslant t_{\text{fr}} \\ -T_{\text{Rev}}(x_2) - D(x_2) - \mu_{\text{brakes}}\big[W - L(x_2)\big] & \text{if } t > t_{\text{fr}} \end{cases} \tag{5.15b}$$

The drag and lift values are calculated in the same way of the take-off ground roll, while T_{Rev} is a reverse thrust produced by thrust reversers, for jet engine, or by a particular regulation of the blade pass, for a propeller engine. The latter is usually neglected for the ground distance calculation because it helps the deceleration of the aircraft, but for more clarity, it has been considered in the equation. Furthermore the instant t_{fr} is related to the end of the *free-roll distance* and it's set at 3 s as explained before.

When both the ground distance, both the airborne phase distance are calculated, it's finally possible to determine the total landing distance by summing these latter.

5.2 Java class architecture

Now that the theoretical background is fully described, it's time to analyze the structure of the Java classes implemented inside JPAD dedicated to the calculation of the take-off and landing distances.

5.2.1 Take-Off

As in previous chapters, a dedicated Java class, named `CalcTakeOff`, has been created to manage all required methods. The first component to be described is the class constructor; the latter, in addition to linking all the input parameter to the related class fields, provides a series of preliminary calculations which defines other required input data, not given from the user, such as the maximum lift coefficient in take-off configuration, or the stalling speed in take-off, and some of the characteristic speeds defined in table 5.1. More in detail, the constructor evaluates, firstly, all high-lift devices effects, which data are supplied in the test

`aircraft`	An `Aircraft` class object representing an aircraft parametric model
`theConditons`	An `OperatingConditions` object representing aircraft flight conditions
`highLiftCalculator`	A `CalcHighLiftDevices` object for managing flap and slat effects
`dtRot`	The assigned time interval of the rotation phase
`dtHold`	The assigned time interval of the constant C_L phase
`kcLMax`	Percentage of the $C_{L\max,\mathrm{TO}}$ not to be surpasses
`kRot`	Percentage of V_s which defines the rotation speed
`kLO`	Percentage of V_s which defines the lift-off speed
`kFailure`	A parameter which defines the drag increment due to engine failure and the consequent rudder deflection
`k1`	Linear correction factor of the parabolic drag polar at high C_L
`k2`	Quadratic correction factor of the parabolic drag polar at high C_L
`phi`	Throttle setting
`kAlphaDot`	A coefficient which defines the decrease of $\dot{\alpha}$ during manouvering
`alphaReductionRate`	A constant negative pitching angular velocity to be maintained after holding the C_L constant
`mu`	The friction coefficient without brakes action
`muBrake`	The friction coefficient with brakes activated
`wingToGroundDistance`	The distance between the wing and the ground
`obstacle`	A given altitude value to overcome which defines the airborne phase ending
`vWind`	The horizontal component of the wind speed, positive if opposite to the aircraft motion
`alphaGround`	The angle of attack, in the ACRF, of the wing when the aircraft is on the ground
`iw`	The angle between the wing root chord and the ACRF x-axis

Table 5.2 `CalcTakeOff` constructor input

class as explained in the paragraph 4.2, by using the method `calculateHighLiftDevicesEffects` of the `CalcHighLiftDevices` class, described in the previous chapter in paragraph 4.2. After that it's possible to define the maximum lift coefficient, $C_{L\max,\mathrm{TO}}$, the C_{L0}, with high-lift devices effects computed, and the lift coefficient during the ground roll phase, C_{Lg}, related to the constant angle of attack α_g; in particular the last two quantities are calculated using the method `calcCLatAlphaHighLiftDevice`, of the `CalcHighLiftDevices` class, respectively with $\alpha_w = 0°$ and $\alpha_w = \alpha_g + i_w$. With the $C_{L\max,\mathrm{TO}}$ known, the stalling speed in take-off

configuration is calculated using the classical formula provieded below.

$$V_s = \sqrt{\frac{2W_{\text{TO}}}{\rho\ S\ C_{L\text{max,TO}}}} \tag{5.16}$$

From this speed, both the V_{Rot} both the V_{LO} are calculated multipling the V_s by the two parameters, k_{Rot} and k_{LO}, defined in table 5.2. At this point, the ratio h_{W}/b and the ground effects correction parameter K_g of the (5.5) are calculated; while all the Lists, which will store all physical quantities of interest at every integration step, are initialized together with a custom Map, named `TakeOffResultsMap`. The latter, in particular, has been created with the purpose of store the state vector, and all the related physical quantities, only at some key point during the take-off run, like the end of the ground roll phase, the end of the rotation phase and the end of the airborne phase.

The `TakeOffResultsMap` class is made up of a builder, which accepts nothing as input and provides the initialization of all the Lists required to store the wanted data, and of other two method which are explained below.

- `initialize`, which clears all the Lists in order to make them reusable for other calculations
- `collectResults`, which accepts as input all data from table 5.3 in order to add them to the related List

Once all data are stored, it's easy with this Map to get one, or more than one, result; all it has to be done is call the related *getter* method of which the class is supplied.

It has to be noted that all these preliminary calculations and Lists initializations are put into the constructor for a reason; in fact, as these are all quite heavy operations in terms of computational cost, put them in the constructor means that they are carried out only once allowing a more rapid use, even iterative, of the main method that will be described shortly.

The most important method of the class is `calculateTakeOffDistanceODE` which is in charge of the resolution of the ODE set presented in (5.3). This method accepts as input two parameters which are used to determine, firstly, if an engine failure has occurred during the take-off run, and then if the take-off run has to be aborted; these are:

- `vFailure`, a `Double` value representing the failure speed in m s^{-1}. Can be set to `null` if the user dosen't want to calculate the OEI take-off distance.
- `isAborted`, a `boolean` flag which is `true` if the method has to calculate the aborted take-off distance.

After performing a check upon these two variables, the method knows which case it has to study and proceeds with the calculation; in particular, it creates the integrator object of the class `HighamHall54Integrator`, which implements the Interface `FirstOrderIntegrator`. For more information, the reader can refer to [13].

More in detail, the `HighamHall54Integrator` class implements a fifth order Higham and Hall integrator which uses seven functions evaluations per step and is supplied with stepsize control,

`timeValue`	The integration time, in s, at the step to save
`thrustValue`	The engine thrust, in N, at the step to save
`thrustHorizontalValue`	The engine thrust component on the ACRF x-axis, in N, at the step to save
`thrustVerticalValue`	The engine thrust component on the ACRF z-axis, in N, at the step to save
`frictionValue`	The friction, in N, at the step to save
`liftValue`	The lift, in N, at the step to save
`dragValue`	The drag, in N, at the step to save
`totalForceValue`	The total force, in brakets in the equation (5.15b), at the step to save in N
`loadFactorValue`	The load factor at the step to save
`speedValue`	The speed, in $\mathrm{m\,s^{-1}}$, at the step to save
`rateOfClimbValue`	The rate of climb from equation (5.3g), in $\mathrm{m\,s^{-1}}$, at the step to save
`accelerationValue`	The acceleration from equation (5.15b), in $\mathrm{m\,s^{-2}}$, at the step to save
`groundDistanceValue`	The horizontal distance, in m, covered at the step to save
`verticalDistanceValue`	The altitude, in m, reached at the step to save
`alphaValue`	The angle of attack α, in °, at the step to save, in ACRF
`alphaDotValue`	The pitching angular velocity, in $^\circ\,\mathrm{s^{-1}}$, at the step to save
`gammaValue`	The ramp angle γ, in °, at the step to save
`gammaDotValue`	The $\dot{\gamma}$ value, in $^\circ\,\mathrm{s^{-1}}$, at the step to save
`thetaValue`	The $\theta = \alpha + \gamma$ value, in °, at the step to save
`cLValue`	The lift coefficient at the step to save
`cDValue`	The drag coefficient at the step to save

Table 5.3 `collectResults` input data

automatic step initialization and continuous output. The latter has proven to be the best choise, among other possible integrators (viewables in [12]) which implement the previous Interface, because it provides the better compromise between calculation time and accuracy using the following settings.

```
1 FirstOrderIntegrator theIntegrator = new HighamHall54Integrator(
2         1e-6,       // minimal step
3         1,          // maximal step
4         1e-17,      // allowed absolute error
5         1e-17       // allowed relative error
6         );
```

Listing 5.1 HighamHall54Integrator class object creation

Beside the integrator, the method needs the set of equation to integrate; these are passed to it through the object of a dedicated inner class, named `DynamicsEquationsTakeOff`, which implements the Interface `FirstOrderDifferentialEquations` [11]. Thus, the `DynamicsEquationsTakeOff` class provides the set of ODE (5.3) and, in particular, is supplied with a series of methods necessary to calculate the required physical quantities used into these equations and described in the previous paragraph. Furthermore, thanks to the use of the variable `isAborted`, the class can easly switch the equations set in case of aborted take-off as shown in the subparagraph 5.1.2.

In order to take into account of particular events which can happen during the take-off run, the method `calculateTakeOffDistanceODE` is supplied with several implementation of the Interface `EventHandler` [13]. The latter, through the definition of a specific function, can determine the occurrence of the wanted event by monitoring whether the sign of the defined function changes. Moreover it allows to manage the time step at which the event has occurred, so that it's possible to save, into the `TakeOffResultsMap`, the state vector and all the physical quantities of interest. Each implementation of the Interface can also impose one of the folllowing actions to the integrator.

- **CONTINUE**, the integration simply goes on
- **STOP**, the integration stops when the event triggers
- **RESET_STATE**, the state vector can be changed when the event triggers
- **RESET_DERVATIVES**, the set of equation can be changed when the event triggers

In the case in exam, six events are monitored by six implementation of the `EventHandler` Interface as shown below.

`ehCheckFailure`	It checks when the speed, x_2, becomes greater than the input `vFailure` determining, in this way, the instant of the engine failure occurrence
`ehCheckVRot`	It checks when the speed, x_2, becomes greater than the rotation speed `vRot` determining, in this way, the instant at which the ground roll ends and the rotation phase begins
`ehEndConstantCL`	It checks when the time, t, becomes greater than the sum of t_{Hold} and of the given time interval Δt_{Hold} determining, in this way, the instant at which the angle of attack, and the related C_L, stops to be kept constant
`ehCheckObstacle`	It checks when the altitude, x_4, becomes greater than the given obstacle height (35 ft) determining, in this way, the instant at which the airborne phase, and so the entire take-off, ends
`ehCheckBrakes`	It checks when the time, t, becomes greater than the sum of t_{Failure} and of the given time interval Δt_{Rec}, required to the pilot to recognize the failure, determining, in this way, the instant at which the pilot, which has decided to abort the take-off, actions the brakes in order to stop the aircraft run
`ehCheckStop`	It checks when the speed, x_2, becomes lower than zero determining, in this way, the instant at which the aircraft has stopped

Table 5.4 `EventHandler` implementation inside the method `calculateTakeOffDistanceODE`

Each event of the table 5.4 defines a time instant usable, by the class `DynamicsEquationsTakeOff`, to determine when the derivatives, or the calculation of the related physical quantities, have to switch from an equation to another, this allows to manage in a very easy way the definition of the profile of $\dot{\alpha}$ and α, as well as the derivatives change shown in (5.15b) and (5.3c).

In order to make each `EventHandler` usable by the integrator, they have to be added to the `HighamHall54Integrator` as follows.

```
1  if(!isAborted) {
2    theIntegrator.addEventHandler(ehCheckVRot, 1.0, 1e-3, 20);
3    theIntegrator.addEventHandler(ehCheckFailure, 1.0, 1e-3, 20);
4    theIntegrator.addEventHandler(ehEndConstantCL, 1.0, 1e-3, 20);
5    theIntegrator.addEventHandler(ehCheckObstacle, 1.0, 1e-3, 20);
6  }
7  else {
8    theIntegrator.addEventHandler(ehCheckVRot, 1.0, 1e-3, 20);
9    theIntegrator.addEventHandler(ehCheckFailure, 1.0, 1e-3, 20);
10   theIntegrator.addEventHandler(ehCheckBrakes, 1.0, 1e-3, 20);
11   theIntegrator.addEventHandler(ehCheckStop, 1.0, 1e-6, 20);
12 }
```

Listing 5.2 HighamHall54Integrator class object creation

As can be seen the `boolean` variable `isAborted` plays an important role in determining which events have to be checked whether or not the take-off is aborted or continued. Also noteworthy, the method `addEventHandler` arguments, which represent, respectively, the following parameters.

- The event to be checked
- The maximal time interval between switching function checks (this interval prevents missing sign changes in case the integration steps becomes very large)
- The convergence threshold in the event time search
- The upper limit of the iteration count in the event time search

The decision of these parameters values has derived from a compromise between accuracy and computational time.

Another important feature that the ODE package provides is the possibility to manage each time step, even if no event is triggered; in this way the developer can, for example, store data into an output file, or manage some events which are independent from the time or the state vector as they were in the `EventHandler` Interface. The tool which allows all these feature is the `StepHandler` Interface [13]; in this particular case, this Interface has only one implementation, added to the integrator, which is in charge of store the state vector, the time and all the related physical quantities, into their related Lists, at every time step in order to make them usable outside this method. Moreover it has a key role in managing three events, to be observed only if the variable `isAborted` is false, that could not be handled well by the `EventHandler` Interface; these are the followings.

- A check upon the load factor to catch the instant at which, for the first time, it reaches a value of 1; this instant is t_{EndRot} and determines the beginning of the airborne phase together with the changes in the derivatives shown in (5.15b) and (5.3c)
- A check upon the C_L in order to determine when it reaches the threshold value defined by `kcLMax` multiplied for the $C_{L\text{max,TO}}$. The related instant is the t_{Hold} of the beginning of the constant α and C_L phase
- A second check on the load factor in order to define the instant at which its value is reduced to 1 after having applied the constant $\dot{\alpha}_{\text{Red}}$ angular velocity. This instant defines t_{climb}

Each of these events is monitored by the value of the variable `isAborted` and by the observation both of the evolution, through a single time step, of the last value added to the List of interest, both of the time, which is compared to one of the reference instants described previously.

The method is, finally, completed by assigning the initial state vector, calling the method `integrate`, which is in charge of the integration process, and ereasing all the `StepHandler` and `EventHandler` implementations at the end of the process; this in order to make the method reusable later.

Now that a method for calculating the take-off distance, in every case, is available, it can be used iteratively in order to determine the *balanced field length* and the related *decision speed* V_1. The method in charge of this is called `calculateBalancedFieldLength` and follows the following guideline. First of all an array of five values of possible failure speeds (between 2 m s^{-1} and the lift-off speed V_{LO}) is defined; after that the method `calculateTakeOffDistanceODE` is called for each of these speeds both in case of OEI continued take-off (`isAborted` set to `false`), both in case of aborted take-off (`isAborted` set to `true`). It's important to highlight that between each call of the method `calculateTakeOffDistanceODE`, all reference instants and Lists have to be reinitialized; this can be done using the dedicated method `initialize` defined into this class.

Each failure speed, OEI take-off distance and aborted take-off distance calculated this way, is then stored into a dedicated array so that it's possible to define an interpolating function for the two distances. More in detail, since five points are few to describe the curves in figure 5.5, a new array of 250 values of failure speeds is defined useing the same constrain of the previous one; the latter is then used to build a spline interpolating function, for both the OEI take-off distance and aborted take-off distance, using the five points calculated previously. From the intersection of the last two interpolated curves is possible to define the *balanced field length* and the *decision speed* V_1 as shown in figure 5.5. It should be noted that the choise of using interpolating functions and not to perform more times the calculation of the take-off distance in the two cases, derives from the will to speed up the calculation so that it's possible to have accurate results in less than one second.

In conclusion, the class is completed by two methods in charge of plotting the curves of interst from the related Lists and the *balanced field length* chart; these are the followings.

- `createTakeOffCharts`, provides the conversion of the previous Lists into `double` arrays so that they can be managed by the plotting method; morever it uses the varaible `isAborted` to determine which charts have to be plotted since some of them are useless in case of the aborted take-off

- `createBalancedFieldLengthChart`, provides the creation of the chart of figure 5.5 using the interpolated curves, on the y-axis, and the speed array with 250 values, on the x-axis

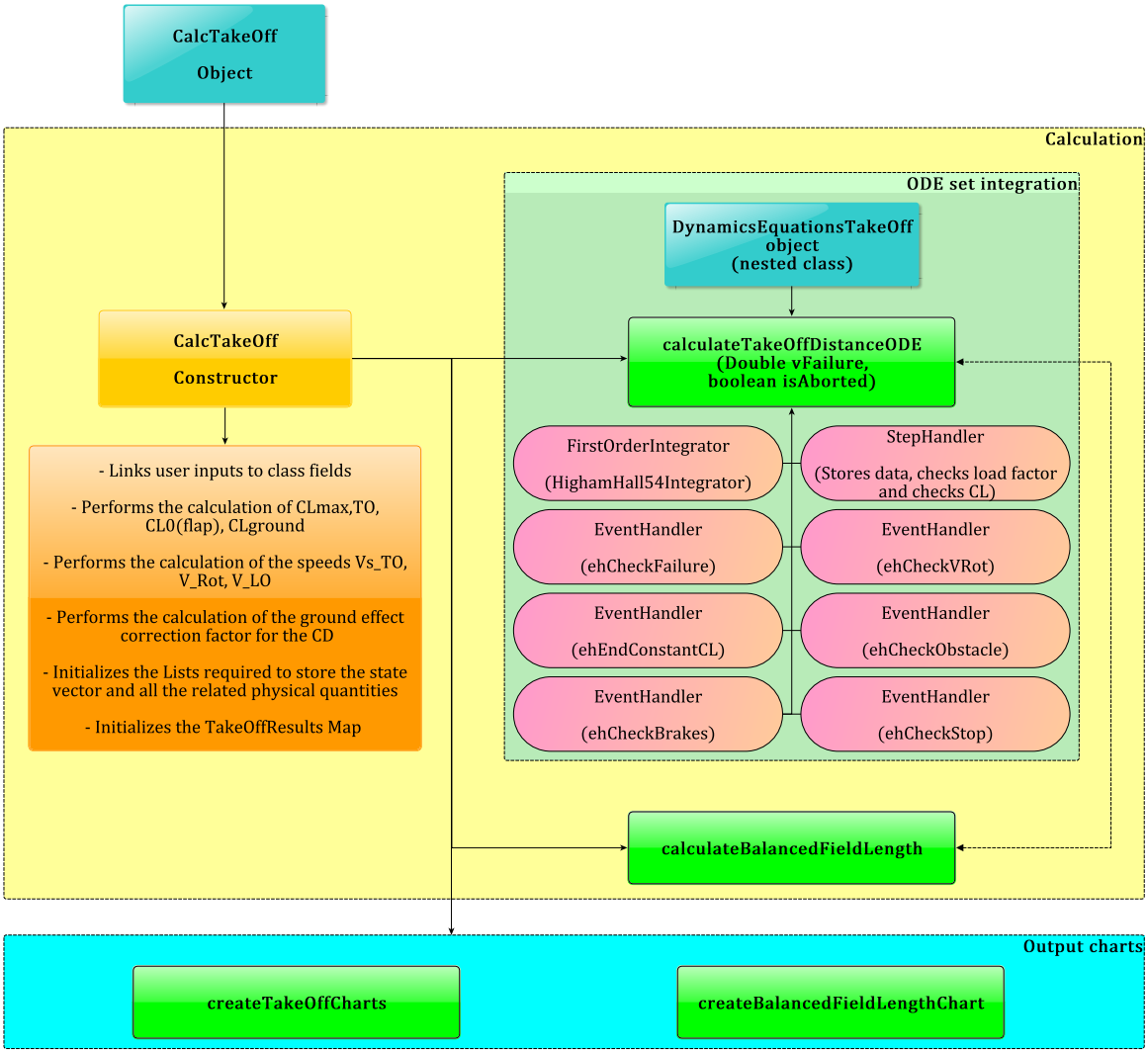

Figure 5.8 `CalcTakeOff` class flowchart

5.2.2 Landing

The guideline followed for implementing the Java class in charge of the landing distance calculation is very similar to the one used in building the take-off class. The choice of having these two classes separated derives from the willing of not make the previous take-off class much complicated, since it's already full of flag and events necessary to manage the different possible situations.

