Università degli Studi di Napoli Federico II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Aerospaziale

Tesi di Laurea Magistrale
in
Ingegneria Aerospaziale

Development of a Java-Based Framework for Aircraft Preliminary Design

Wing Aerodynamic Analysis Module,
Aircraft Longitudinal Static Stability Module

Relatori:
Prof. Ing. Fabrizio Nicolosi
Prof. Ing. Agostino De Marco

Candidato:
Manuela Ruocco
Matricola M53/408

Anno Accademico 2014-2015

Dedica

Abstract

The purpose of this Thesis work is to introduce the features and the potenciality of ADOpT (*Aircraft Design and Optimization Tool*), a java-based framework concieved as a fast and efficient tool useful as support in the preliminary design phases of an aircraft, and during its optimizaton process.
The ADOpT development originates in the Departement of Industrial Engineering of University of Naples "Federico II", where is still in development. At present this tool is capable to perform a multi-disciplinary analysis of an aircraft whose data can be entered by the user, with an XML, or loaded into memory. The ultimate goal of ADOpT is to carry out an optimization process where the analysis are cyclically repeated in order to optimize some parameters while keeping others in fixed limits. Currently the software is able to esimate the aircraft weight breackdown, the center of gravity location, calculate some aerodynamic parameters and estimate the performance. All these types of estimates can be usually performed using several interchangeable analysis methods, comparable and interchangeable. It is also provided a static longitudinal stability analysis, take-off and landing performances and the generation of Payload Range chart.
ADOpT can be used from the command line or with a dedicated graphical user interface (GUI). The GUI allows the user to have an immediate feedback about the aircraft features when changing the input parameters, to manage multiple aircraft simultaneously and compare them side by side, and to view a 3D CAD model of the aircraft.
The ADOpT potentiality, in the world of research or in industry, are remarkable and the software strengths are a considerable computing speed and flexibility, with an user-friendly GUI.

The structure of this thesis work has as ultimate goal to provide a comprehensive overview about ADOpT and, at the same time, it is intended to be a developer's manual. The first chapters provide a complete software overview paying particular attention at actual features and future goals. Following chapters introduce some case of study and the results achieved. At the beginning of each chapter is exposed the theoretical background, afterwards there is a description of the Java architecture and, at the end is reported the Test class used for the analysis and its results.

Sommario

Lo scopo che il presente lavoro di Tesi auspica raggiungere è quello di presentare le capacità possedute e le potenzialità future di ADOpT, un software scritto in Java che si configura come uno strumento veloce ed efficiente per il supporto nella fase di progetto preliminare di un velivolo e per la sua ottimizzazione.
Lo sviluppo di ADOpT (*Aircraft Design and Optimization Tool*) nasce all' interno del Dipartimento di Ingegneria Industriale dell' Università degli Studi di Napoli Federico II, ove è tutt'ora in fase di progresso. Il software è attualmente in grado di svolgere una parziale analisi multi-disciplinare di un velivolo i cui dati sono immessi dall' utente tramite XML o caricati in memoria. La linea guida dello sviluppo del software porta verso l'implementazione di un processo di ottimizzazione nel quale le analisi sono ciclicamente ripetute al fine di ottimizzare alcuni parametri mantenendone altri all' interno di limiti imposti.
Attualmente ADOpT è in grado di effettuare una completa stima dei pesi, valutare la posizione del baricentro, calcolare un notevole numero di parametri aerodinamici e caratteristiche di performance. La stima di ciascun parametro può essere effettuata tramite diversi metodi implementati, tra di loro confrontabili ed intercambiabili. Inoltre è prevista un' analisi di stabilità statica, prestazioni di decollo ed atterraggio e la generazione del diagramma *Payload Range*.
ADOpT può essere utilizzato sia in modalità *batch*, ossia da riga di comando, che tramite interfaccia grafica. Tale duplice scelta consente di ottenere le migliori prestazioni sia nei processi di analisi, ove la GUI consente di avere un immediato riscontro grafico, sia nei processi di ottimizzazione.
Dunque le potenzialità di ADOpT nel mondo della ricerca od anche in quello industriale sono senza dubbio notevoli e il software gioca i suoi punti di forza in un' elevata flessibilità e una notevole rapidità di calcolo, senza dimenticare e una *user-friendly* interfaccia grafica.

L' organizzazione di questo lavoro di Tesi è stata studiata per cercare di fornire una completezza di esposizione, ma allo stesso tempo risultare un utile manuale per lo sviluppatore. I primi capitoli forniscono una visione globale del software con particolare attenzione alle funzionalità presenti e alle scelte effettuate. I capitoli che seguono presentano una panoramica su alcune funzionalità di ADOpT con i relativi casi di studio e i risultati ottenuti dalle analisi. Per ogni capitolo viene preliminarmente fornita una visione globale circa la teoria alla base dei metodi implementati, seguita dalla descrizione delle classi e dei metodi relativi in Java ed è, infine, riportato il codice della *Test Class* implementata per lo svolgimento dell' analisi con i relativi risultati.

Contents

Part I

Aircraft Design Overview

Chapter 1

Aircraft Design

Chapter 2

ADOpT applicaton overview

Part II

Development of Application

Chapter 3

Introdution to Java

Chapter 4

Work Object

Citazione
citazione
– Autore

4.1 Introduction

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 methos is that using an .XML file, user can to define its own aircraft having 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.

4.2 Input data from .XML file

XML is a file extension for an *Extensible Markup Language (XML)* file format used to create common information formats and share both the format and the data on the World Wide Web, intranets, and elsewhere using standard ASCII text. It is defined “Markup Language” due to the use of tags that describes the content. XML is considered extensible because the markup symbols are unlimited and self-defining. So it is possible to use personal tag for each data. In this way to read an .XML file results relatively simple.[9]
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 4.1: 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 4.2: Use of attributes in XML language.

An .XML file has a tree structure where there are extenal knots that branch into internal knots.

In order to read an XML file it is necessary, first of all, to give the file path. 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`, 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 `JPADDataWriter` class. First of all it is necessary to define and build the xml tree structure. After each variable is associated to a name that is the markup symbol of the XML file.

4.3 Default Aircraft

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** is a twin-engine turboprop made by the French-Italian aircraft manufacturer ATR entered service in 1989. It was developed as a variant of the ATR 42 with a 4.5 m stretched fuselage. The ATR 72 was developed from the ATR 42 in order to increase the seating capacity (48 to 66 in standard configuration) by stretching the fuselage, increasing the wingspan, adding more powerful engines, and increasing fuel capacity by approximately 10 percent.[5] It has been typically employed as a regional airliner, although other roles have been performed by the type such as corporate transport, cargo aircraft and maritime patrol aircraft. [4]

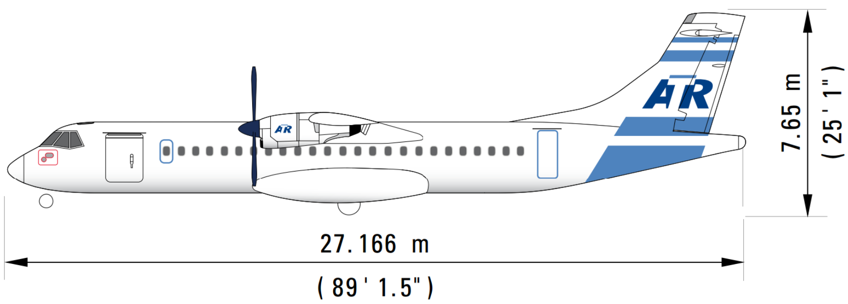


Figure 4.3: ATR 72. Side view.

The **Boeing 747-100B** is a four-engined long-range widebody commercial jet airliner and cargo aircraft produced by the American manufacturer Boeing Commercial Airplanes. It has a capacity of maximum 480 passengers in a partial double deck configuration. The Boeing 747 It is also known as Jumbo Jet. The basic B747-100 entered service with Pan American On January 15, 1970.
One of the reason to create the 747 was reductions in airfares with a consequent increase of

passenger traffic[6]. The original version of the 747 had two and a half times greater capacity than the Boeing 707, one of the common large commercial aircraft of the 1960s and it was the largest passenger carrier from 1970 until the introduction of Airbus A380.[7] The Boeing 747 had two aisle and four wing-mounted engines. The upper deck is its distinctive "hump" along the forward part of the aircraft. It provides space for a lounge or extra seating. The raised cockpit allows front loading of cargo on freight variants.
The 747-100B model was developed from the 747-100SR. This configuration had a typical 452 passengers and 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.

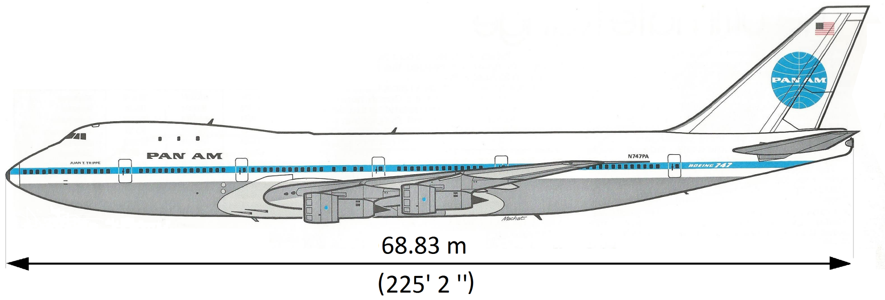


Figure 4.4: Boeing 747-100B. Side view.

4.3.1 How is made a default Aircraft

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 step. First of all it is necessary to initialize the working directory tree using the method `initWorkingDirectoryTree` of `MyConfiguration` class located in `JPADConfigs` package that initializes the working directory tree and fill the map of folders. This step is required in order to create the following default folders that are necessary for the right behavior of the code:

- Database directory
- Input directory
- Output directory

Using `MyConfiguration` class it's possible to point at a specific folder, like the input or output directory, with the static method `getDir`. This is a crucial step that must be execute at the beginning of every test. To set the working directory with the useful folders, it's necessary to call the function `initWorkingDirectoryTree()` at the beginning of each test. The function creates all necessary folders. Morover the function has been overloaded and it can be even called with a variable number of arguments (`initWorkingDirectoryTree( String...str)`). These strings are the directory strings in `MyConfiguration` class. After it is possible to create an `Aircraft` object choosing between “ATR-72” or “B747-100B” using the method `createDefaultAircraft` from `Aircraft` class. This method defines a new Aircraft object and invokes another Aircraft's methods that creates the component using deafult data. In the method `createDeafaultAircraft` there is a calling to the builder of `Aircraft` class that initializes the objects of the classes that perform calculations. At this step all the components of the aircraft are created. It is possible also to define new airfoil for the aircraft or change some data from the existing.
Afterwards it is necessary to set the operating conditions such as the number of Mach of analysis or altitude. Each default aircraft has a set of default condition but the user could to change them.

In order to manage all the aircraft related analysis it is necessary to define an object of the class `ACAnalysisManager`. Similarly to the aircraft, exist an analysis manager also for the wing that is an object of the `LSAerodynamicAnalysis` class.
The next step is to define and assign the needed databases. This will be explained in detail in the next section. Finally it is possible to do analysis.

Listing 4.1 Generation of default aircraft

```
public static void main(String[] args) {

        // ---------------------------------------------------------------
        // Define directory
        // ---------------------------------------------------------------
        MyConfiguration.initWorkingDirectoryTree();


        // ---------------------------------------------------------------
        // Generate default Aircraft
        // ---------------------------------------------------------------
        Aircraft aircraft = Aircraft.createDefaultAircraft("B747-100B");
        LiftingSurface theWing = aircraft.get_wing();

        // Default operating conditions
        OperatingConditions theConditions = new OperatingConditions();


        // ---------------------------------------------------------------
        // Define an ACAnalysisManager Object
        // ---------------------------------------------------------------
        ACAnalysisManager theAnalysis = new ACAnalysisManager(theConditions);
        theAnalysis.updateGeometry(aircraft);


        // ---------------------------------------------------------------
        // Define an LSAerodynamicsManager Object
        // ---------------------------------------------------------------
        LSAerodynamicsManager theLSAnalysis = new LSAerodynamicsManager (
                        theConditions,
                        theWing,
                        aircraft
                        );


        // ---------------------------------------------------------------
        // Setup database(s)
        // ---------------------------------------------------------------

        theLSAnalysis.setDatabaseReaders(
                        new Pair(DatabaseReaderEnum.AERODYNAMIC,
                          "Aerodynamic_Database_Ultimate.h5"),
                        new Pair(DatabaseReaderEnum.HIGHLIFT,
                          "HighLiftDatabase.h5")
                        );


        // ---------------------------------------------------------------
        // Do analysis
        // ---------------------------------------------------------------
        theAnalysis.doAnalysis(aircraft,
                        AnalysisTypeEnum.AERODYNAMIC);
}
```

4.3.2 How is made a default Wing

Similary to the default aircraft it is possible to define a default wing. This is very useful if the user wants to make an analysis only on a wing. In this case it is necessary to define the origin of the Local Reference Frame (LRF) in Body Reference Frame (BRF) and the coordinates of the Gravity Center (GC).