The class in exam is named `CalcLanding` and, similarly to `CalcTakeOff`, has a constructor in charge of performing all the preliminary operations such as linking all input data to the class

`aircraft`	An `Aircraft` class object representing an aircraft parametric model
`theConditons`	An `OperatingConditions` object representing aircraft flight conditions
`highLiftCalculator`	A `CalcHighLiftDevices` object for managing flap and slat effects
`kA`	Percentage of V_s which defines the approach speed V_A
`kFlare`	Percentage of V_s which defines the flare speed V_{Flare}
`kTD`	Percentage of V_s which defines the touchdown speed V_{TD}
`mu`	The friction coefficient without brakes action
`muBrake`	The friction coefficient with brakes activated
`wingToGroundDistance`	The distance between the wing and the ground
`obstacle`	A given altitude value to overcome which defines the airborne phase ending
`vWind`	The horizontal component of the wind speed, positive if opposite to the aircraft motion
`alphaGround`	The angle of attack, in the ACRF, of the wing when the aircraft is on the ground
`iw`	The angle between the wing root chord and the ACRF x-axis
`thetaApproach`	The pitch angle during the approach phase (in °)
`nFreeRoll`	The number of seconds that characterize the free-roll phase

Table 5.5 `CalcLanding` constructor input

fields, evaluate the high-lift devices effects on the wing and calculate the characteristics speeds during the landing phase. Since the class constructor is almost equal to the one in `CalcTakeOff` the reader can refer to previous paragraph for more detailed information.

The first calculation method is `calculateAirborneDistance` which implements equations (5.11), (5.12) and (5.13) and populates the fields `sApproach` and `sFlare` with the calculated distances. Furthermore it begin to fill the `speed`, `landingDistance` and `verticalDistance` Lists in order to allow the plot of the trajecroty and the speed during the whole landing phase.

The second calculation method is `calculateGroundRollLandingODE`, which shares the same architecture of the method `calculateTakeOffDistanceODE` and accepts as input a `double` value, named `phiRev`, which represents the percentage of the thrust, calculated from the engine database, to be be used as reverese thrust in eqaution (5.15b). The `FirstOrderIntegrator` is still the `HighamHall54Integrator`, while the `FirstOrderDifferentialEquations` implementation is an instance of the nested class `DynamicsEquationsLanding` defined in the same way of the take-off class.

In order to manage the only significant event, which is when the aircraft reaches a speed equal to zero, only one `EventHandler` has been implemented following the structure of `ehCheckStop` defined for the aborted take-off run. Moreover all the physical quantities related to the ground

run are stored in the related Lists at every integragtion step using the an istance of the Interface `StepHandler`.

The peculiarity of this case, unlike the take-off, is that the integration starts from the end of the flare phase, with an initial position and a speed different from zero. This means that this method requires having previously carried out the calculation of the airborne distance, or having previously assigned values to `sApproach` and `sFlare`.

To allow the calculation of the total landing distance, without forcing the user to call each of the previous methods, these latter are enclosed inside a new method named `calculateLandingDistance`. This choice derives both from the willing of simplify the usage of the class, both from the willing of preventing runtime errors in case the user wants to only calculate the distance to the ground without having assigned the values of the distances of the previous phase. The class is then completed by the method `createLandingCharts` which is in charge of creating the trajectory and speed charts, during the whole landing phase, along with the charts of all the physical quantities of interest during the ground run.

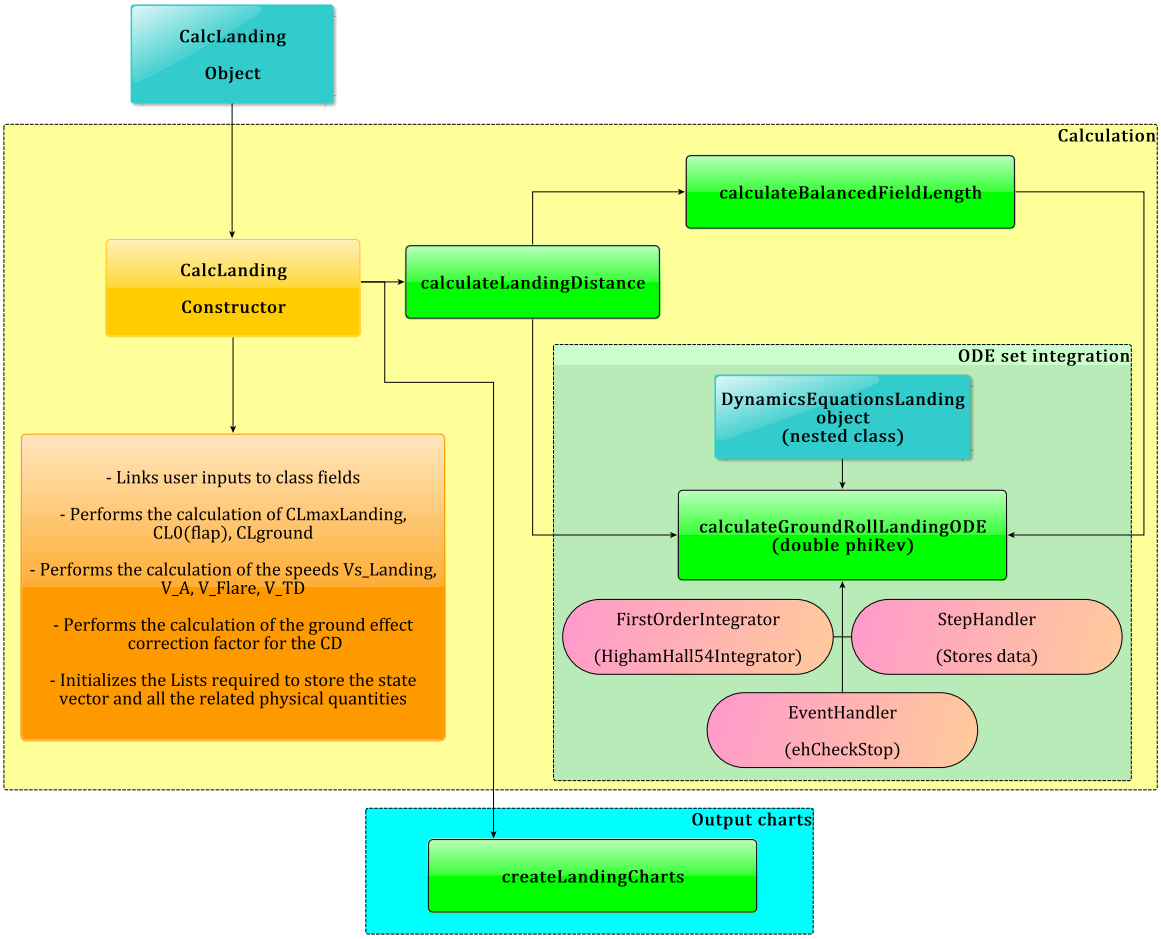

Figure 5.9 `CalcLanding` class flowchart

5.3 Case study: ATR-72

This conclusive paragraph has the aim of describing the Java class created to test the features implemented into `CalcTakeOff` class and `CalcLanding` classes; in particular a description of how to build these classes as well as a presentation, with comments, of the most important charts created, will be provided. Since the calculations are the same both for the B747-100B, both for the ATR-72, only the latter will be analyzed.

The first thing to do is to build the aircraft parametric model, and its analysis object, as described in paragraph 1.4; then all information regarding high-lift devices have to be provided, in the same way of paragraph 4.3, in order to allow the constructors of `CalcTakeOff` and `CalcLanding` classes to evaluate their effects (for the landing configuration the flaps deflection has been changed to 45 ° from the initial 20 ° related to the take-off). At this point all the preliminary steps are completed and it's possible to create `CalcTakeOff` and `CalcLanding` objects giving as input all data from tables 5.2 and 5.5.

```
1      Amount<Duration> dtRot = Amount.valueOf(3, SI.SECOND);
2      Amount<Duration> dtHold = Amount.valueOf(0.5, SI.SECOND);
3      double mu = 0.025;
4      double muBrake = 0.3;
5      double kAlphaDot = 0.06; // [1/deg]
6      double kcLMax = 0.85;
7      double kRot = 1.05;
8      double kLO = 1.1;
9      double kFailure = 1.1;
10     double phi = 1.0;
11     double alphaReductionRate = -3; // [deg/s]
12     Amount<Length> wingToGroundDistance = Amount.valueOf(4.0, SI.METER);
13     Amount<Length> obstacle = Amount.valueOf(35, NonSI.FOOT).to(SI.METER);
14     Amount<Velocity> vWind = Amount.valueOf(0.0, SI.METERS_PER_SECOND);
15     Amount<Angle> alphaGround = Amount.valueOf(0.0, NonSI.DEGREE_ANGLE);
16     Amount<Angle> iw = Amount.valueOf(2.0, NonSI.DEGREE_ANGLE);
17 //    PARAMETERS USED TO CONSIDER THE PARABOLIC DRAG POLAR CORRECTION AT HIGH CL
18     double k1 = 0.0, k2 = 0.0;
19
20     CalcTakeOff theTakeOffLandingCalculator = new CalcTakeOff(
21         aircraft,
22         theCondition,
23         highLiftCalculator,
24         dtRot,
25         dtHold,
26         kcLMax,
27         kRot,
28         kLO,
29         kFailure,
30         k1,
31         k2,
32         phi,
33         kAlphaDot,
34         alphaReductionRate,
```

```
        mu,
        muBrake,
        wingToGroundDistance,
        obstacle,
        vWind,
        alphaGround,
        iw
        );
```

Listing 5.3 Input data and `CalcTakeOff` object creation

```
    double mu = 0.025;
    double muBrake = 0.3;
    double kA = 1.3; // [1/deg]
    double kFlare = 1.23;
    double kTD = 1.15;
    double phiRev = 0.0; // this takes into account of the reverse thrust
    Amount<Duration> nFreeRoll = Amount.valueOf(3, SI.SECOND);
    Amount<Length> wingToGroundDistance = Amount.valueOf(4.0, SI.METER);
    Amount<Length> obstacle = Amount.valueOf(50, NonSI.FOOT).to(SI.METER);
    Amount<Velocity> vWind = Amount.valueOf(0.0, SI.METERS_PER_SECOND);
    Amount<Angle> alphaGround = Amount.valueOf(0.0, NonSI.DEGREE_ANGLE);
    Amount<Angle> iw = Amount.valueOf(2.0, NonSI.DEGREE_ANGLE);
    Amount<Angle> thetaApproach = Amount.valueOf(3.0, NonSI.DEGREE_ANGLE);

    CalcLanding theLandingCalculator = new CalcLanding(
        aircraft,
        theCondition,
        highLiftCalculator,
        kA,
        kFlare,
        kTD,
        mu,
        muBrake,
        wingToGroundDistance,
        obstacle,
        vWind,
        alphaGround,
        iw,
        thetaApproach,
        nFreeRoll
        );
```

Listing 5.4 Input data and `CalcLanding` object creation

Now, regarding the take-off, if the user wants to perform a single calculation of the take-off distance, he can call the method `calculateTakeOffDistanceODE`, specifing the condition he wants to analyze. The possible situations are resumed below, where `vFailure` is an assigned `Double` value.

```
// AOE condition
theTakeOffLandingCalculator.calculateTakeOffDistanceODE(null, false);
```

```
// OEI continued take-off
theTakeOffLandingCalculator.calculateTakeOffDistanceODE(vFailure, false);
// OEI aborted take-off
theTakeOffLandingCalculator.calculateTakeOffDistanceODE(vFailure, true);
```

Listing 5.5 Possible scenarios of calculation of the take-off distance

Since this method provides only calculations, the user may want to generate charts regarding the evolution of the state vector and of the related physical quantities during the take-off run; in this case all it has to be done is call the method `createTakeOffCharts` as shown below.

```
// Generates all the output charts
theTakeOffLandingCalculator.createTakeOffCharts();
```

Listing 5.6 Take-off charts creation

Following the flowchart of figure 5.8, if the user wants, instead, to calculate the *balanced field length*, and plot its chart, it's still very easy; in fact he has only to call the two methods `calculateBalancedFieldLength` and `createBalancedFieldLengthChart`.

```
// Calculation of the balanced field length
theTakeOffLandingCalculator.calculateBalancedFieldLength();
// Plot of the balanced field length chart
theTakeOffLandingCalculator.createBalancedFieldLengthChart();
```

Listing 5.7 Balanced field length calculation and plot

Regarding the landing, if the user wants to calculate the total landing distance, he has only to call for the method `calculateLandingDistance`, giving as input the wanted `phiRev`; this will then call the methods `calculateAirborneDistance` and `calculateGroundRollLandingODE` as described in the flowchart of figure 5.9. Also in this case the previous call is used only to perform calculations so that, if the user wants to plot the results, the call that has to be made is to the method `createLandingCharts`. The choice of separating the charts creation from the calculations derives from the willing of not to slow down these latter; in particular, because charts creation requires some time, a future iterative use of the calculation method will last longer if the method has to generate charts every time, with a negative effect upon a future optimization process.

```
// Calculation of the landing distance and its components
theLandingCalculator.calculateLandingDistance(phiRev);
// Results plot
theLandingCalculator.createLandingCharts();
```

Listing 5.8 Landing calculations and charts plot

In conclusion, the following pages show, firstly, a summary of take-off charts created in the three conditions of the listing 5.5 (by choosing a `vFailure` of 30 $\mathrm{m\,s^{-1}}$); after that, the output charts of the landing test will be reported in order to comment the main results. Instead, in the listings below, are resumed the main results of the take-off AOE condition (together with the results of the *balanced field length* calculation) and of the landing analysis.

```
CLmaxTO = 2.0212498531249747
CL0 = 1.0741952614143324
CLground = 1.2573301946144486
VsTO = 54.7262494000153 m/s
VRot = 57.4625618700161 m/s
VLO = 60.1988743400169 m/s
-----------------------------------------------------------
    END OF GROUND ROLL PHASE
  switching function changes sign at t = 25.93720130952948
  x[0] = s = 812.0430268595399 m
  x[1] = V = 57.46256187001608 m/s
  x[2] = gamma = 0.0 deg
  x[3] = altitude = 0.0 m
  COLLECTING DATA AT THE END OF GROUND ROLL PHASE ...
---------------------------DONE!-----------------------------
    END OF ROTATION PHASE
  x[0] = s = 979.678510543621 m
  x[1] = V = 61.724444902473444 m/s
  x[2] = gamma = 0.0 deg
  x[3] = altitude = 0.0 m
  t = 28.748792893777576 s
  COLLECTING DATA AT THE END OF ROTATION PHASE ...
---------------------------DONE!-----------------------------
    BEGIN BAR HOLDING
  CL = 1.7189121346068004
  Alpha Body = 5.041413155519889 deg
  t = 29.999717401303357 s
---------------------------DONE!-----------------------------
    END BAR HOLDING
  switching function changes sign at t = 30.499717401303354
---------------------------DONE!-----------------------------
    LOAD FACTOR = 1 IN CLIMB
  t = 31.536083011364767 s
---------------------------DONE!-----------------------------
    END OF AIRBORNE PHASE
  switching function changes sign at t = 33.75129940331029
  x[0] = s = 1302.3695981924898 m
  x[1] = V = 66.79007719128757 m/s
  x[2] = gamma = 2.707958902096113 deg
  x[3] = altitude = 10.668000000000006 m
  COLLECTING DATA AT THE END OF AIRBORNE PHASE ...
---------------------------DONE!-----------------------------
BALANCED FIELD LENGTH = 1863.27447504457 m
Decision Speed (V1/VsTO) = 1.04447828088380
---------------------------END!!-----------------------------
```

Listing 5.9 ATR-72 take-off test results

```
CLmaxLanding = 2.9499357749411175
CL0 = 1.5390863614594144
CLground = 1.7150043879106331
VsLanding = 45.3001255420356 m/s
```

```
V_Approach = 58.890163204646 m/s
V_Flare = 55.719154416704 m/s
V_TouchDown = 52.0951443733410 m/s
-----------------------------------------------------------
    END OF THE LANDING GROUND ROLL
  switching function changes sign at t = 24.192677800708584
  x[0] = s = 1071.2865561533845 m
  x[1] = V = -1.0615119894197278E-13 m/s
---------------------------DONE!-------------------------------



-----------------------RESULTS---------------------------
AIRBORNE DISTANCE = 249.403167762090 m
FLARE DISTANCE = 82.8435167233407 m
GROUND ROLL DISTANCE = 739.03987166795 m
TOTAL LANDING DISTANCE = 1071.28655615338 m
-----------------------------------------------------------
```

Listing 5.10 ATR-72 landing test results

As can be seen the take-off distance required in AOE condition is about 1302 m, at the maximum take-off mass of 23063,579 kg and at sea level in standard atmospheric conditions, within a time of about 33,750 s; while the *balanced field lenght* is bigger than the AOE take-off distance of about the 30%. Moreover the *decision speed* V_1 is bigger than the stalling speed V_s and lower than the rotation speed V_{Rot} as required by the FAR-25 from table 5.1. These results are almost the same shown in the aircraft brochure available on the Internet, so that the calculation carried out results correct.

Regarding the landing test, the results obtained show that, at a maximum landing mass of 20757,221 kg and at sea level in standard atmospheric conditions, the required landing distance is around 1071 m with a ground roll distance of about 739 m covered in 24 s. As expected the landing distance is lower than the take-off distance and is almost equal to the one proposed onto the ATR-72 brochure proving that the calculation method works.

Finally, with reference to the landing charts at the end of the chapter (from figure 5.42 to 5.49), the reader can easly recognize the effect of the reverse thrust upon the landing distance; in particular, in case of `phiRev`=0.25, the deceleration is bigger so that the speed decreases faster making the total landing distance shorter. It has also to be noted that during the air run, the speed is assumed to be constant during the approach phase for then decrease, with a linear trend, from the V_A to the V_{TD} during the flare phase.

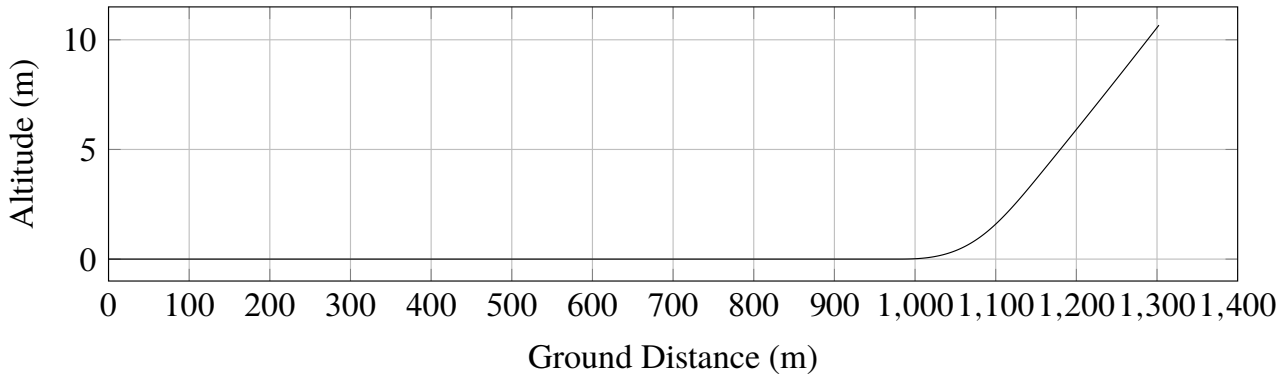

Figure 5.10 Take-off trajectory in AOE condition - ATR-72

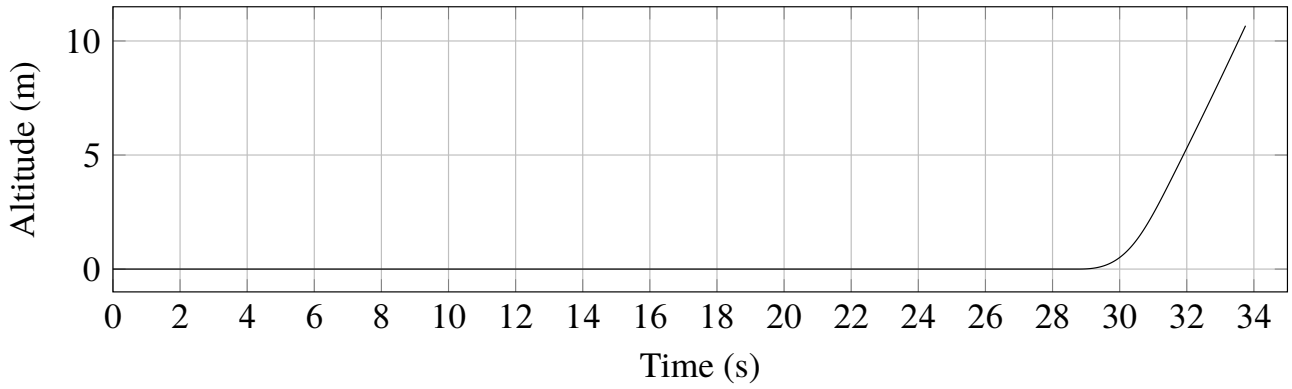

Figure 5.11 Altitude evolution in AOE condition - ATR-72

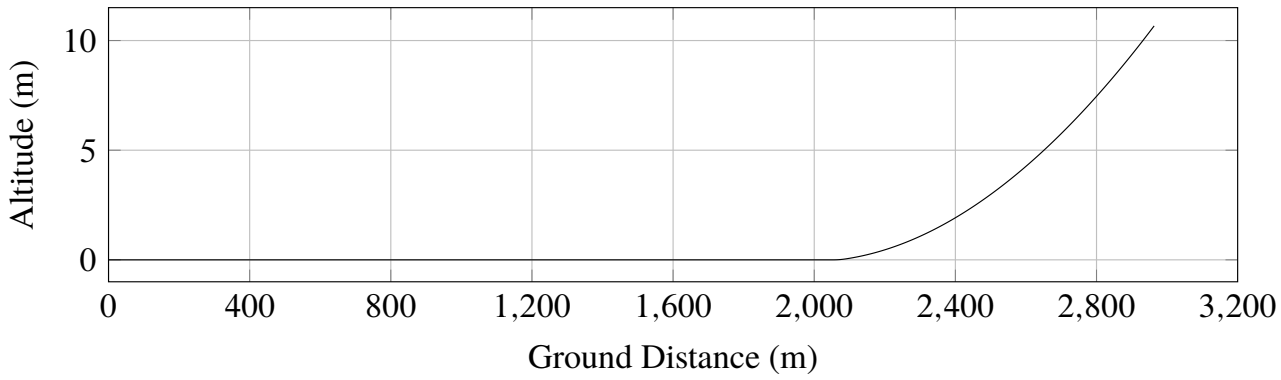

Figure 5.12 Take-off trajectory in OEI condition - ATR-72

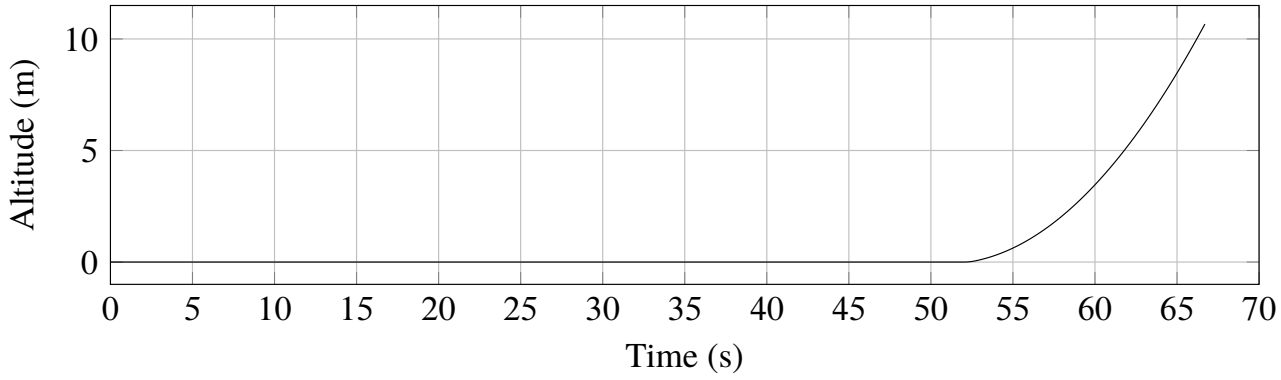

Figure 5.13 Altitude evolution in OEI condition - ATR-72

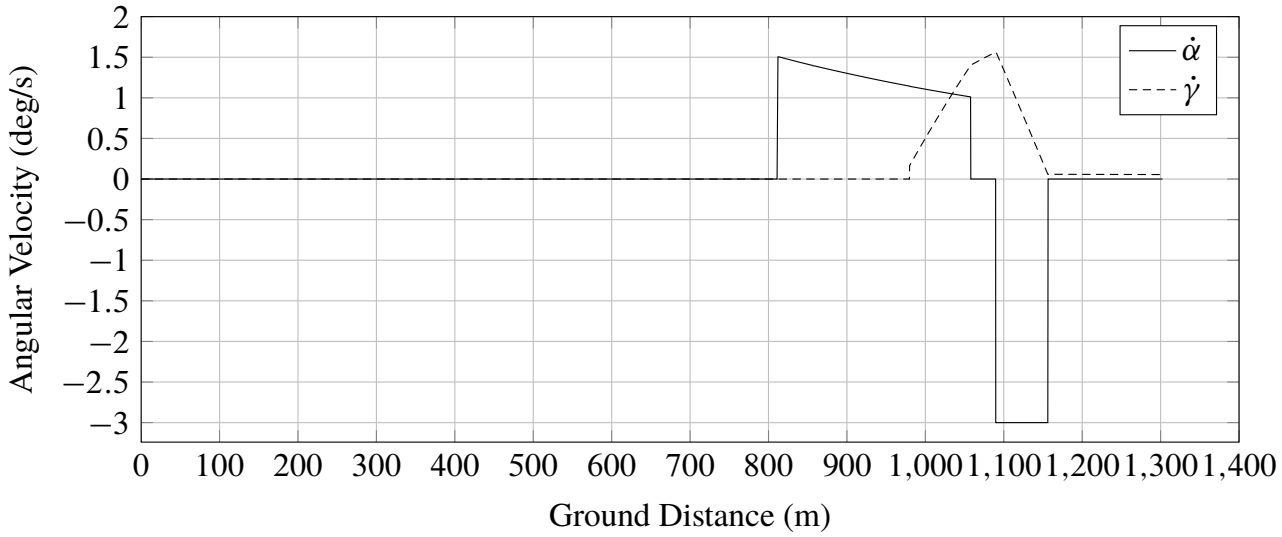

Figure 5.14 Angular velocities v.s. ground distance in AOE condition - ATR-72

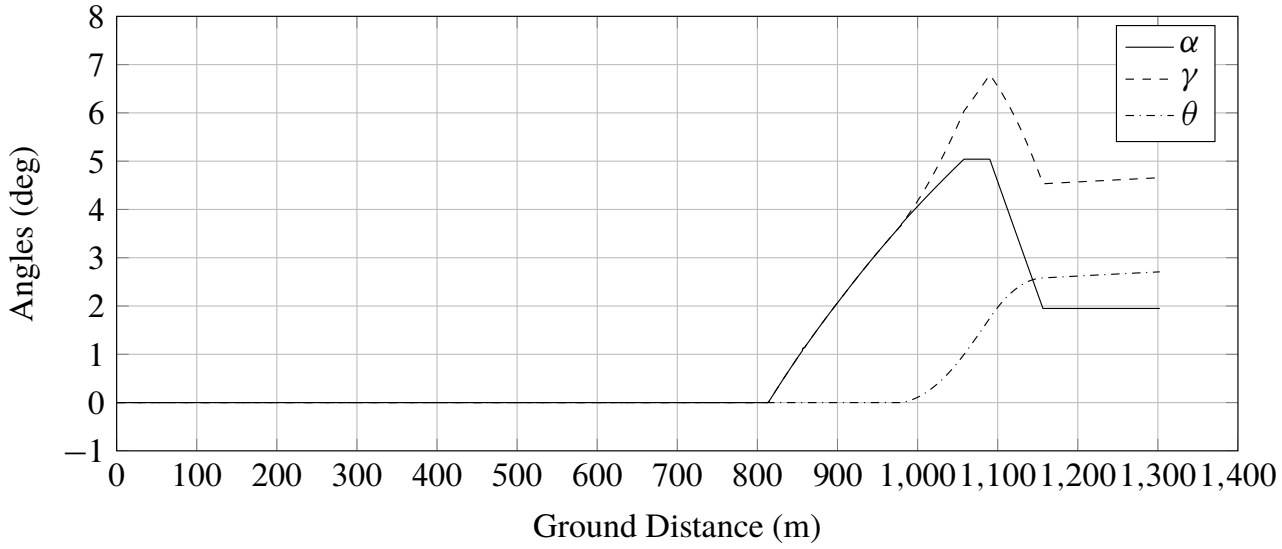

Figure 5.15 Angles v.s. grpund distance in AOE condition - ATR-72

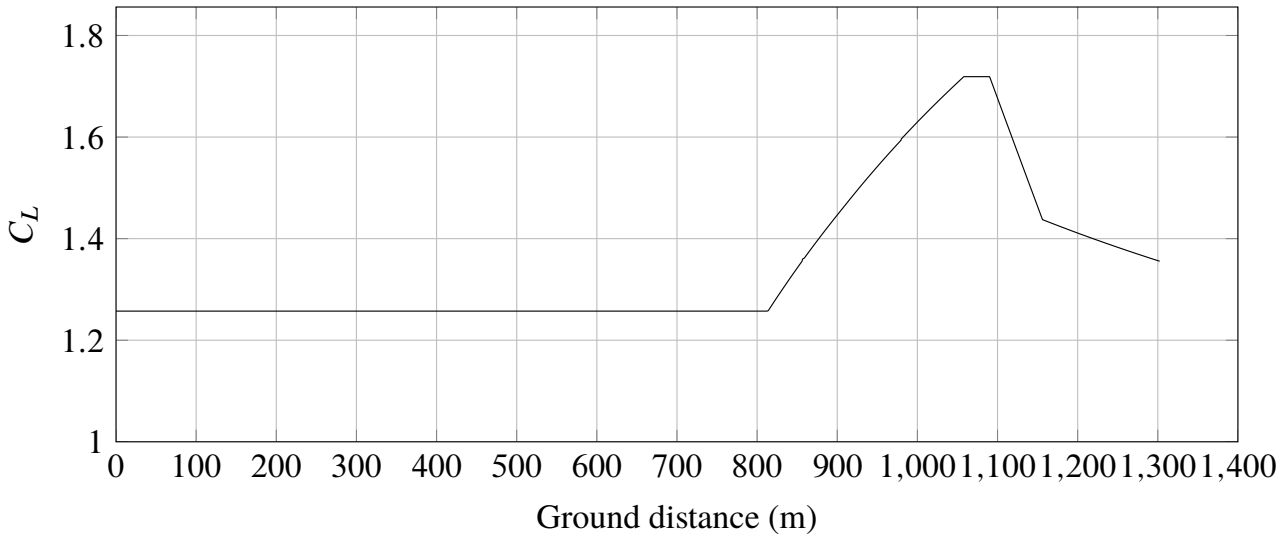

Figure 5.16 C_L v.s. ground distance in AOE condition - ATR-72

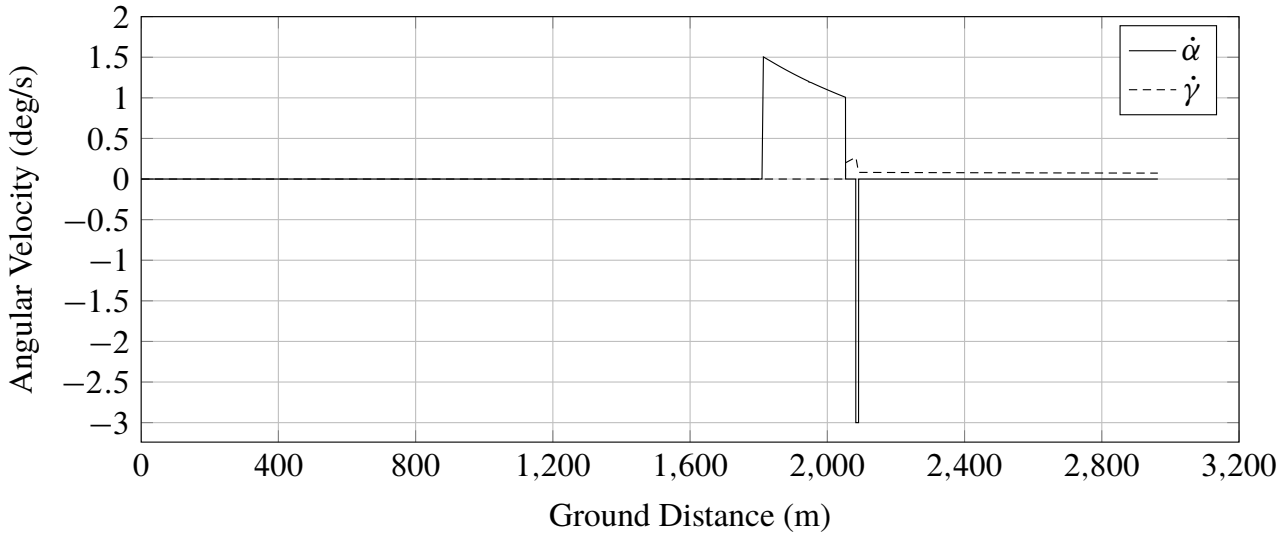

Figure 5.17 Angular velocities v.s. ground distance in OEI condition - ATR-72