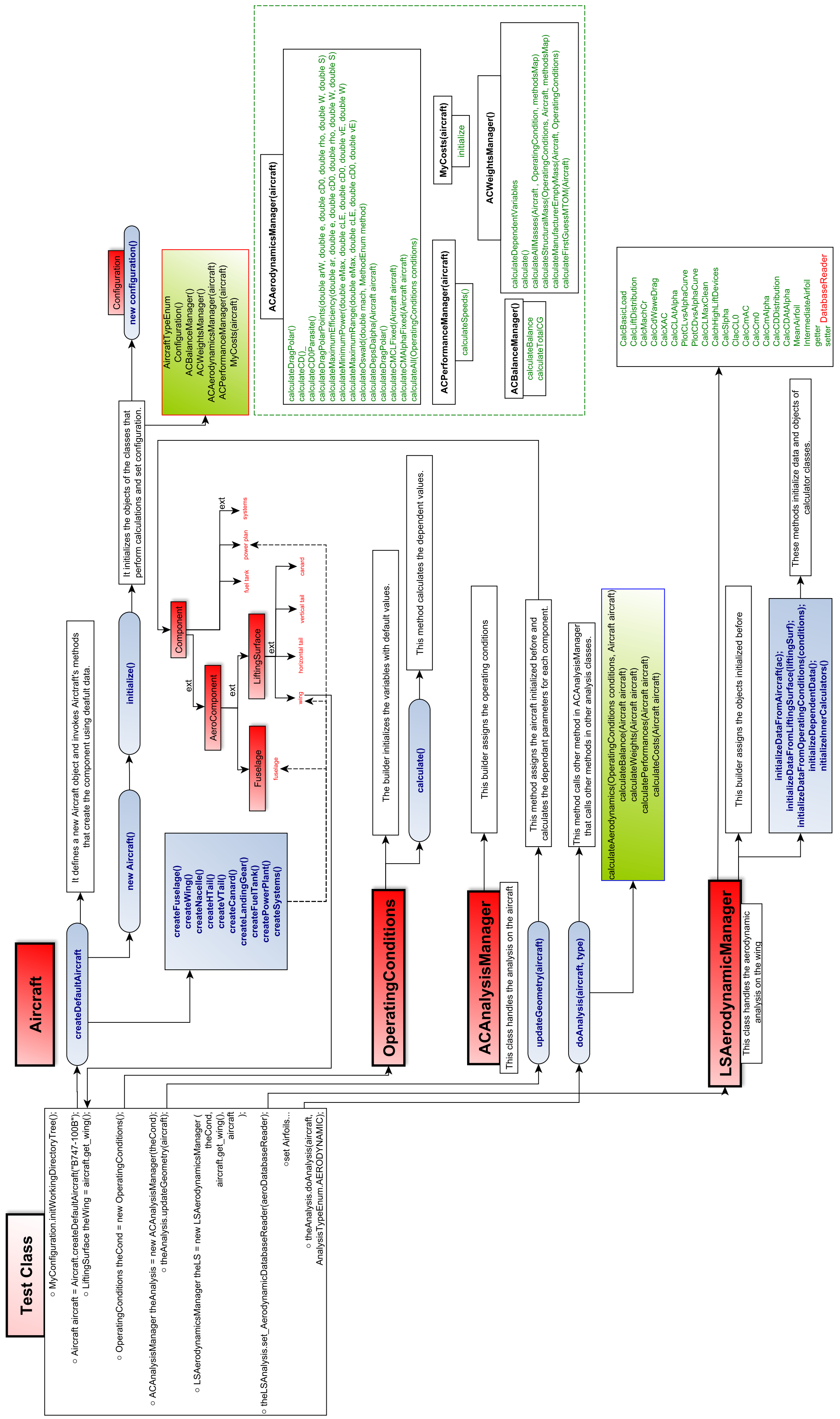


Figure 4.5: Flow chart of the creation of default Aircraft.

Contrary to the case of the aircraft, for an isolated wing there isn't necessary to define a fuselage in order to create a `Lifting Surface` object, but there is an overload of the builder that doesn't need a fuselage as input. In this case the exposed surface is calculated as the surface of the wing.

Listing 4.2 Generation of an isolated Wing

```java
public static void main(String[] args) {

        // Assign all default folders
        MyConfiguration.initWorkingDirectoryTree();

        // -----------------------------------------------------------------------
        // Coordinates of LRF
        // -----------------------------------------------------------------------

        double xAw = 11.0; //meter
        double yAw = 0.0;
        double zAw = 1.6;
        double iw = 0.0;

        // -----------------------------------------------------------------------
        // Generate default Wing
        // -----------------------------------------------------------------------

        LiftingSurface theWing = new LiftingSurface(
                        "Wing", // name
                        "Data␣from␣AC_ATR_72_REV05.pdf",
                        xAw, yAw, zAw, iw,
                        ComponentEnum.WING
                        );

        theWing.calculateGeometry();
        theWing.getGeometry().calculateAll();

        // -----------------------------------------------------------------------
        // Center of Gravity
        // -----------------------------------------------------------------------

        double xCgLocal= 1.5; // meter
        double yCgLocal= 0;
        double zCgLocal= 0;

        CenterOfGravity cg = new CenterOfGravity(
                        Amount.valueOf(xCgLocal, SI.METER), // coordinates in LRF
                        Amount.valueOf(yCgLocal, SI.METER),
                        Amount.valueOf(zCgLocal, SI.METER),
                        Amount.valueOf(xAw, SI.METER), // origin of LRF in BRF
                        Amount.valueOf(yAw, SI.METER),
                        Amount.valueOf(zAw, SI.METER),
                        Amount.valueOf(0.0, SI.METER),// origin of BRF
                        Amount.valueOf(0.0, SI.METER),
                        Amount.valueOf(0.0, SI.METER)
                        );

        cg.calculateCGinBRF();
        theWing.set_cg(cg);
        theWing.set_aspectRatio(6.0);

        // Default operating conditions
        OperatingConditions theOperatingConditions = new OperatingConditions();
        theOperatingConditions.set_alphaCurrent(Amount.valueOf(2.0, NonSI.DEGREE_ANGLE)

        // --------------------------------------------------------------
        // Define an LSAerodynamicsManager Object
        // --------------------------------------------------------------

        LSAerodynamicsManager theLSAnalysis = new LSAerodynamicsManager (
                        theOperatingConditions,
                        theWing
                        );

        // --------------------------------------------------------------
```

```
        // Setup database(s)
        // ---------------------------------------------------------------

        theLSAnalysis.setDatabaseReaders(
                        new Pair(DatabaseReaderEnum.AERODYNAMIC,
                                "Aerodynamic_Database_Ultimate.h5"),
                        new Pair(DatabaseReaderEnum.HIGHLIFT, "HighLiftDatabase.h5")
                        );

        // ---------------------------------------------------------------
        // Assign Airfoil(s) ...
        // ---------------------------------------------------------------

          // Define airfoilRoot...

        // ---------------------------------------------------------------
        // Set Airofoil(s)
        // ---------------------------------------------------------------
        List<MyAirfoil> myAirfoilList = new ArrayList<MyAirfoil>();
        myAirfoilList.add(0, airfoilRoot);
        myAirfoilList.add(1, airfoilKink);
        myAirfoilList.add(2, airfoilTip);
        theWing.set_theAirfoilsList(myAirfoilList);
        theWing.updateAirfoilsGeometry();
        theLSAnalysis.initializeDependentData();

}
```

4.4 Database in JPAD

In JPAD it is possible to consult external databases in .h5 format. **HDF 5** (Hierarchical Data Format Release 5) is a data file format designed by the *National Center for Supercomputing Applications* (NCSA) to assist users in the storage and manipulation of scientific data across different operating systems and machines.
To obtain the useful data in JPAD interpolating functions are used . These functions can be of one, two or three dimensions and read data from graphics that have been digitize previously. Starting from these digitalizations, databases in .h5 format are built. Reading data from databases is entrusted to methods of classes in the `database` package.
In order to read these databases, and obtain the useful data, it is necessary to define an object of the database reading class and associate it with the object of analysis.
This is a crucial step to read correctly the external data. In fact JPAD allows to work with an aircraft object or only with an isolated lifting surface object. Aircraft is usually composed of a fuselage, lifting surfaces, nacelle and power plant. Furthermore, `Aircraft` and `Wing` are associated with classes of calculation like `LSAerodynamicManager` or `ACAnalysisManager`. So it is necessary that these databases are also visible from these classes.
So because both in aircraft and in wing there is a lifting surface object, databases relative to wing are associated to `LSAerodynamicManager`.