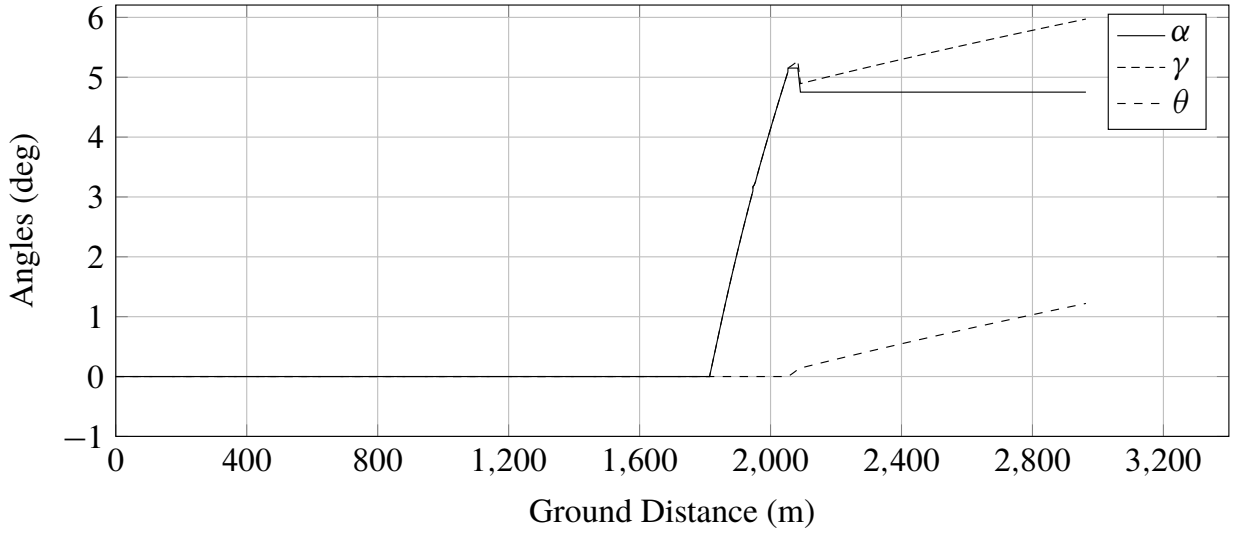

Figure 5.18 Angles v.s. ground distance in OEI condition - ATR-72

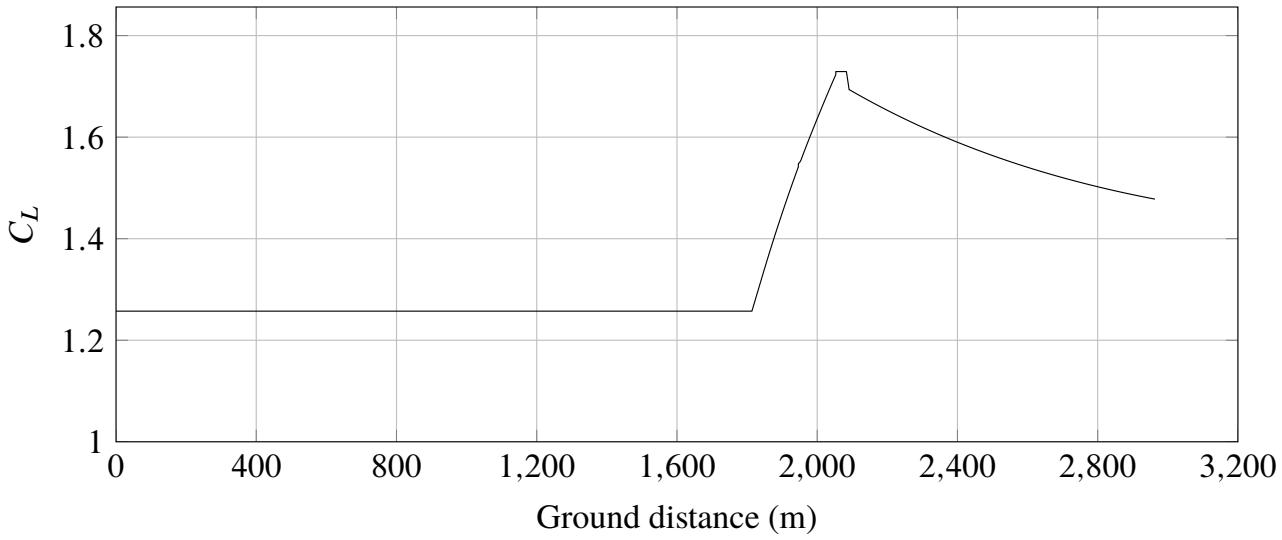

Figure 5.19 C_L v.s. ground distance in OEI condition - ATR-72

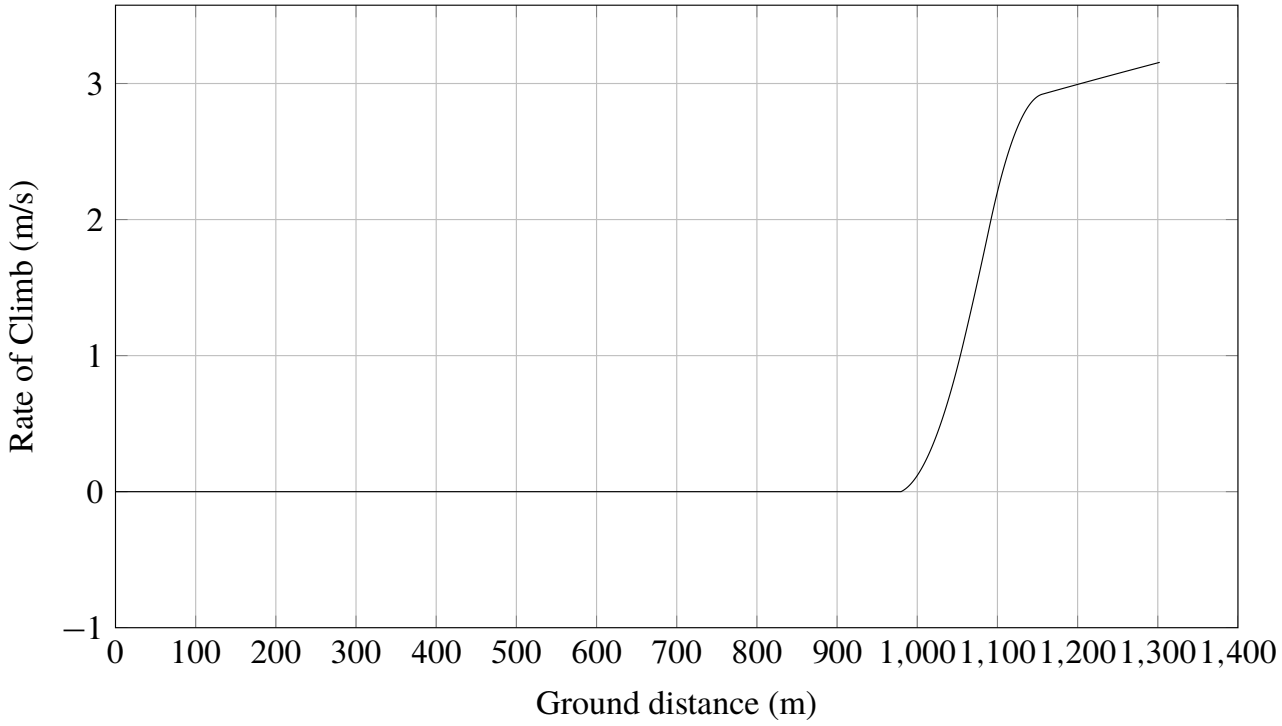

Figure 5.20 Rate of climb in AOE condition - ATR-72

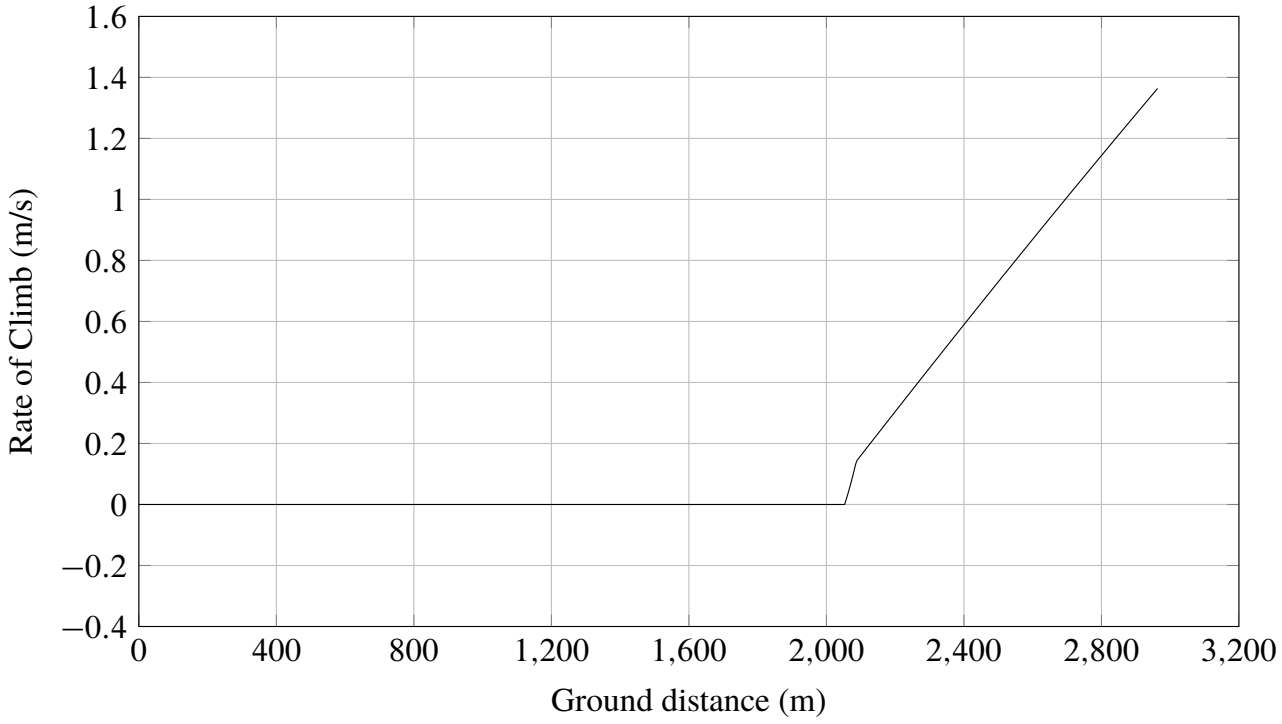

Figure 5.21 Rate of climb in OEI condition - ATR-72

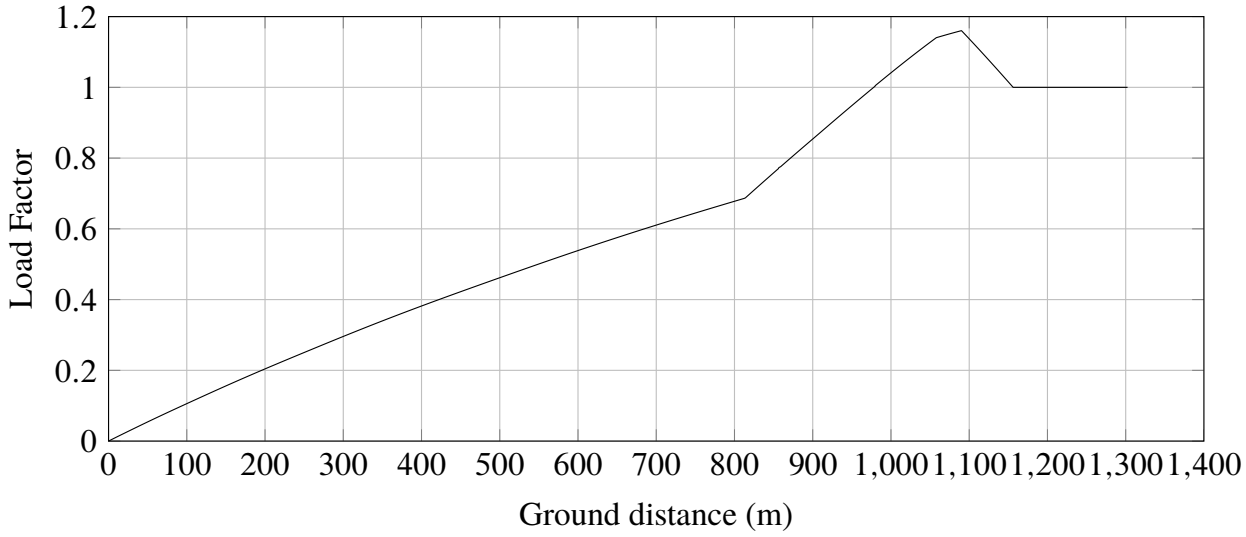

Figure 5.22 Load factor in AOE condition - ATR-72

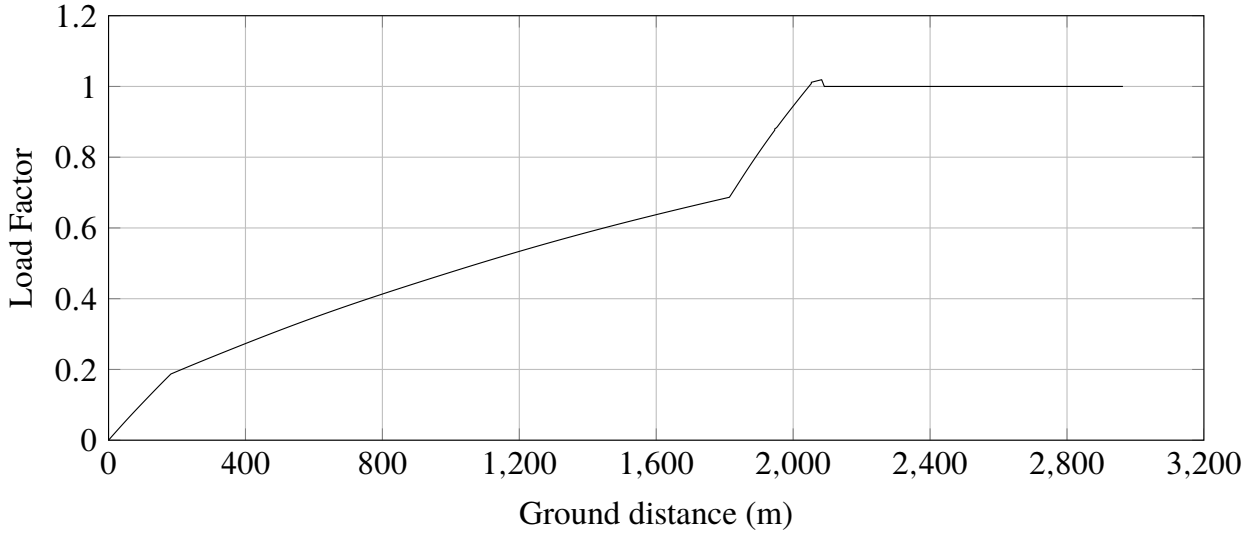

Figure 5.23 Load factor in OEI condition - ATR-72

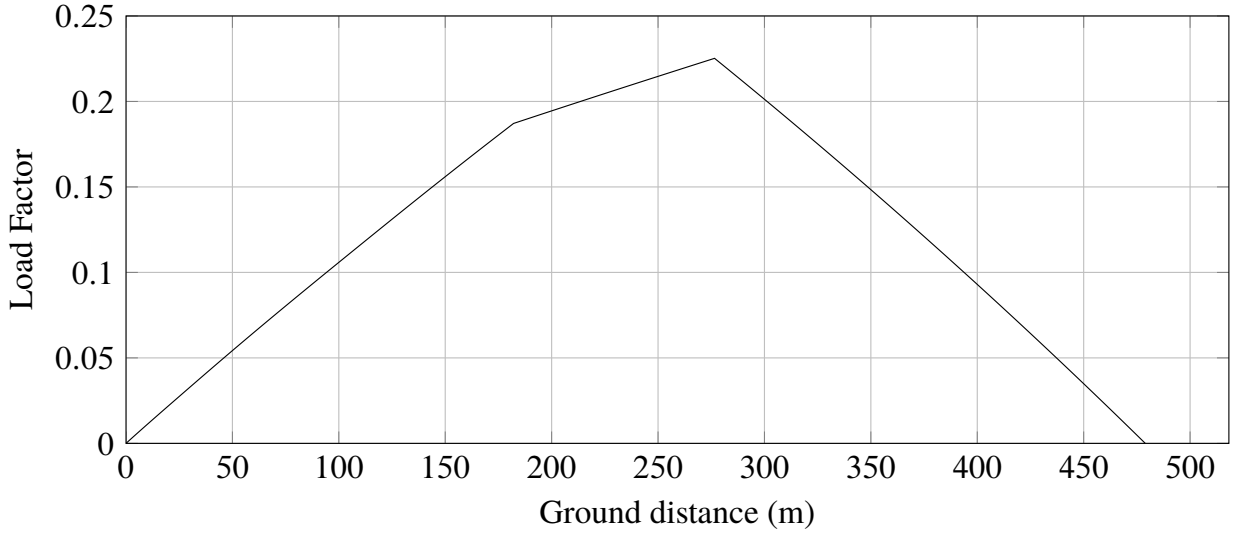

Figure 5.24 Load factor in aborted take-off condition - ATR-72

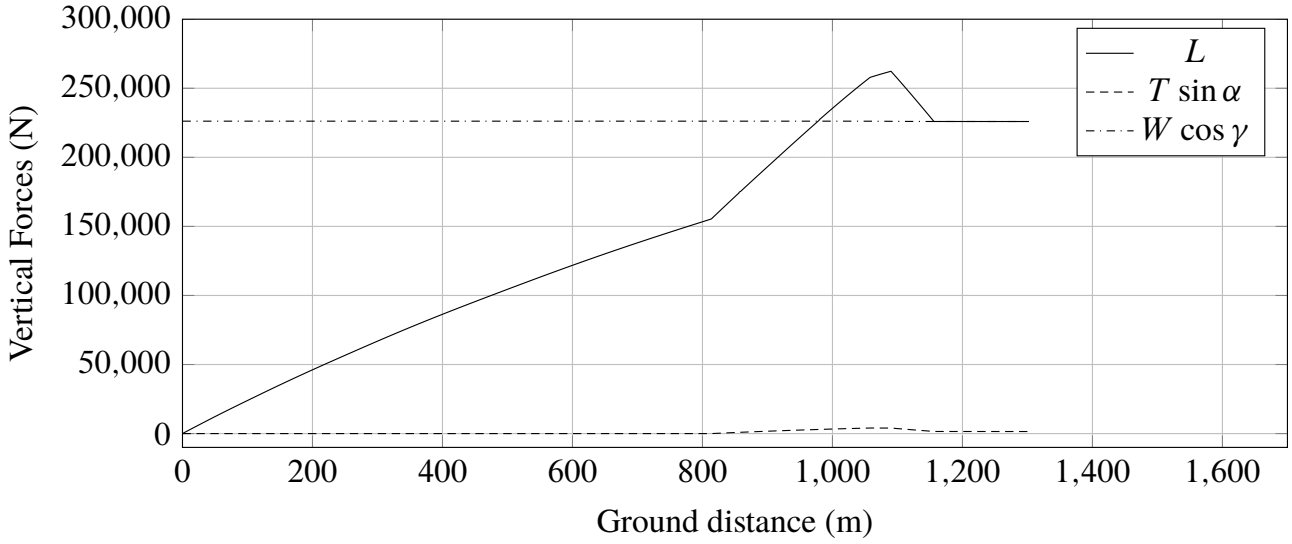

Figure 5.25 Vertical forces in AOE condition - ATR-72

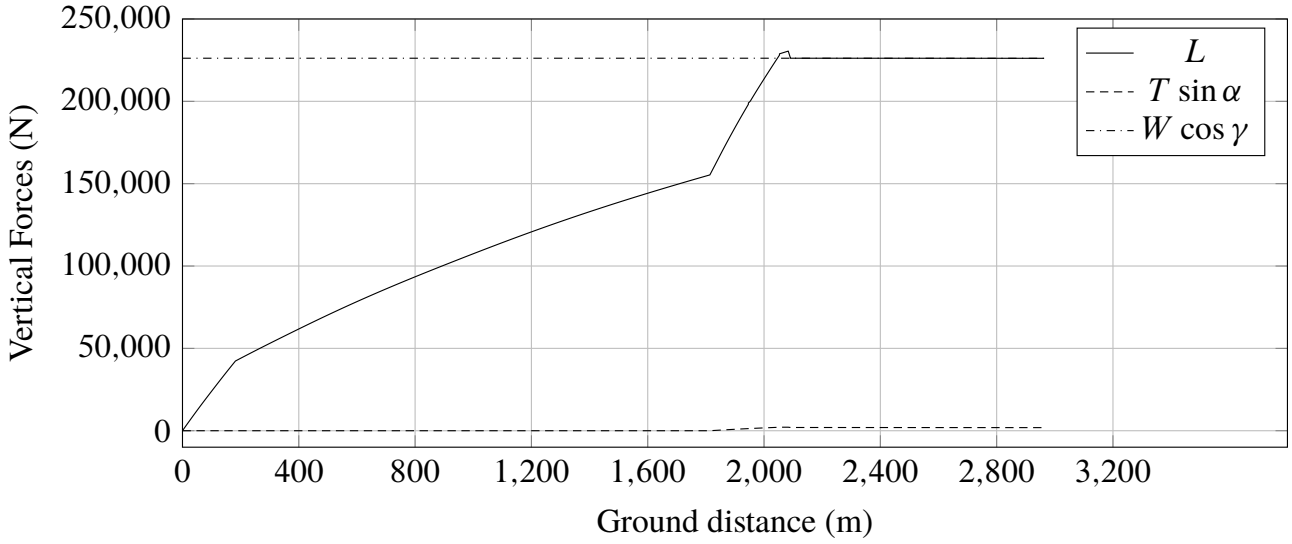

Figure 5.26 Vertical forces in OEI condition - ATR-72

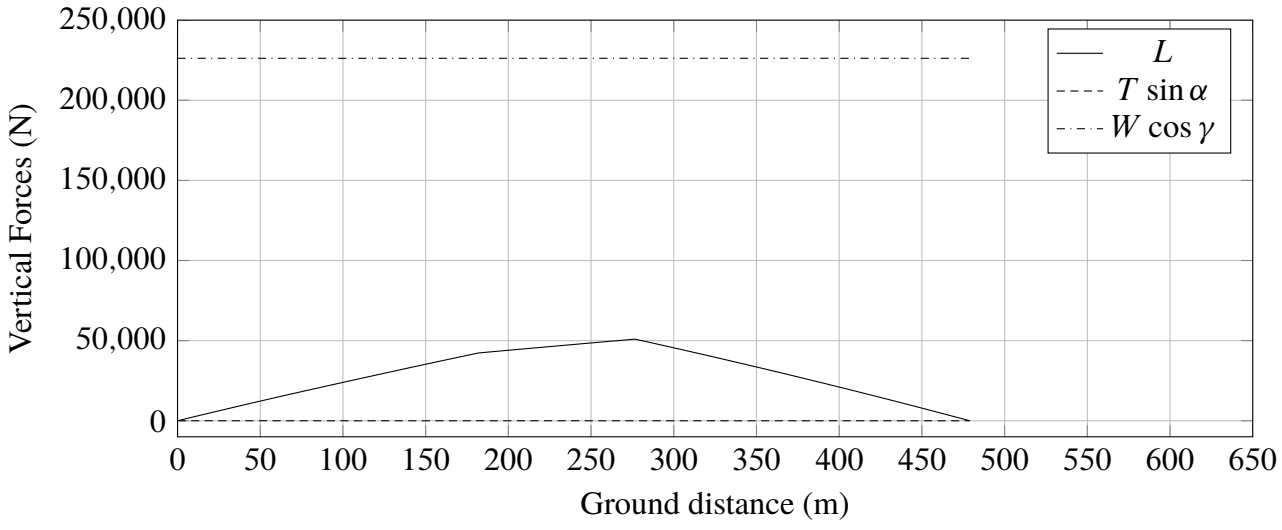

Figure 5.27 Vertical forces in aborted take-off condition - ATR-72

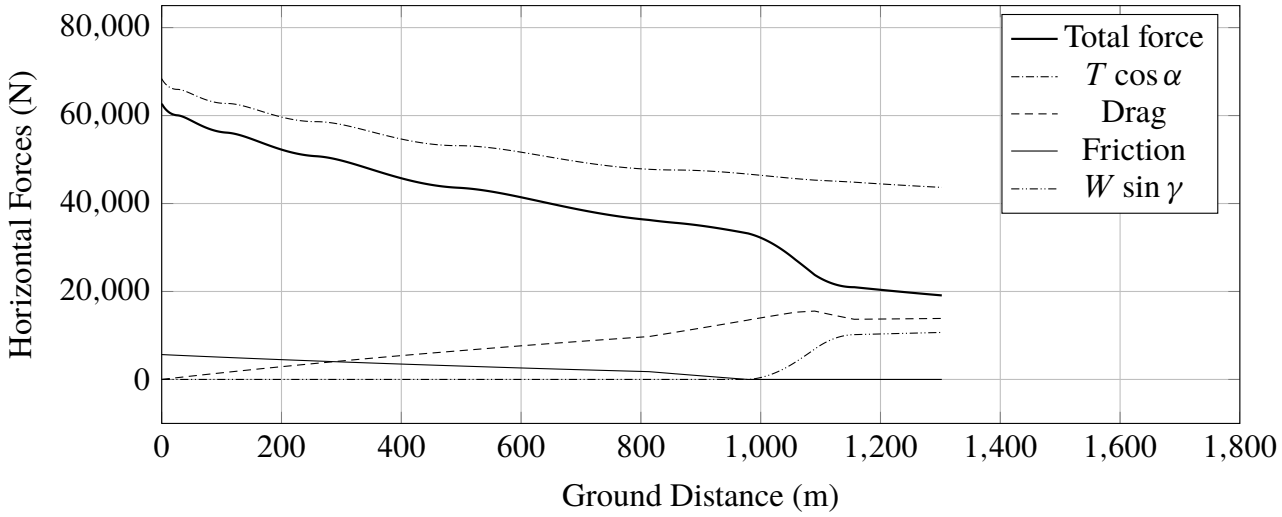

Figure 5.28 Horizontal forces in AOE condition - ATR-72

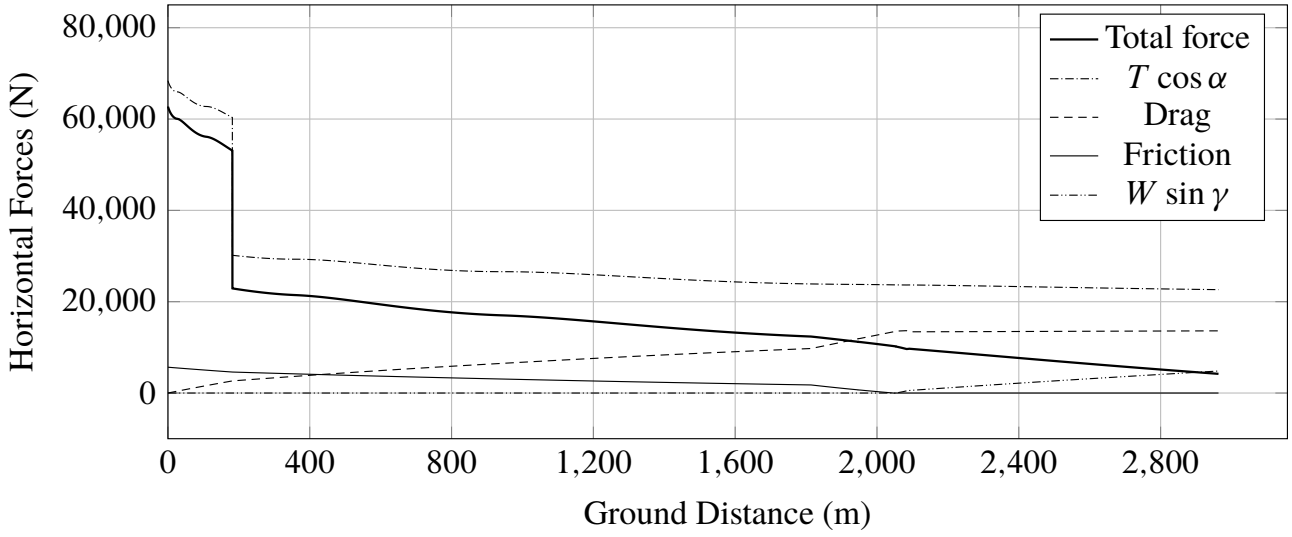

Figure 5.29 Horizontal forces in OEI condition - ATR-72

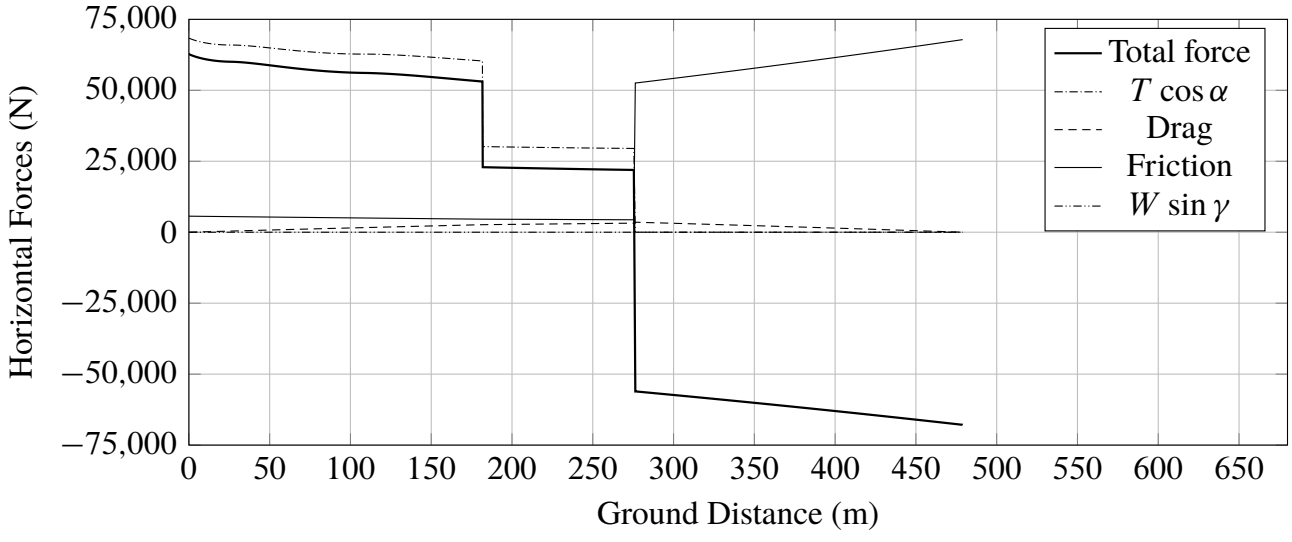

Figure 5.30 Horizontal forces in aborted take-off condition - ATR-72

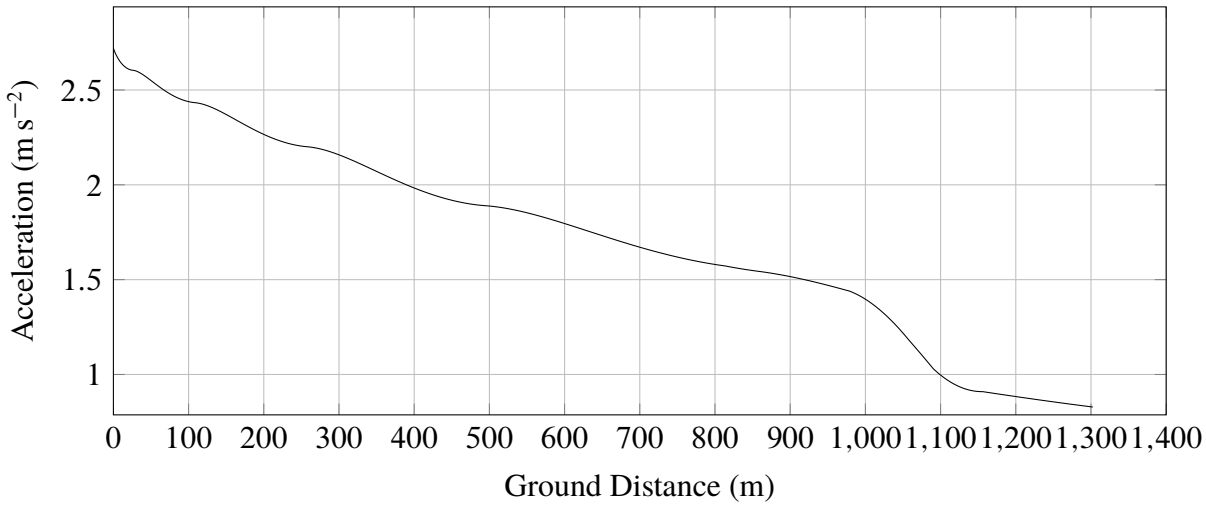

Figure 5.31 Acceleration v.s. ground distance in AOE condition - ATR-72

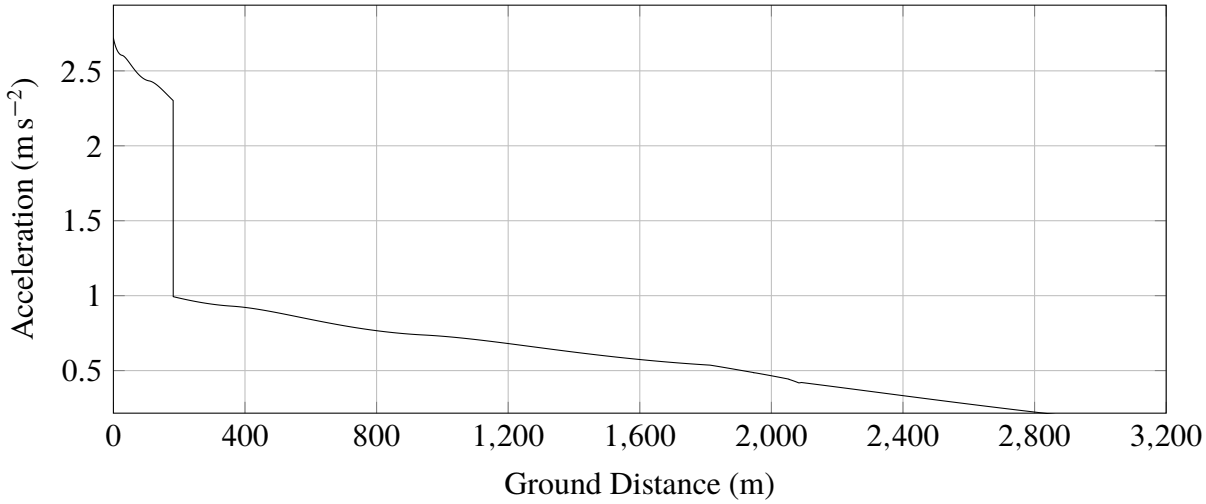

Figure 5.32 Acceleration v.s. ground distance in OEI condition - ATR-72

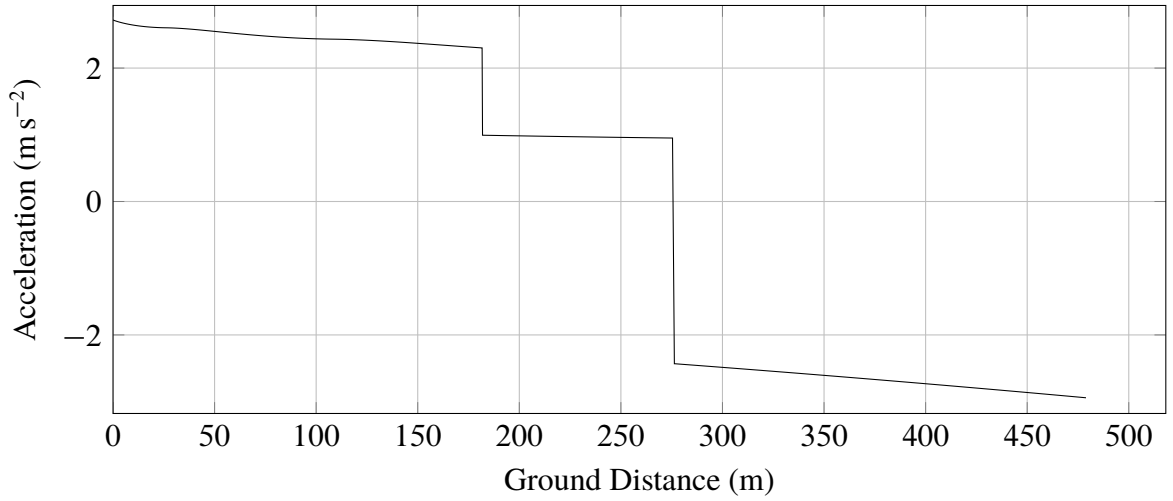

Figure 5.33 Acceleration v.s. ground distance in aborted take-off condition - ATR-72