4.4.1 Setup database

Here the database path it's created and associated to object that interpolates the required data from the .h5 file using a `MyInterpolatingFunction` object. After this it's possible to access the double value of the interpolating function using the `standaloneutils` method called `value`.

Now the procedure to assign the database is different if is used an Aircraft object or a Wing object.

4.4.2 Assign database using an Aircraft object

In order to assign correctly the database and associate it to all analysis management is necessary to practise the following order.

1. Define an Aircraft Object.
   This command associates to Aircraft an object that defines the aerodynamic. From the wing it is possible to obtain the Wing, 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.

4.4.3 Assign database using a Wing object

Using a Wing object it isn't neccessary to define a manager for Aircraft aerodynamic analysis. So the step to follow are the same of aircraft starting from the third.

1. Define an Wing Object.

2. Define an `LSAerodynamicManager` object.

3. Associate database to `LSAerodynamicManager`.

The definition of a isolated Wing is explained in the relative section.

Listing 4.3 Assign database using an Aircraft object

```
        // ---------------------------------------------------------------
        // Setup database(s)
        // ---------------------------------------------------------------

        theLSAnalysis.setDatabaseReaders(
                        new Pair(DatabaseReaderEnum.AERODYNAMIC,
                          "Aerodynamic_Database_Ultimate.h5"),
                        new Pair(DatabaseReaderEnum.HIGHLIFT,
                          "HighLiftDatabase.h5")
                        );
```

The databases are assigned to `LSAerodynamic` using a method of this class. This method accept as input a variable number of `Pair` objects.Using `Pair` objects it is possible to assign, for each database, both name and type.

Listing 4.4 `setDatabaseReaders` method

```
public void setDatabaseReaders(Pair... args) {
        String databaseFolderPath = MyConfiguration.getDir(FoldersEnum.DATABASE_DIR);

        for (Pair a : args) {
                DatabaseReaderEnum key = (DatabaseReaderEnum)a.getKey();
                String databaseFileName = (String)a.getValue();
```

```
            switch (key) {
                    case AERODYNAMIC:
                            _aerodynamicDatabaseReader =
                            new AerodynamicDatabaseReader(
                                            databaseFolderPath,
                                            databaseFileName);
                            listDatabaseReaders.add(_aerodynamicDatabaseReader);
                            break;

                    case HIGHLIFT:
                            _highLiftDatabaseReader =
                            new HighLiftDatabaseReader(
                                            databaseFolderPath,
                                            databaseFileName);
                            listDatabaseReaders.add(_highLiftDatabaseReader);
                    break;

            }
```

4.4.4 User's guide

In order to execute some analysis in JPAD it is necessary, first of all, to define an analysis object in the Test class. The method `createDefaultAircraft` creates a new aircraft and the object that composes it. This method also populates the data of aircraft with default value corresponding to ATR-72 or Boieng 747_100B. Moreover the method `createDefaultAircraft` calls another method in `Aircraft` class: `initialize` that initializes the objects of the classes that perform calculations.
The purpose of this structure is to have only a way to assign the databases at an aircraft. Inasmuch as the wing is always present, the chosen strategy is to assign the database to the aerodynamic manager of the wing.
In order to bring to use the database also for the aircraft calculation, it is assigned at the aerodynamic manager of the aircraft in the method called `doAnalysis`.
At the same time `LSAerodynamicManager` sets itself as aerodynamic in the wing object. So it is 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;`

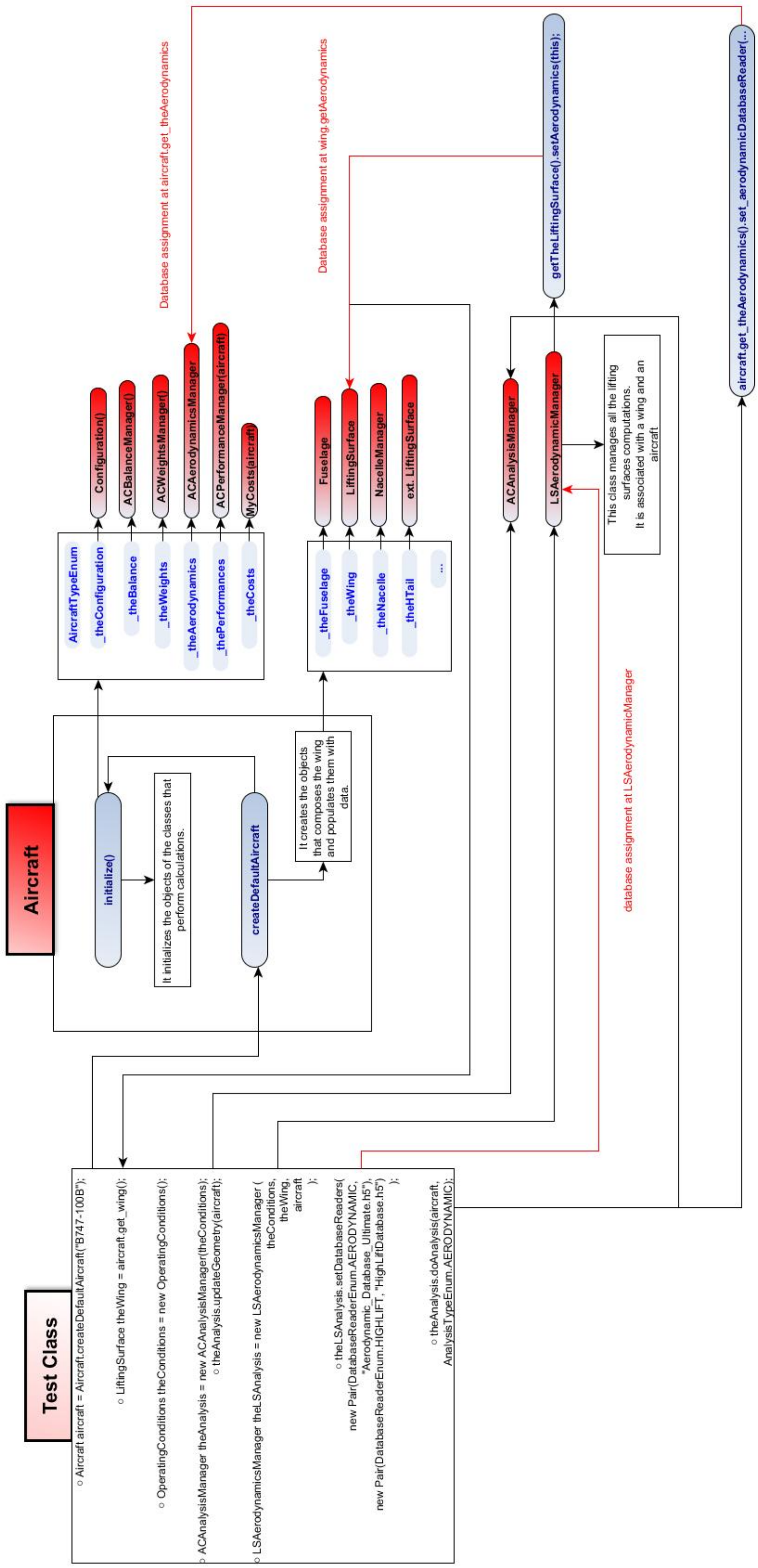


Figure 4.6: Flow chart of database assignment.