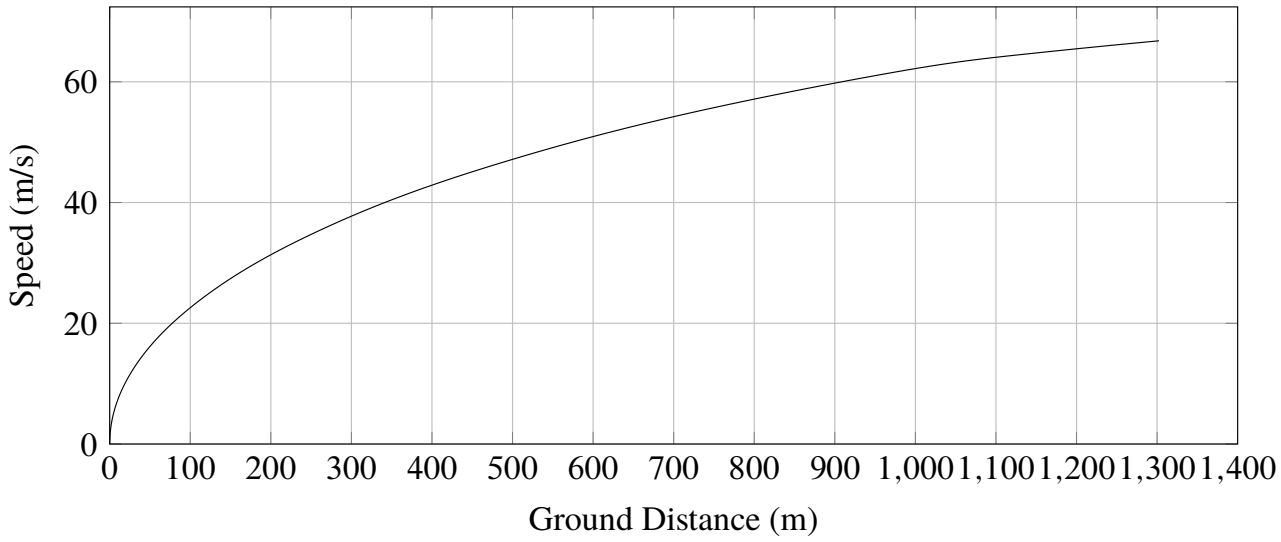

Figure 5.34 Speed v.s. ground distance in AOE condition - ATR-72

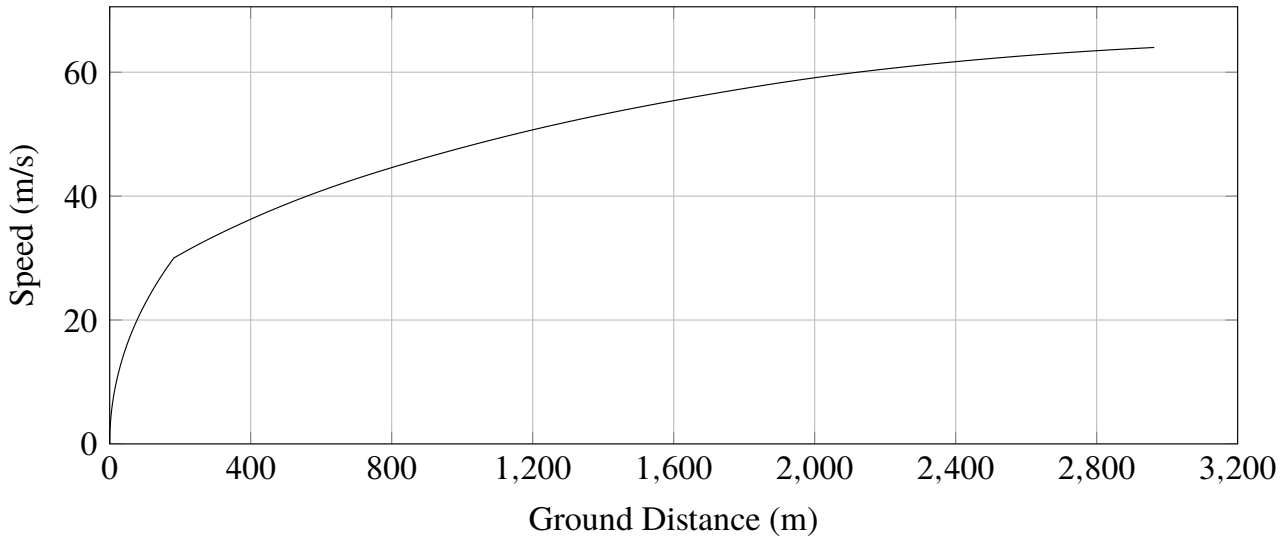

Figure 5.35 Speed v.s. ground distance in OEI condition - ATR-72

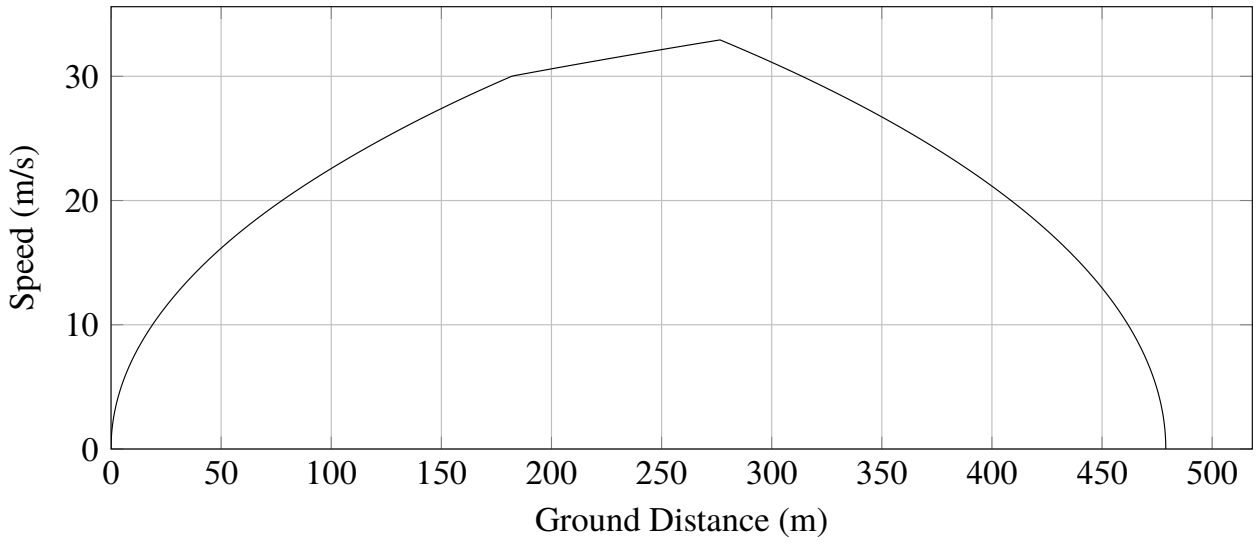

Figure 5.36 Speed v.s. ground distance in aborted take-off condition - ATR-72

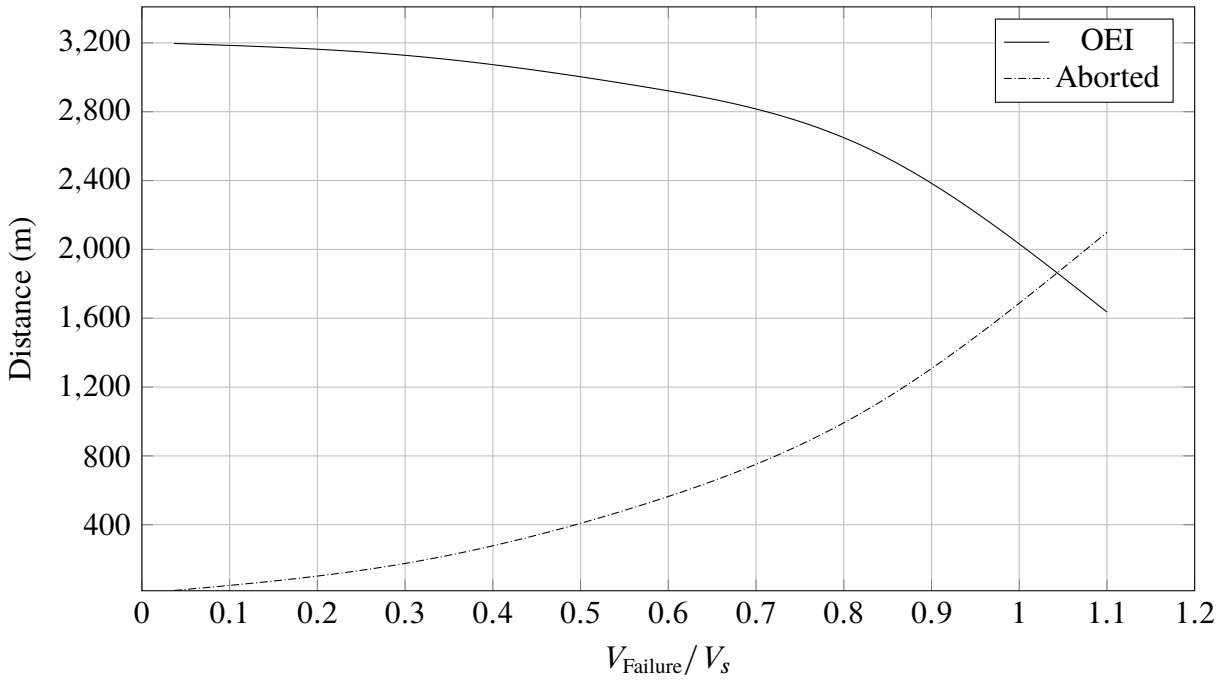

Figure 5.37 Balanced take-off length chart - ATR-72

In conclusion of the take-off analysis, it's useful to highlight the effects that k_α and α_{red} have on the take-off run. The first one influences the rotation phase making the pilot's manouver faster as it grows; in particular, from figure 5.38, a faster rotation manouver makes α to rise faster reaching the limit angle (related to the C_L limitation imposed) before with the effect of having a high drag for more time so that the take-off run is longer. The second parameter influences the airborne phase making the reduction in angle of attack faster as it grows; the result is that the load factor reaches the value of 1 later (eventually never) and so the rate of climb is higher making the airborne phase to be shorter with a lower value of the take-off run (see figures 5.39, 5.40 and 5.41).

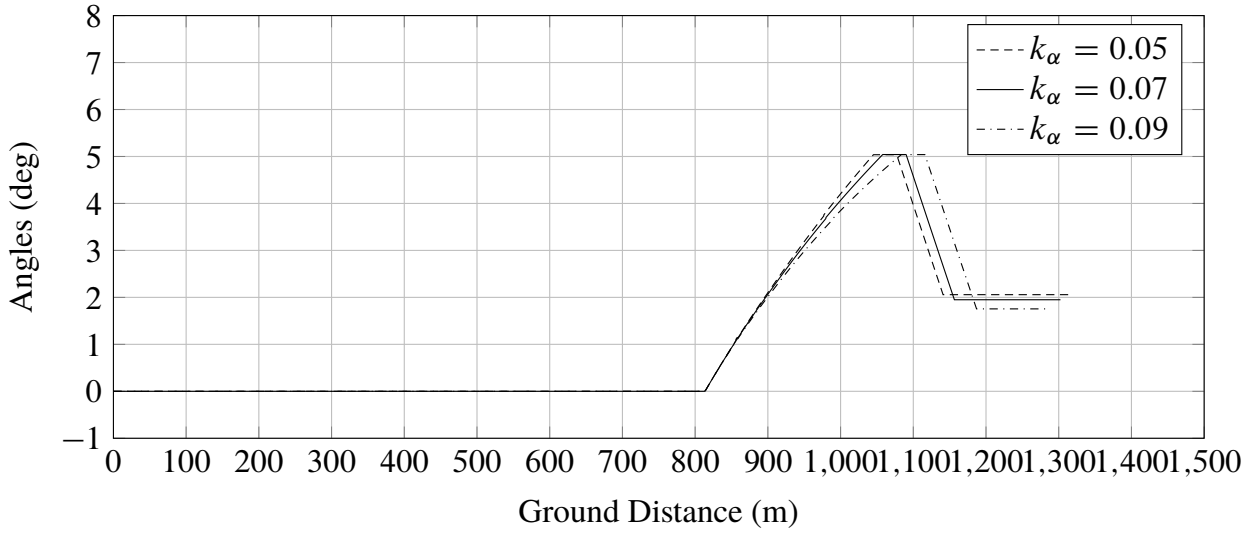

Figure 5.38 Effect of α_{red} on the C_L during the airborne phase - ATR-72

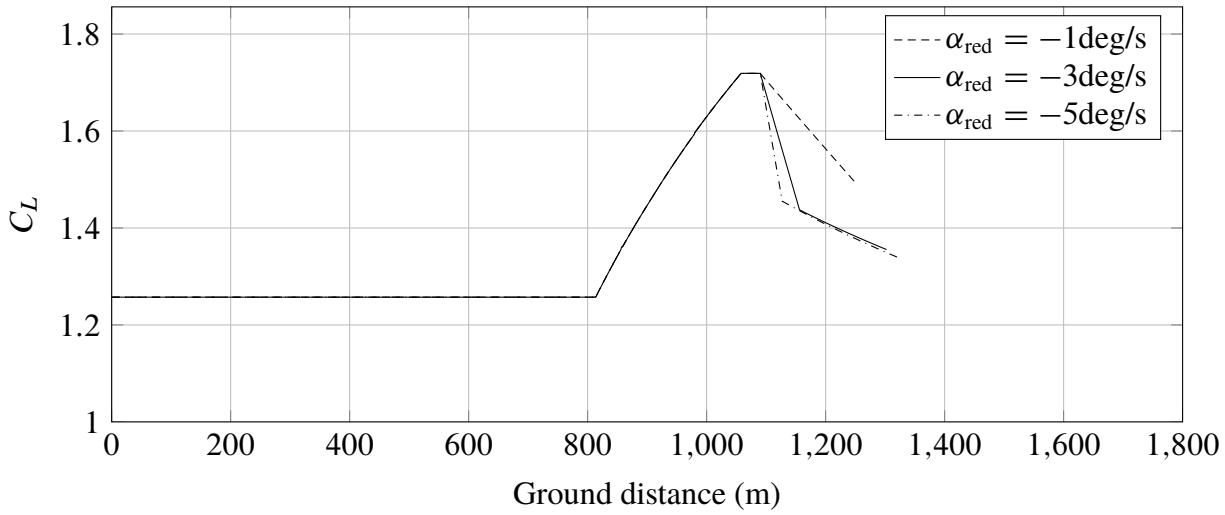

Figure 5.39 Effect of α_{red} on the load factor during the airborne phase - ATR-72

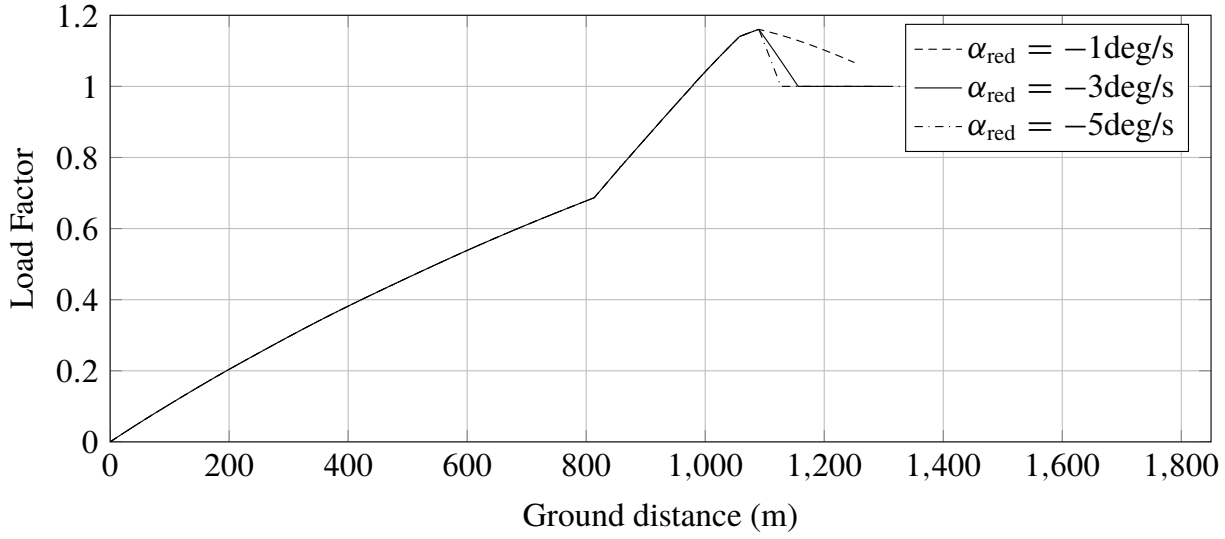

Figure 5.40 Effect of α_{red} on the rate of climb during the airborne phase - ATR-72

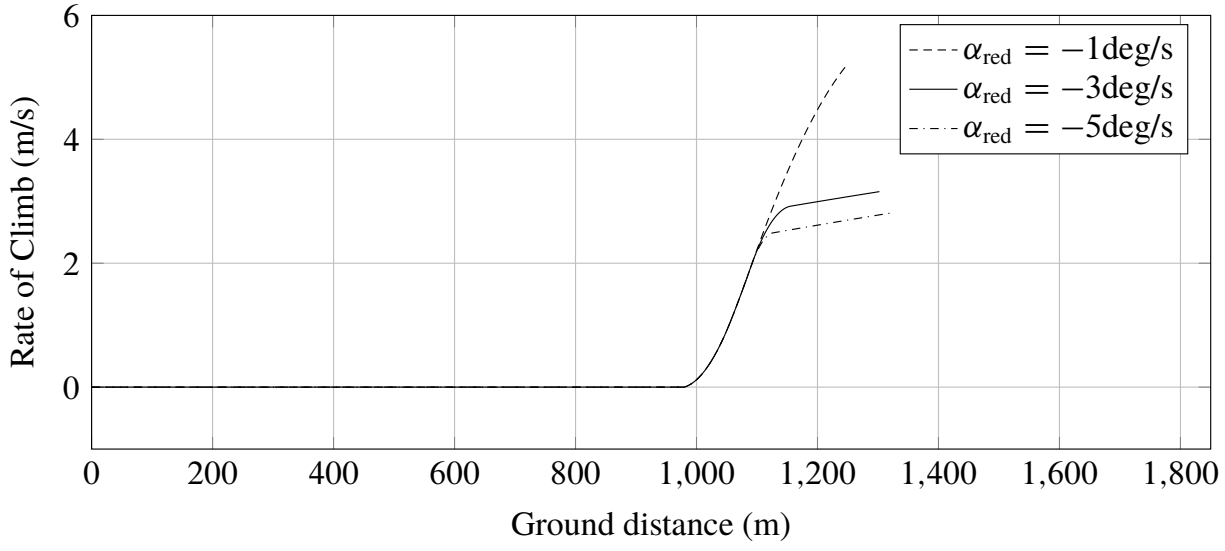

Figure 5.41 Effect of k_α on the rotation phase - ATR-72

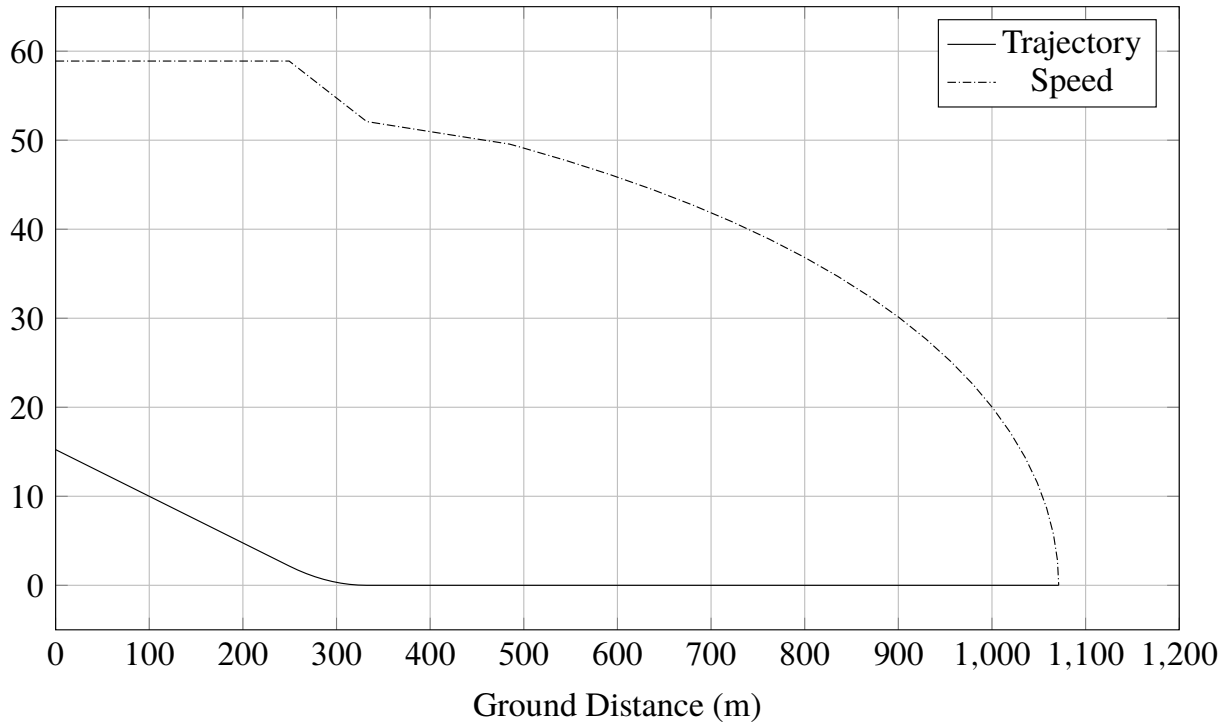

Figure 5.42 Landing trajectory and speed evolution with `phiRev`=0.0 - ATR-72

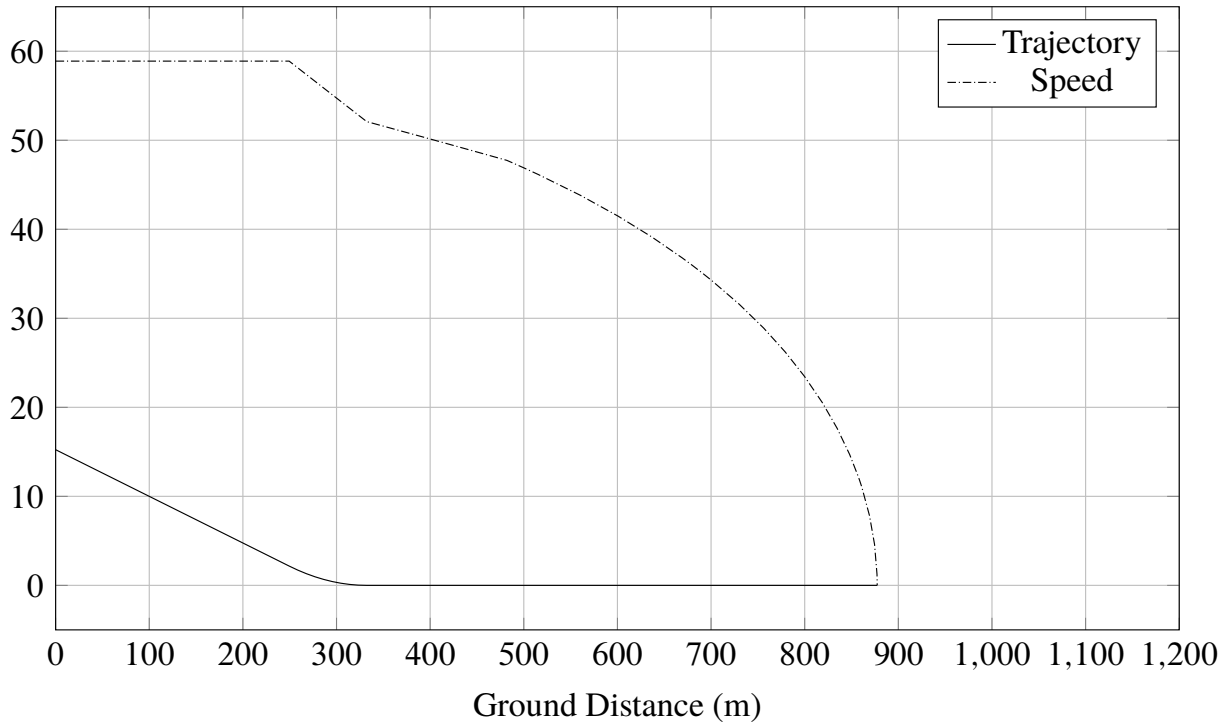

Figure 5.43 Landing trajectory and speed evolution with `phiRev`=0.25 - ATR-72

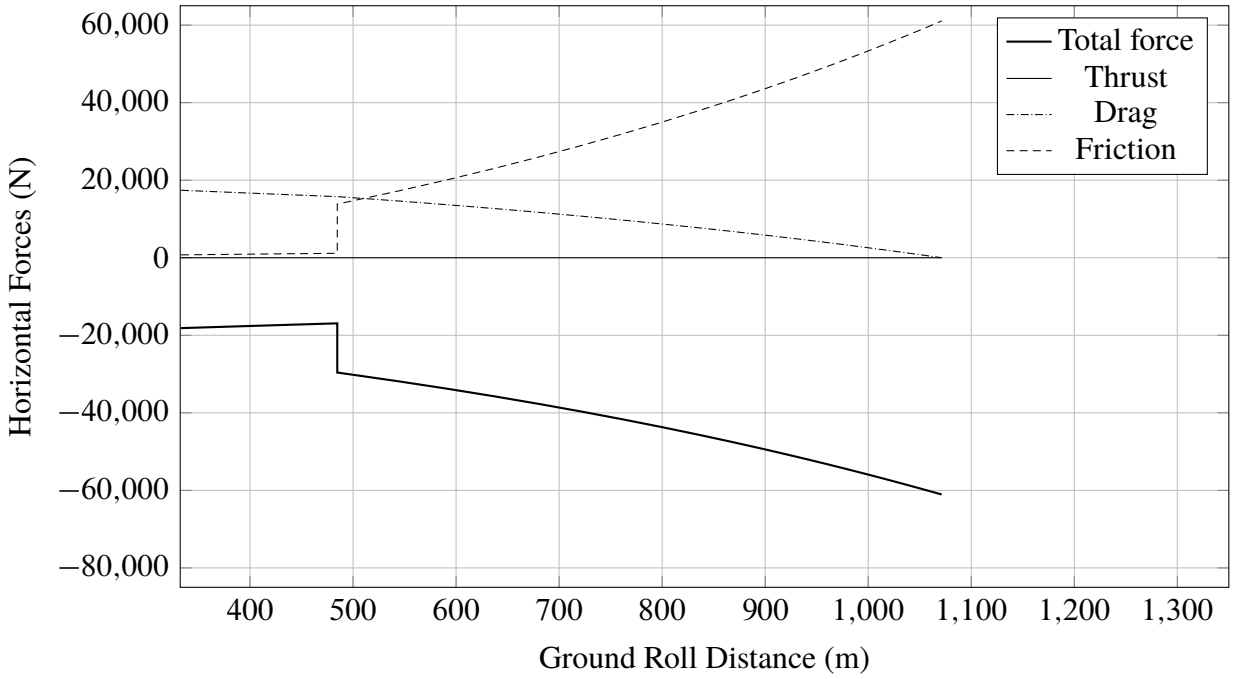

Figure 5.44 Forces evolution in landing, during the ground run, with `phiRev`=0.0 - ATR-72

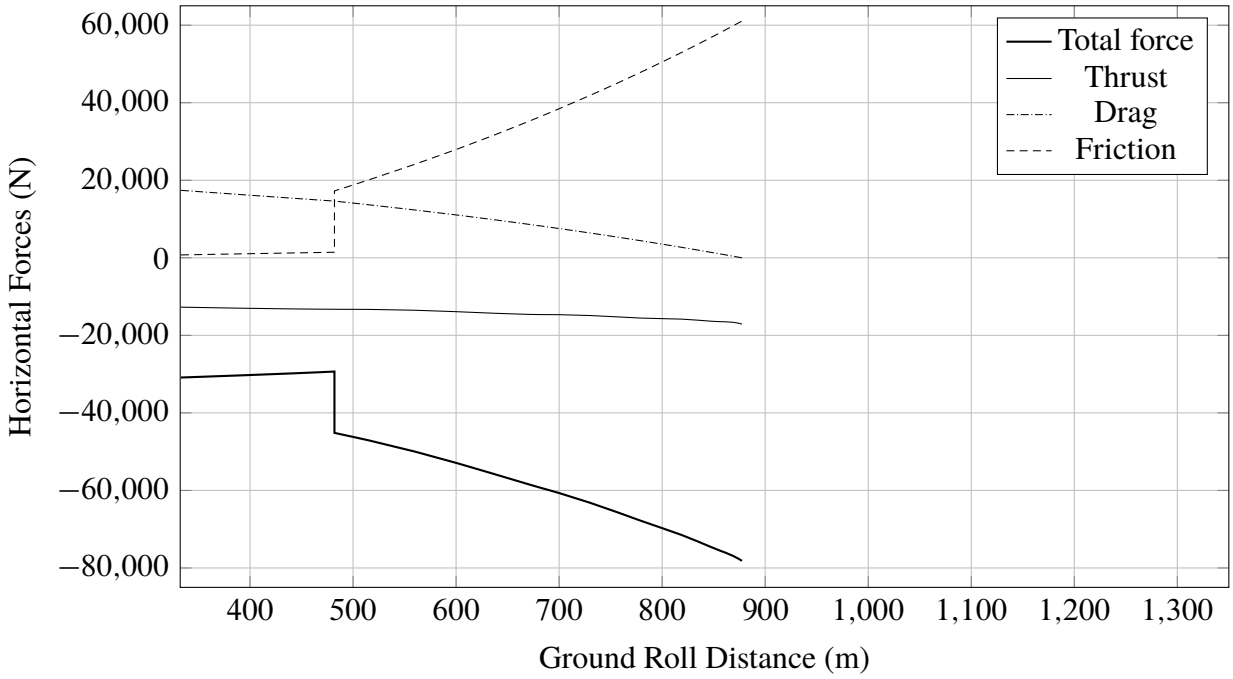

Figure 5.45 Forces evolution in landing, during the ground run, with `phiRev`=0.25 - ATR-72

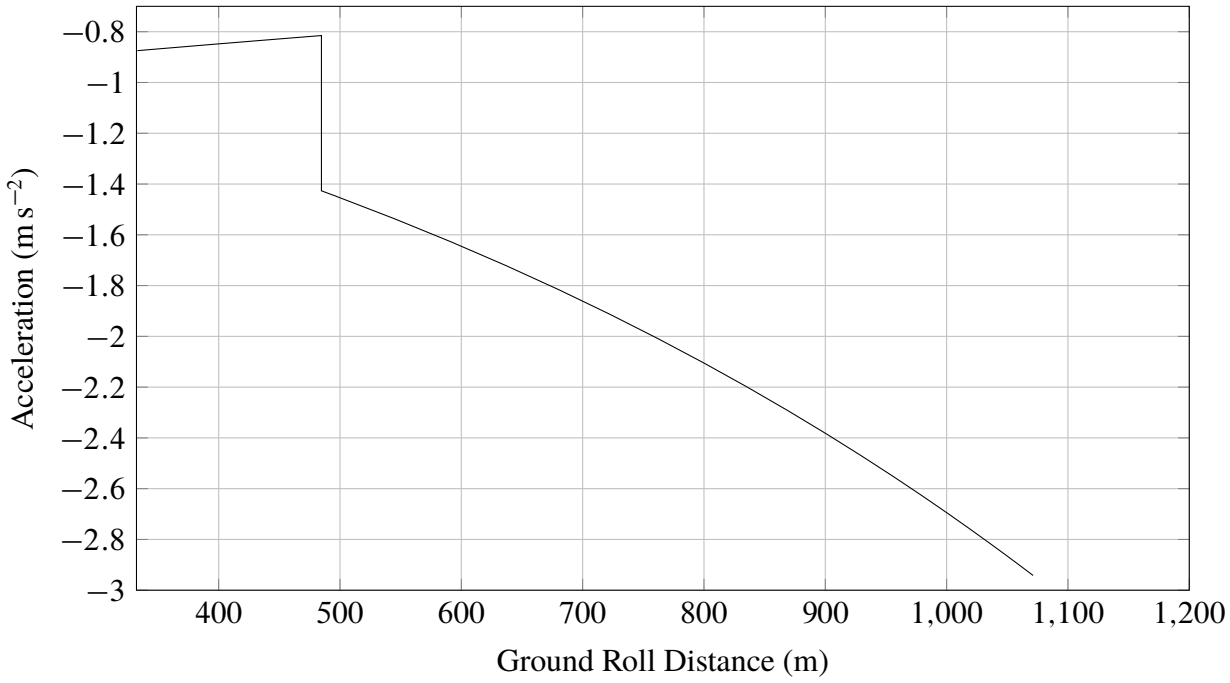

Figure 5.46 Acceleration evolution in landing, during the ground run, with `phiRev`=0.0 - ATR-72

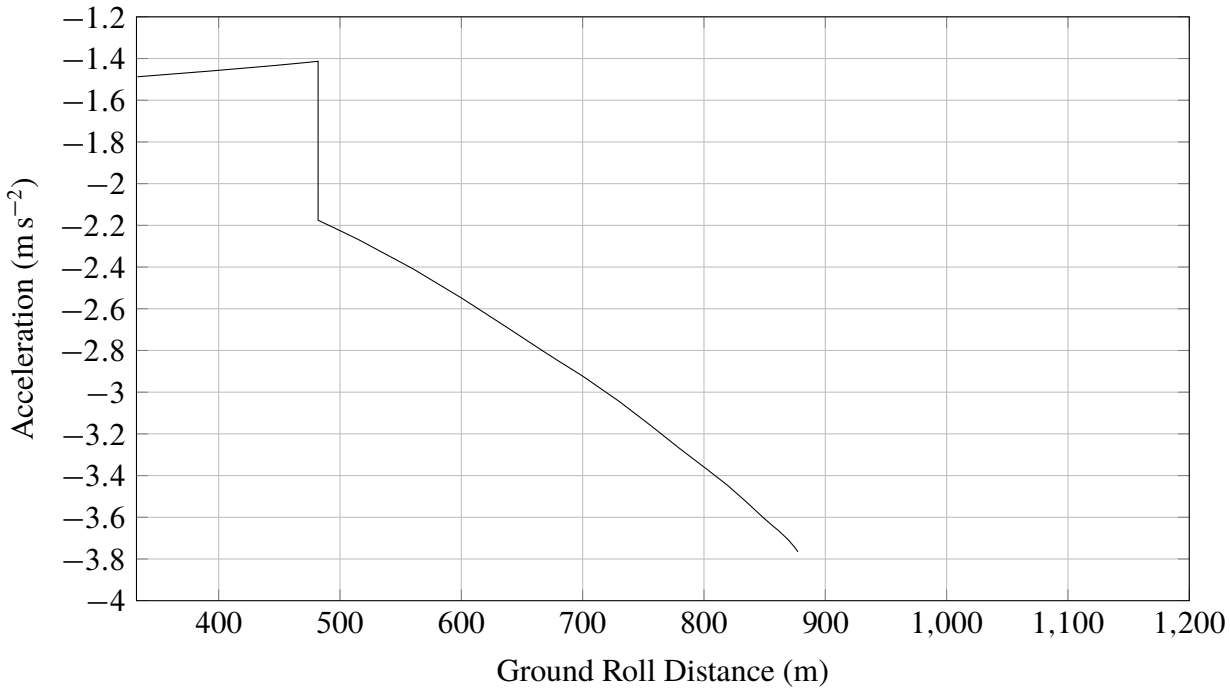

Figure 5.47 Acceleration evolution in landing, during the ground run, with `phiRev`=0.25 - ATR-72

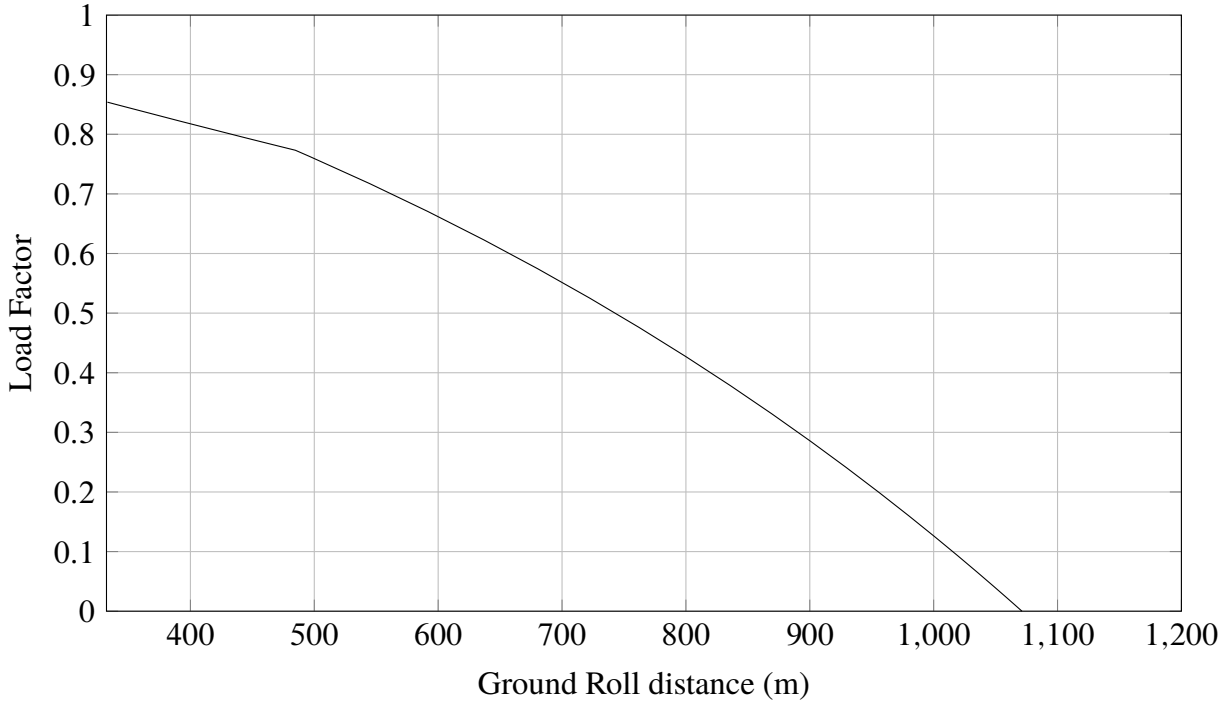

Figure 5.48 Load factor evolution in landing, during the ground run, with `phiRev`=0.0 - ATR-72

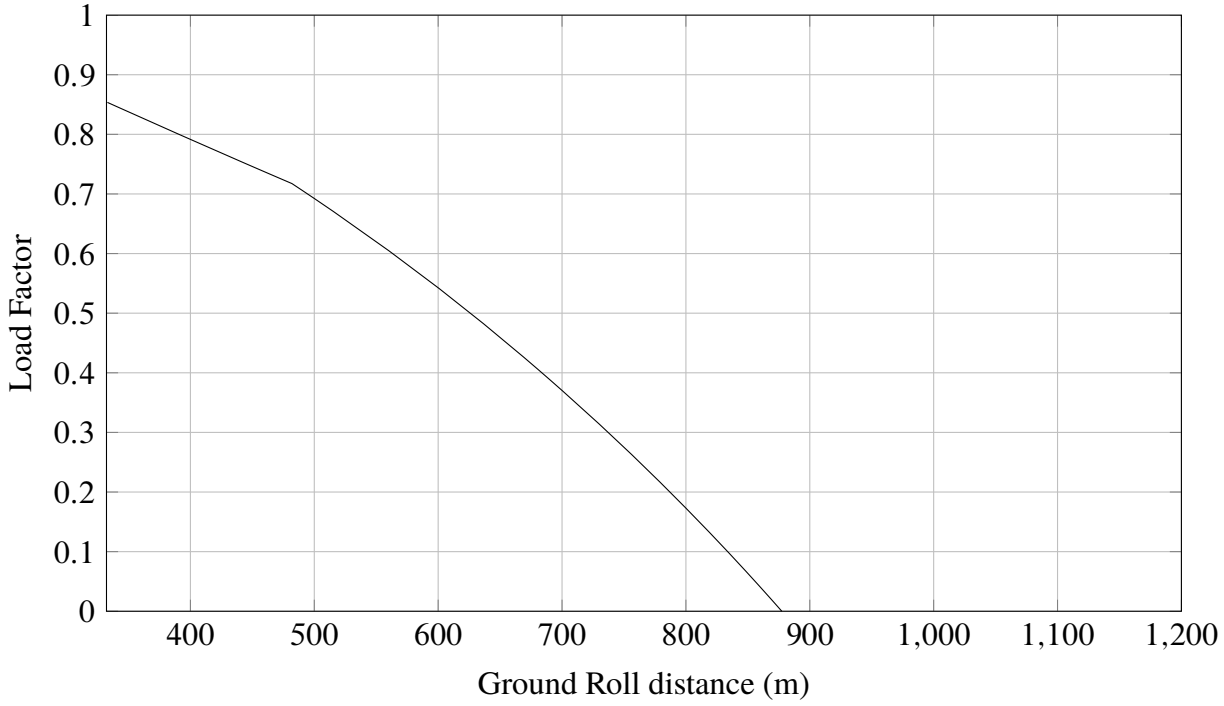

Figure 5.49 Load factor evolution in landing, during the ground run, with `phiRev`=0.25 - ATR-72

Conclusions

The JPAD library implements, indeed, a lot of functionalities and is almost complete in terms of analysis types. The present version of the library can perform a complete aircraft analysis in few seconds, including weights, balance, aerodynamics, costs and performance analysis; but some work has still to be done.

In particular the definition of a comprehensive and user-friendly input file structure is in development, following the guideline described in paragraph 1.2; moreover the library doesn't have yet a working module for the optimization process (see paragraph 1.6) which will be subject of future developments once all main features will be completed (for example the lateral-directional stability module or the landing distance calculation). Last, but not least, is the creation of a smart and pleasant GUI using as baseline the one described in paragraph 1.5; this feature has not to be underestimated since can provide significantly benefits to the user like a simply way of managing and view results as well as managing many aircrafts, or different versions of the same one, at the same time making easy to compare their characteristics.

Appendices

Appendix A

HDF DATABSE CREATION AND READING

A.1 Creation of a database using MATLAB

Creation and mangment of an HDF Dataset are very important to handle because they allow to generate resources which are required by a lot of analysis; for example by using this datasets it's possible to implement a new engine type allowing the analysis of the preformance of a new aircraft. This feature has not been implemented inside JPAD with the purpose of being able to generate the required resources independently. For more information regarding the HDF, the reader can refer to [16].

First of all it's necessary to have curves of the database that has to be digitized; then, with the use of sotware like *PlotDigitizer*, it's possible to acquire them using a finite number of points chosen by the user. The output of this procedure is a *.csv* file containing all the copule of points which have been used to digitize the specific curve. Now the matlab code comes in play to manage these data and to generate the digitized curves and the HDF dataset. In the example reported there are four curves defined by points through *PlotDigitizer* which have, firstly, been imported in MATLAB generating four *.mat* files; at this point the code interpolates curves points with cubic splines in order to have more points to plot for each curve. Finally curves are plotted and the HDF Dataset is populated by using *h5create* and *h5write*; in particular curves points, abscissas and parameterization values are attached to the h5 file through these commands.

```
1 clc; close all; clear all;
2
3 %% Import data
4 DeltaAlphaCLmax_vs_LambdaLE_dy1p2 =
      importdata('DeltaAlphaCLmax_vs_LambdaLE_dy1p2.mat');
5 DeltaAlphaCLmax_vs_LambdaLE_dy2p0 =
      importdata('DeltaAlphaCLmax_vs_LambdaLE_dy2p0.mat');
6 DeltaAlphaCLmax_vs_LambdaLE_dy3p0 =
      importdata('DeltaAlphaCLmax_vs_LambdaLE_dy3p0.mat');
```

```
DeltaAlphaCLmax_vs_LambdaLE_dy4p0 =
    importdata('DeltaAlphaCLmax_vs_LambdaLE_dy4p0.mat');

nPoints = 30;
lambdaLEVector_deg = transpose(linspace(0, 40, nPoints));

%% dy/c = 1.2
smoothingParameter = 0.999999;
DAlphaVsLambdaLESplineStatic_Dy1p2 = csaps( ...
    DeltaAlphaCLmax_vs_LambdaLE_dy1p2(:,1), ...
    DeltaAlphaCLmax_vs_LambdaLE_dy1p2(:,2), ...
    smoothingParameter);
DAlphaVsLambdaLEStatic_Dy1p2 = ppval( ...
    DAlphaVsLambdaLESplineStatic_Dy1p2, ...
    lambdaLEVector_deg);
%% dy/c = 2.0
smoothingParameter = 0.999999;
DAlphaVsLambdaLESplineStatic_Dy2p0 = csaps( ...
    DeltaAlphaCLmax_vs_LambdaLE_dy2p0(:,1), ...
    DeltaAlphaCLmax_vs_LambdaLE_dy2p0(:,2), ...
    smoothingParameter);
DAlphaVsLambdaLEStatic_Dy2p0 = ppval( ...
    DAlphaVsLambdaLESplineStatic_Dy2p0, ...
    lambdaLEVector_deg);
%% dy/c = 3.0
smoothingParameter =0.999999;
DAlphaVsLambdaLESplineStatic_Dy3p0 = csaps( ...
    DeltaAlphaCLmax_vs_LambdaLE_dy3p0(:,1), ...
    DeltaAlphaCLmax_vs_LambdaLE_dy3p0(:,2), ...
    smoothingParameter);
DAlphaVsLambdaLEStatic_Dy3p0 = ppval( ...
    DAlphaVsLambdaLESplineStatic_Dy3p0, ...
    lambdaLEVector_deg);
%% dy/c = 4.0
smoothingParameter = 0.999999;
DAlphaVsLambdaLESplineStatic_Dy4p0 = csaps( ...
    DeltaAlphaCLmax_vs_LambdaLE_dy4p0(:,1), ...
    DeltaAlphaCLmax_vs_LambdaLE_dy4p0(:,2), ...
    smoothingParameter);
DAlphaVsLambdaLEStatic_Dy4p0 = ppval( ...
    DAlphaVsLambdaLESplineStatic_Dy4p0, ...
    lambdaLEVector_deg);

%% Plots
figure(1)
plot (lambdaLEVector_deg, DAlphaVsLambdaLEStatic_Dy1p2, '-*b' ... , ...);
hold on;
plot (lambdaLEVector_deg, DAlphaVsLambdaLEStatic_Dy2p0, '-b' ... , ...);
plot (lambdaLEVector_deg, DAlphaVsLambdaLEStatic_Dy3p0, '*b' ... , ...);
plot (lambdaLEVector_deg, DAlphaVsLambdaLEStatic_Dy4p0, 'b' ... , ...);
xlabel('\Lambda_{le} (deg)'); ylabel('\Delta\alpha_{C_{L,max}}');
title('Angle of attack increment for wing maximum lift in subsonic flight');
legend('\Delta y/c = 1.2', '\Delta y/c = 2.0', '\Delta y/c = 3.0','\Delta y/c =
```

```
    4.0');
axis([0 50 0 9]);
grid on;

%% preparing output to HDF
% dy/c
dyVector = [1.2;2.0;3.0;4.0];
%columns --> curves
myData = [ ...
        DAlphaVsLambdaLEStatic_Dy1p2, ...
        DAlphaVsLambdaLEStatic_Dy2p0, ...
        DAlphaVsLambdaLEStatic_Dy3p0, ...
        DAlphaVsLambdaLEStatic_Dy4p0];

hdfFileName = 'DAlphaVsLambdaLEVsDy.h5';
if ( exist(hdfFileName, 'file') )
    fprintf('file %s exists, deleting and creating a new one\n', hdfFileName);
    delete(hdfFileName)
else
    fprintf('Creating new file %s\n', hdfFileName);
end
% Dataset: data
h5create(hdfFileName, '/DAlphaVsLambdaLEVsDy/data', size(myData'));
h5write(hdfFileName, '/DAlphaVsLambdaLEVsDy/data', myData');
% Dataset: var_0
h5create(hdfFileName, '/DAlphaVsLambdaLEVsDy/var_0', size(dyVector'));
h5write(hdfFileName, '/DAlphaVsLambdaLEVsDy/var_0', dyVector');
% Dataset: var_1
h5create(hdfFileName, '/DAlphaVsLambdaLEVsDy/var_1', size(lambdaLEVector_deg'));
h5write(hdfFileName, '/DAlphaVsLambdaLEVsDy/var_1', lambdaLEVector_deg');
```