Part III

Functionality Overview

Chapter 5

Work Object Data

Citazione citazione
– Autore

All the following analysis will be carried out using a default aircraft. This choice is made in order to avoid the collection and validation data phase for each analysis and maake focus on the results. As mentioned, there are two default aircraft in the code: ATR-72 AND Boieng 747_100B whose data are shown in the table below.
All the analysis that follow, refer to the data presented in this chapter and those derived from them .

Chapter 6

Wing Lift Characteristics

Citazione
citazione
– Autore

Any body in motion in a fluid presents a result of force acting on it, which can be decomposed in two components:

- A **Lift** acting normal to the Velocity direction and is positive upward.
- A **Drag** acting in the opposite direction to the airspeed vector.

The lifting surfaces of an airplane are designed to generate lift exceeding their drag, in order to obtain a positive efficience.

Chapter 7

Wing Drag Characteristics

Citazione
citazione
– Autore

As mentioned in the previous chapter, the drag is the force component acting in the opposite direction to the airspeed vector.
There isn't a single classification of the drag but, dependent on the purpose of the work, the drag may be broken down in different way. Following will be explained the two main classification.

- The drag is subdivided using a causal breakdown. In this way the drag contributes are in accordance with the physical mecchanism such as the viscosity of the flow.
- The drag is subdivided using a component breakdown. Every component of aircraft added an own drag contribute.

According to the casual breakdown it's possible to make a preliminary division considering normal and tangential stress. The tangential forces produce the *friction drag*. While it's possible to divide the drag due of the normal component in viscous, that generates *form drag*, and inviscid. A further division can be made for the last one, in *induced drag* and *wawe drag*.

7.1 Theoretical background

7.1.1 Integral estimation

7.1.2 Sforza

7.2 Java Class Architecture

7.3 Case Study

Chapter 8

Downwash Estimation

Citazione
citazione
– Autore

In order to evaluate the characteristics of longitudinal stability of an aircraft it's necessary to assess the flow direction aft of the wing. The contribution of horizontal tail surface to the airplane equilibrium and stability, in fact, depends seriously on the flow direction. The pourpose of this chapter is to introduce and evaluate the downwash gradient due from the wing's vortex system, considering a dipendence of the downwash angle from the absolute angle of attack.

8.1 Theoretical background

Due to the finite extention of the wing the lift distribution in span is not uniform. For this reason the difference of pressure between upper and lower surfaces generates a movement of air aroud the wingtips. The tendency is for particles of air to move from the region of high pressure around the wing tip to the region of low pressure (for positive lift from the lower wing surface to the upper surface). This made the wing's vortex system that consisting of the bound vortex, located at the wing quarter chord and a vortex sheet which rolling up, at the wing tip, in two trailing vortex.[50] [63]

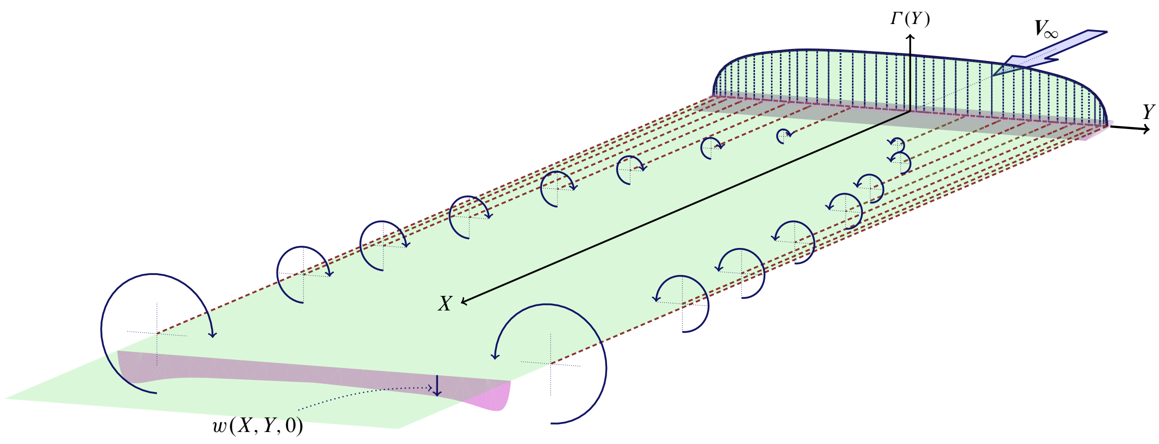


Figure 8.1: The wing vortex sheet.

The main effect of this vortex system is to deflect the airflow behind the wing downward relative to the direction of freestream flow. This angle of deviation is known as *Downwash Angle* ϵ. This phenomenon occurs for every lifting surface, but in subsonic flow a lifting surface also affects the flow forward of itself. In this region the vortex creates an itshape upwash, that is an upward flow deflection.

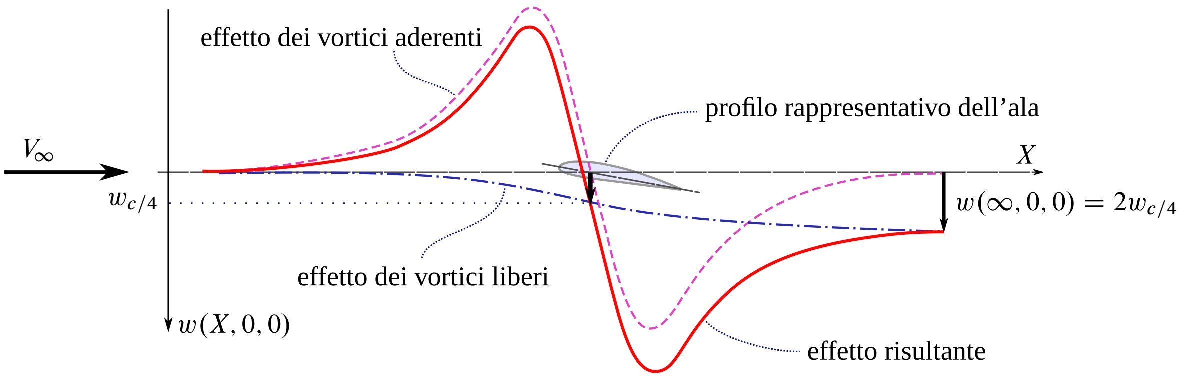


Figure 8.2: Upwash and Downwash in a finite wing.

As consequence of the downwash behind the wing, the local angle of attack on the horizontal tail is reduced by ϵ. In order to evaluate the flow direction behind the wing, an other important parameter is the change in downwash angle with angle of attack, that is the *Downwash Gradient* $\frac{d\epsilon}{d\alpha}$.
This parameter depends principally on the location of the horizontal tail with respect to the wing and the vortex plane. As first approssimation this value could be considered constant in alpha, but more accurately it's possible to evaluate this dependence considering the reference variable for the calculation of the distances. Both if the vertical distance is considered as constant and it is considered variable with alpha, starting from the downwash gradient, the downwash angle is :

$$\epsilon = \frac{d\epsilon}{d\alpha_w}(\alpha_w - \alpha_{0_w}) \tag{8.1}$$

In order to evaluate the downwash gradient it refers to fig. 8.3, where "$r\frac{b}{2}$" is the distance between the aerodynamic center of wing and the aerodynamic center of the horizontal tail. This is a geometric an fixed distance. Conversely, in order to have a greates accuracy it's possible to consider the distance "$m\frac{b}{2}$" variable with the angle of attack. Properly this is the distance between the horizontal tail and the vortex shed plane, but it's possible to approximate it with the distance between the horizontal tail and the wing root chord.[61]

Referring to the equation used in order to evaluate the downwash gradient is the following:

$$\begin{aligned}\frac{d\epsilon}{d\alpha} = \frac{K_{\epsilon\Lambda}}{K_{\epsilon\Lambda=0}}\Bigg(&\frac{r}{r^2+m_{tv}^2}\frac{0.4876}{\sqrt{r^2+0.6319+m_{tv}^2}}+\\ &\left[1+\left(\frac{r^2}{r^2+0.7915+5.0734m_{tv}^2}\right)^{0.3113}\right]\left\{1-\sqrt{\frac{m_{tv}^2}{1+m_{tv}^2}}\right\}\Bigg)\frac{C_{L_{\alpha w}}}{\pi Æ}\end{aligned} \tag{8.2}$$

Considering a variable downwash gradient the changing parameter is $m\frac{b}{2}$.

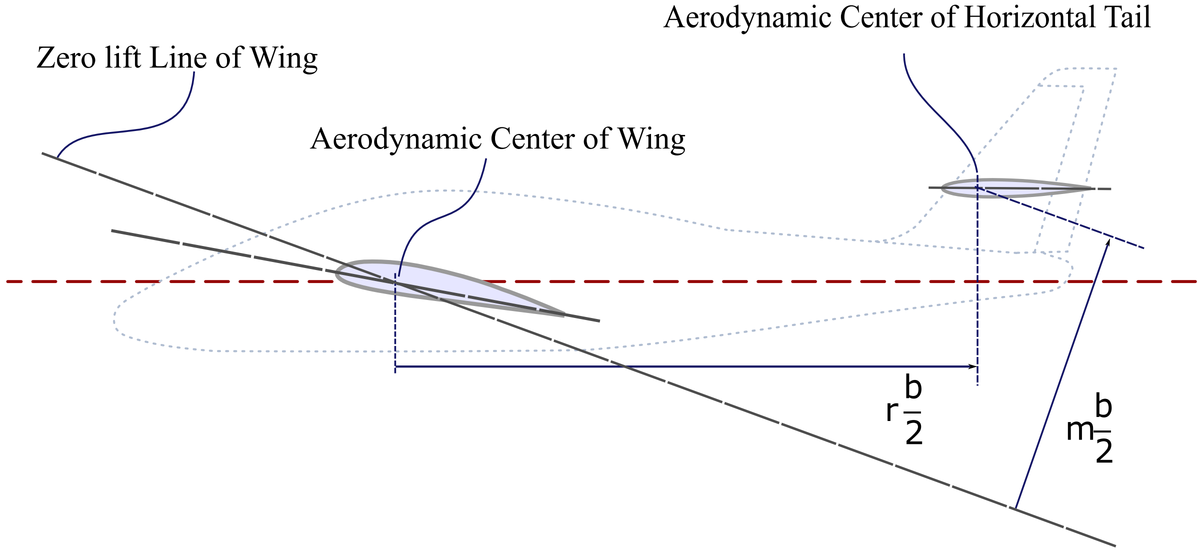


Figure 8.3: Dimensions for determination of Downwash Gradient, considering constant distances.

The two K_ϵ terms in the eq 8.3 accounting for the wing sweep angle effect are defined as follow (where Λ expressed in radians):

$$K_{\epsilon\Lambda} = \frac{0.1124 + 0.1265\Lambda + 0.1766\Lambda^2}{r^2} + \frac{0.1024}{r} + 2 \tag{8.3}$$

$$K_{\epsilon_{\Lambda=0}} = \frac{0.1124}{r^2} + \frac{0.1024}{r} + 2 \tag{8.4}$$

8.2 Java Class Architecture

In order to simplify the calculation of downwash, as mentioned, it's possible to assume the downwash gradient constant with the angle of attack. In this case the reference line to calculate the distance along z axis is the plane from the wing root chord or else the zero-lift line of the wing.
To obtain a more accurate analysis it's possible to consider the variation in alpha of the downwash gradient. So the reference line of wing it's not costant, but is the vortex sheed plane. The location of this plane depend from the value of downwash, but this location is itself necessary to evaluate the downwash. So it's necessary an iterative process in which the position of the vortex reference line at alpha is calculated from the value of downwash gradient at previous step.

In this process the reference angle of attack is the absolute angle α_a, that is the angle between the flow direction and the zero lift line of the wing. This choice is necessary because for $\alpha_a = 0$ it's possible to assume the downwash zero, but the downwash gradient is not null. In this way it's possible to assume the downwash value in the first step of the iteration and to continue for each step with the previous value as first attempt. In view of stability, however, the reference angle of attack is the α_B. The relation between these angles is in the fig 8.4.

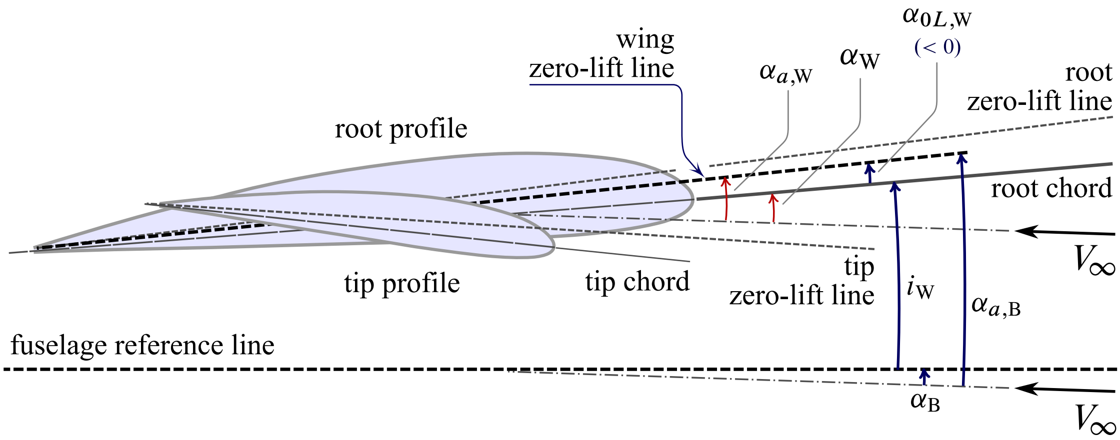


Figure 8.4: Definition of wing angles.

The downwash angle is calculated by a class named `DownwashCalculator`. The builder of this class defines and assings all the geometrical variables, necessary to implement the calculation. The `DownwashCalculator` class has seven methods and some overloads. The methods are explained in the table 8.1. The methods will be explained more in detail below.