Listing A.1 MATLAB script for creating the HDF Database

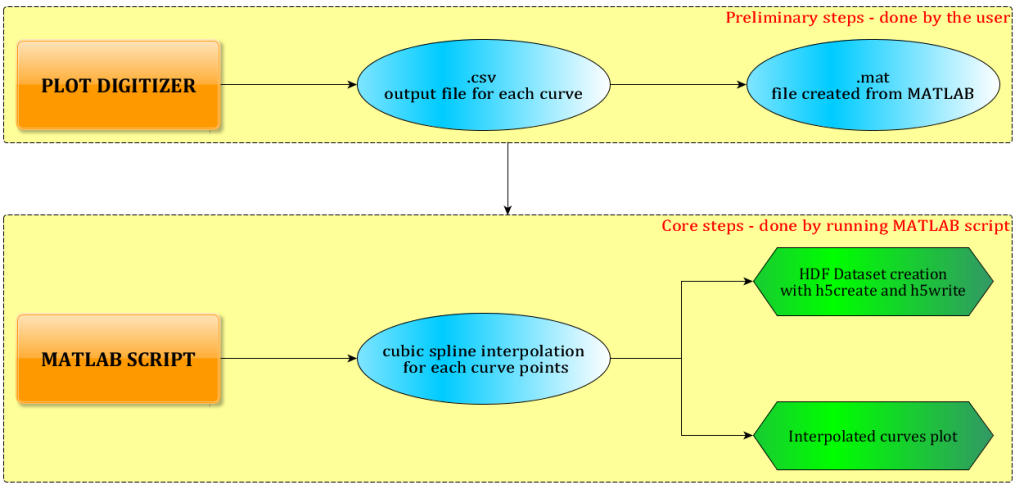

Figure A.1 Flowchart of an HDF Database creation

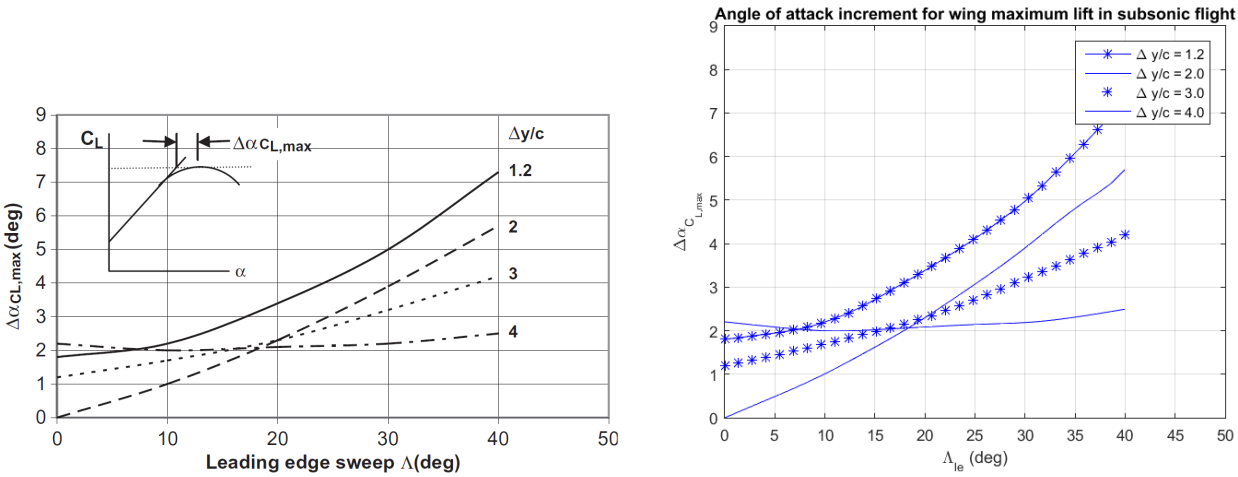

Figure A.2 Comparison between the initial graph and the digitized one

A.2 Reading data from an HDF database in JPAD

After creating the database, this has to be read in order to obtain the required data; inside JPAD this operation can be done defining a specific class, which extends the abstract class `DatabaseReader` that is designed for HDF dataset reading. This son class has a specific structure which main key points can be summarized in the following ones:

1. Creation of variables, in number equal to the function to be interpolated, using the type `MyInterpolatingFunction`
2. Creation of variables for all values that are wanted to be read from the interpolating functions
3. Creation of a constructor that accepts the folder path string and the file name string of the database. This constructor has to launch the interpolating method for all functions contained into the database by using `MyInterpolatingFunction` methods.
4. Creation of a getter method for each of the variables allocated at point 2 in order to obtain values from interpolated functions by giving in input the required parameters

In particular the class `MyInterpolatingFunction` implements methods for a spline, bicubic and tricubic data interpolation as well as three methods for extracting a specific value from each of the previou interpolated curve.

The following listing describes, with an example upon the aerodynamic database, how the reader class should be built up following the previous steps.

```
1  public class AerodynamicDatabaseReader extends DatabaseReader {
2  // STEP 1:
3    private MyInterpolatingFunction
4            c_m0_b_k2_minus_k1_vs_FFR,
5            ar_v_eff_c2_vs_Z_h_over_b_v_x_ac_h_v_over_c_bar_v;
6  // STEP 2:
7    double cM0_b_k2_minus_k1, ar_v_eff_c2;
```

```
8  // STEP 3:
9    public AerodynamicDatabaseReader(String databaseFolderPath, String
       databaseFileName) {
10     super(databaseFolderPath, databaseFileName);
11
12     c_m0_b_k2_minus_k1_vs_FFR =
13             database.interpolate1DFromDatasetFunction(
14                     "(C_m0_b)_k2_minus_k1_vs_FFR"
15             );
16     ar_v_eff_c2_vs_Z_h_over_b_v_x_ac_h_v_over_c_bar_v =
17             database.interpolate2DFromDatasetFunction(
18                     "(AR_v_eff)_c2_vs_Z_h_over_b_v_(x_ac_h-_v_over_c_bar_v)"
19                     );
20   }
21 // STEP 4:
22   public double get_C_m0_b_k2_minus_k1_vs_FFR(double length, double diameter) {
23     return c_m0_b_k2_minus_k1_vs_FFR.value(length/diameter);
24   }
25 // STEP 4:
26   public double get_AR_v_eff_c2_vs_Z_h_over_b_v_x_ac_h_v_over_c_bar_v(double zH,
       double bV, double xACHV, double cV) {
27     return ar_v_eff_c2_vs_Z_h_over_b_v_x_ac_h_v_over_c_bar_v.value(zH/bV, xACHV/cV);
28   }
29 }
```

Listing A.2 DatabaseReader son class creation

Once the class is created, is possible to create an object of it in any test class in order to have access to all its methods; in particular the user needs to invoke the getter related to the quantity he wants to read from the interpolating function.

Bibliography

[1] *A Java desktop application for Aircraft Preliminary Design*. Naples, 2015.

[2] H.I. Abbott and E.A. Von Doenhoff. *Theory Of Wing Sections*. Dover Publications, Inc., 1949.

[3] L. Attanasio. «Development of a Java Application for Parametric Aircraft Design». Master thesis. University of Naples "Federico II", 2014.

[4] Various Authors. *ATR 72*. URL: https://en.wikipedia.org/wiki/ATR_72.

[5] Various Authors. *Boeing 747*. URL: https://en.wikipedia.org/wiki/Boeing_747.

[6] Various Authors. *eXtensible Markup Language (XML)*. URL: https://en.wikipedia.org/wiki/XML.

[7] S. Ciornei. *Mach number, relative thickness, sweep and lift coefficient of the wing - An empirical investigation of parameters and equations*. Paper. 31.05.2005.

[8] Oracle Corporation. *Enumeration*. URL: https://docs.oracle.com/javase/8/docs/api/java/util/Enumeration.html.

[9] Oracle Corporation. *List*. URL: https://docs.oracle.com/javase/8/docs/api/java/util/List.html.

[10] Jean-Marie Dautelle. *Jscience*. URL: http://jscience.org/api/org/jscience/physics/amount/Amount.html.

[11] The Apache Software Foundation. *Interface FirstOrderDifferentialEquations*. URL: https://commons.apache.org/proper/commons-math/apidocs/org/apache/commons/math3/ode/FirstOrderDifferentialEquations.html.

[12] The Apache Software Foundation. *Interface FirstOrderIntegrator*. URL: https://commons.apache.org/proper/commons-math/apidocs/org/apache/commons/math3/ode/FirstOrderIntegrator.html.

[13] The Apache Software Foundation. *Ordinary Differential Equations Integration*. URL: https://commons.apache.org/proper/commons-math/userguide/ode.html.

[14] D. Gambardella. «Development of a Java-Based Framework for Aircraft Preliminary Design». Master thesis. University of Naples "Federico II", 2014.

[15] D. et al Gilbert. *JFreeChart*. URL: http://www.jfree.org/jfreechart/.

[16] The HDF Group. *HDF*. URL: https://www.hdfgroup.org/products/java/hdf-java-html/javadocs/ncsa/hdf/hdf5lib/H5.html.

[17] W.F. Hilton. *High-speed Aerodynamics*. Longmans, Green, 1951.

[18] D. Howe. *Aircraft conceptual design synthesis*. Aerospace Series. Professional Engineering Publishing, 2000.

[19] P. Jackson and FRAeS. *Jane's All the World's Aircraft*. Jane's All the World's Aircraft. Jane's Information Group, 2004-2005.

[20] K. Kawaguchi. *Args4j*. URL: http://args4j.kohsuke.org/.

[21] A. Lausetti. *Decollo e atterramento di aeroplani, idrovolanti trasportati*. Levrotto & Bella Editrice S.a.s., 1992.

[22] B. W. McCormick. *Aerodynamics, Aeronautics, and Flight Mechanics*. John Wiley & Sons, 1979.

[23] L.M. Nicolai and G. Carichner. *Fundamentals of Aircraft and Airship Design*. AIAA education series v. 1. American Institute of Aeronautics and Astronautics, 2010.

[24] F. Nicolosi and G. Paduano. *Behind ADAS*. 2011.

[25] E. Obert. *Aerodynamic Design of Transport Aircraft*. IOS Press, 2009.

[26] P.M. Sforza. *Commercial Airplane Design Principles*. Elsevier Science, 2014.

[27] Aerospace Design Lab at Stanford University. *SUAVE - An Aerospace Vehicle Environment for Designing Future Aircraft*. URL: http://suave.stanford.edu/.

[28] E. Torenbeek. *Advanced Aircraft Design: Conceptual Design, Technology and Optimization of Subsonic Civil Airplanes*. Aerospace Series. Wiley, 2013.

[29] E. Torenbeek. *Optimum Cruise Performance of Subsonic Transport Aircraft*. Delft University Press, 1998.

[30] E. Torenbeek. *Synthesis of Subsonic Aircraft Design*. 1976.

[31] E. Torenbeek. *Synthesis of Subsonic Airplane Design: An Introduction to the Preliminary Design of Subsonic General Aviation and Transport Aircraft, with Emphasis on Layout, Aerodynamic Design, Propulsion and Performance*. Springer, 1982.

[32] Dimitri Van Heesch. *Doxygen*. 1997-2014. URL: www.doxygen.org.

[33] A.D. Young. *The Aerodynamic Characteristics of Flaps*. Technical Report. 1947.

Glossary

Aicraft Construction Reference Frame The reference frame which has its origin in the fuselage forwardmost point, the x-axis pointing from the nose to the tail, the y-axis from fuselage plane of symmetry to the right wing (from the pilot's point of view) and the z-axis from pilot's feet to pilot's head.

Autonomy Factor Is the combination of three main efficiency: the propulsive efficiciency represented by SFC, the propeller efficiency η_p or the jet efficiency represented by V and, finally, the aerodynamic efficinecy $\frac{L}{D}$.

DATCOM Digital Datcom is a computer program which calculates static stability, high lift and control, and dynamic derivative characteristics using the methods contained in the USAF Stability and Control Datcom (Data Compendium). Configuration geometry, attitude, and Mach range capabilities are consistent with those accommodated by the Datcom. The program contains a trim option that computes control deflections and aerodynamic increments for vehicle trim at subsonic Mach numbers.

Direct Operative Cost The totality of aircraft costs directly connected to the aircraft flight. It can be seen as the amount of money necessary to carry 1 ton of payload upon 1 km.

Enumeration The java.util.Enumeration interface represents a special data type that enables for a variable to be a set of predefined constants. The variable must be equal to one of the values that have been predefined for it. Because they are constants, the names of an enum type's fields are in uppercase letters..

Federal Aviation Regulations The Federal Aviation Regulations, or FARs, are rules prescribed by the Federal Aviation Administration (FAA) governing all aviation activities in the United States.

Hierarchical Data Format A set of file formats (HDF4, HDF5) designed to store and organize large amounts of data.

Integrated Development Environment An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger. Most modern IDEs have an intelligent code completion.

Some IDEs contain a compiler, interpreter, or both, such as NetBeans and Eclipse; others do not, such as SharpDevelop and Lazarus. The boundary between an integrated development environment and other parts of the broader software development environment is not well-defined. Sometimes a version control system, or various tools to simplify the construction of a Graphical User Interface (GUI), are integrated. Many modern IDEs also have a class browser, an object browser, and a class hierarchy diagram, for use in object-oriented software development.

Interface In the Java programming language, an interface is a reference type, similar to a class, that can contain only constants, method signatures, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods. Interfaces cannot be instantiated—they can only be implemented by classes or extended by other interfaces..

Java Program toolchain for Aircraft Design Collection of libraries and classes with the aim of providing complete aircraft preliminary design analyses through the use of several semi-empirical formulas tested against experimental data.

Knowledge-Based Engineering KBE is essentially engineering on the basis of knowledge models. A knowledge model uses knowledge representation to represent the artifacts of the design process (as well as the process itself) rather than, or in addition to, conventional programming and database techniques. KBE can have a wide scope that covers the full range of activities related to Product Lifecycle Management and Multidisciplinary Design Optimization. KBE's scope includes design, analysis, manufacturing, and support.

List The java.util.List interface is a subtype of the java.util.Collection interface. It represents an ordered list of objects, meaning you can access the elements of a List in a specific order, and by an index too. You can also add the same element more than once to a List.

Map The java.util.Map interface represents a mapping between a key and a value. The Map interface is not a subtype of the Collection interface. Therefore it behaves a bit different from the rest of the collection types.

Multi-disciplinary Design Optimization Multi-disciplinary design optimization (MDO) is a field of engineering that uses optimization methods to solve design problems incorporating a number of disciplines.MDO allows designers to incorporate all relevant disciplines simultaneously. The optimum of the simultaneous problem is superior to the design found by optimizing each discipline sequentially, since it can exploit the interactions between the disciplines. However, including all disciplines simultaneously significantly increases the complexity of the problem.

parsing Parsing or syntactic analysis is the process of analysing a string of symbols, either in natural language or in computer languages, conforming to the rules of a formal grammar.

Portable Network Graphics A raster graphics file format that supports lossless data compression. PNG was created as an improved, non-patented replacement for Graphics Interchange Format (GIF), and is the most used lossless image compression format on the Internet.

static method The term static means that the method is available at the Class level, and so does not require that an object is instantiated before it's called.

TikZ Is a set of higher-level macros that use PGF.

True AirSpeed Is the speed of the aircraft relative to the airmass in which it is flying; TAS is the true measure of aircraft performance in cruise, thus it is the speed listed in aircraft specifications, manuals, performance comparisons, pilot reports, and every situation when cruise or endurance performance needs to be measured. It is the speed normally listed on the flight plan, also used in flight planning, before considering the effects of wind.

user developer The term refers to the developer which will use a method without being interested in how the method performs the required action. This is the case of a utility method: the developer is the one who writes the method, while the user developer is who uses that method to accomplish some action which requires the functionality provided by the utility method. It has to be noticed that the user developer and the developer can be the same person.

Acronyms

A.F. Autonomy Factor.

ACRF Aicraft Construction Reference Frame.

AIAA American Institute of Aeronautics and Astronautics.

AOE All Operative Engines.

CAE Computer-Aided Engineering.

D.O.C. Direct Operative Cost.

FAR Federal Aviation Regulations.

GUI Graphical User Interface.

HDF Hierarchical Data Format.

IDE Integrated Development Environment.

IVP Initial Value Problem.

JPAD Java Program toolchain for Aircraft Design.

KBE Knowledge-Based Engineering.

LER Leading Edge Radius.

MDO Multi-disciplinary Design Optimization.

MZFW Maximum Zero Fuel Weight.

ODE Ordinary Differential Equations.

OEI One Inoperative Engine.

PNG Portable Network Graphics.

SFC Specific Fuel Consumption.

SFCJ Jet Specific Fuel Consumption.

TAS True AirSpeed.

UAV Unmanned Aerial Vehicles.