`calculateDownwashGradientConstantDelft`	This method calculates the downwash gradient considering the vertical distance geometrical and constant
`calculateDownwashNonLinearDelft`	This method calculates the downwash considering a non constant downwash gradient.
`getDownwashAtAlphaBody`	This method returns the value of downwash angle, interpolating data filled before.
`calculateZDistanceZeroLift`	This method calculates the distance between the aerodynamic centre of horizontal tail and the zero lift line of the wing.
`Plot Methods ...`	Using these methods it's possible to plot the downwash angle, the downwash gradient and the distance in function of α_B

Table 8.1: Methods of `DownwashCalculator` class.

8.2.1 Constant Downwash Gradient

In order to evaluate the downwash angle case of constant downwash gradient it's necessary only to call the method `calculateDownwashGradientConstantDelft` using the distance from aerodynamic center of horizontal tail and the alpha zero lift line as input. It's possible to calculate this distance geometrically using the method `calculateZDistanceZeroLift` of the same class. The choice to calculate separatly the distance and the downwash gradient is made to reuse the method for calculate downwash gradient simply

varying the input distance.
This method has the downwash gradient as output. To obtain the angle of downwash it simply need to moltiplicate the output value and the absolute angle of attack.

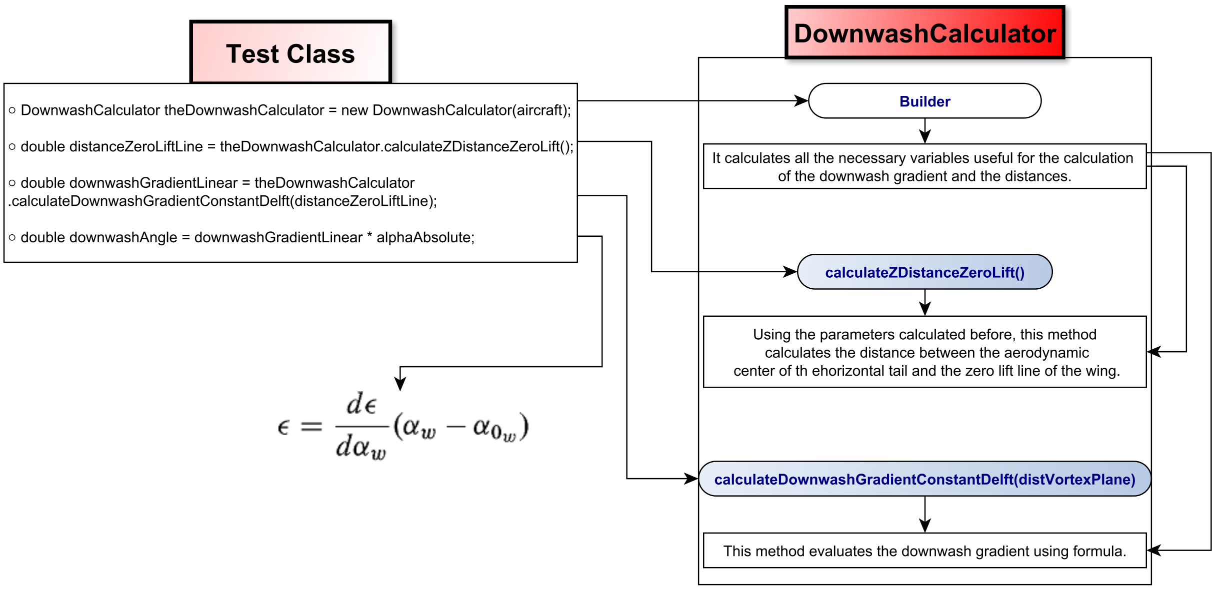


Figure 8.5: Flow chart of the calculation of linear downwash angle.

8.2.2 Variable Downwash Gradient

In order to evaluate the non-constant downwash gradient must to use the method texttt calculateDownwashNonLinearDelft. This method calculates the downwash gradient using Delft formula. The downwash gradient is considered variable in alpha absolute. The distance along x considered in the formula is geometric and fixed. Conversely the other distance is variable and it is considered as the distance between the horizontal tail the vortex shed plane.
This method works through the following steps:

1. First of all this method creates an array of absolute angle of attack starting from $\alpha = 0°$ to $\alpha = 20°$ with a step of $0.25°$.
2. The results array are initialized (α_a , α_b , $\frac{d\epsilon}{d\alpha}$, ϵ , $m\frac{b}{2}$).
3. For the first step the state is the following:
   - $\alpha_a = 0°$
   - $\alpha_b = \alpha_{0L} - i_w$
   - $\frac{d\epsilon}{d\alpha}$ is the constant value calculated with the method
   - $m\frac{b}{2}$ distance is the same of the previous case and it is calculated using the method `calculateZDistanceZeroLift`.
   - $\epsilon = 0$
4. Starting from the second step the process is iterative. Starting from $\alpha_a = 0°$ the absolute angle of attack increase of $\Delta\alpha$. For the step i:
   - $\alpha_a|_i = i\,\Delta\alpha$
   - $\epsilon_{temp} = \frac{d\epsilon}{d\alpha}|_{i-1} * \alpha_a|_i$

- $m\frac{b}{2}_{temp}$ is calculated considering the temporary value of downwash angle.
- $\frac{d\epsilon}{d\alpha}_{temp}$ is calculated using the formula and the temporary value of distance.
- $\epsilon_i = \frac{d\epsilon}{d\alpha}_{temp} * \alpha_a|_i$
- $m\frac{b}{2}|_i$ is calculated considering the new value of downwash angle.
- $\frac{d\epsilon}{d\alpha}|_i$ is updated.
- $\alpha_b = \alpha_{0L} - i_w + \alpha_a$

In order to relieve the calculations, the evaluation of downwash angle and downwash gradient should be done only one time. To obtain the value of epsilon at alpha body it's possible to call the method texttgetDownwashAtAlphaBody that interpolates the value of downwash angle and angle of attack which field must be filled before.

Step by step the value of $m\frac{b}{2}$ is calculated with the following geometrical construction:

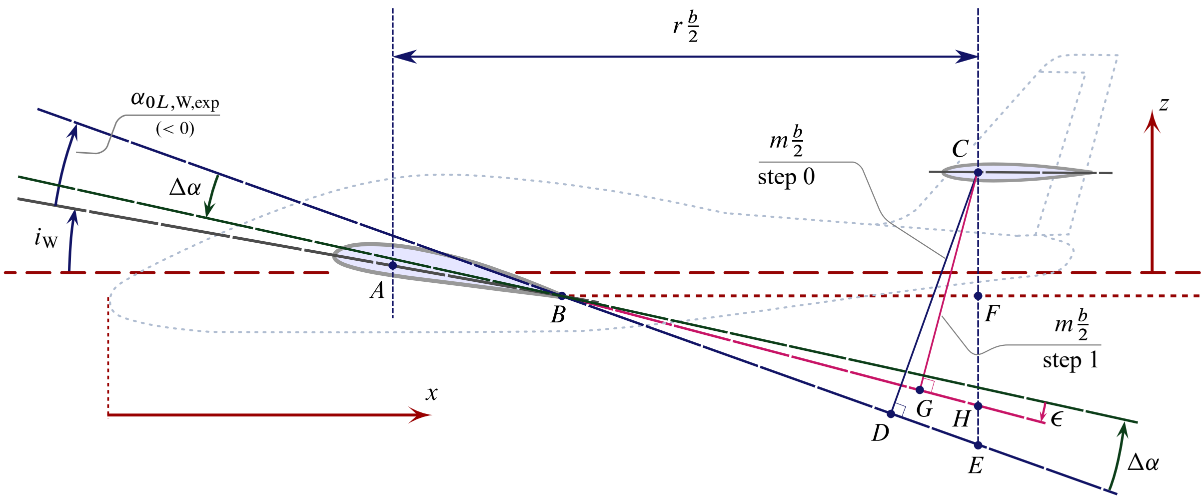


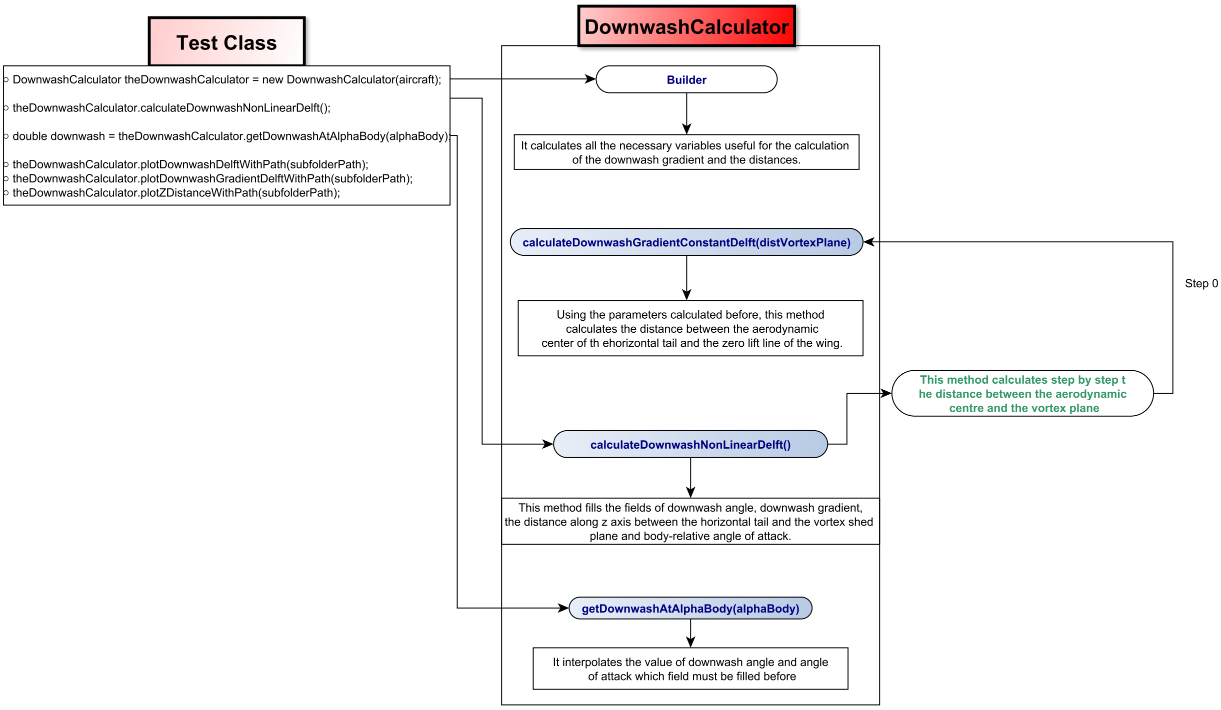
Figure 8.6: Arm definitions for downwash gradient evaluation.

Referring to the step 0, the distance $m\frac{b}{2}$ is the segment $\overline{CD}$. It's possible to calculate this distance geometrically as follow:

$$\overline{CD} = (\overline{CF} + \overline{FE}) * \cos i_w - \alpha_{0L} \tag{8.5}$$

$\overline{CF}$ is the distance along Z between the aerodynamic center of horizontal tail and the trailing edge of root airfoil of the wing.
Is possible to calculate the distance $\overline{FE}$ considering the triangle BFE. The side $\overline{BF}$ is the tistance along X axis between the aerodynamic center of horizontal tail and the trailing edge of root airfoil of the wing, while the angle between this side and the hypotenuse is $i_w - \alpha_{0L}$.

For each step the method is the same, but the difference is that the angle $i_w - \alpha_{0L}$ becomes $i_w - \alpha_{0L} - i\,\Delta\alpha + \epsilon$.



$$\epsilon = \frac{d\epsilon}{d\alpha_w}(\alpha_w - \alpha_{0_w})$$

Figure 8.7: Flow chart of the calculation of non-linear downwash angle.

8.3 Case Study

Listing 8.1 Downwash Test Class

```
import static java.lang.Math.toRadians;

import java.io.File;

import javax.measure.quantity.Angle;
import javax.measure.unit.NonSI;
import javax.measure.unit.SI;

import org.jscience.physics.amount.Amount;

import aircraft.OperatingConditions;
import aircraft.calculators.ACAnalysisManager;
import aircraft.components.Aircraft;
import aircraft.components.liftingSurface.LSAerodynamicsManager;
import aircraft.components.liftingSurface.LiftingSurface;
import configuration.MyConfiguration;
import configuration.enumerations.DatabaseReaderEnum;
import javafx.util.Pair;
import writers.JPADStaticWriteUtils;

public class prova {

    public static void main(String[] args) {

        // -----------------------------------------------------------------------
        // INITIALIZE TEST CLASS
        // -----------------------------------------------------------------------

        System.out.println("Initializing test class...");
        String folderPath = MyConfiguration.currentDirectoryString + File.separator
                + "out" + File.separator;
        String subfolderPath = JPADStaticWriteUtils.createNewFolder(folderPath
                + "Longitudinal_Static_Stability" + File.separator);
```

```
//---------------------------------------------------------------------------
// Default folders creation:

MyConfiguration.initWorkingDirectoryTree();


//---------------------------------------------------------------------------
// Operating Condition

OperatingConditions theConditions = new OperatingConditions();
theConditions.set_alphaCurrent(Amount.valueOf(toRadians(2.), SI.RADIAN));


//---------------------------------------------------------------------------
// Default Aircraft
Aircraft aircraft = Aircraft.createDefaultAircraft("ATR-72");
System.out.println("Default␣aircraft:␣" + aircraft.get_name() + "\n");


//---------------------------------------------------------------------------
// Wing and Tail
LiftingSurface theWing = aircraft.get_wing();
LiftingSurface horizontalTail = aircraft.get_HTail();


//---------------------------------------------------------------------------
// Aerodynamic managers
ACAnalysisManager theAnalysis = new ACAnalysisManager(theConditions);
theAnalysis.updateGeometry(aircraft);
LSAerodynamicsManager theLSAnalysis = new LSAerodynamicsManager(
                theConditions,
                theWing,
                aircraft
                );


aircraft.get_wing().setAerodynamics(theLSAnalysis);

aircraft.get_exposedWing().updateAirfoilsGeometryExposedWing( aircraft);
//---------------------------------------------------------------------------
// Set databases
theLSAnalysis.setDatabaseReaders(
                new Pair(DatabaseReaderEnum.AERODYNAMIC,
                                "Aerodynamic_Database_Ultimate.h5"),
                new Pair(DatabaseReaderEnum.HIGHLIFT,
                "HighLiftDatabase.h5")
                );


//---------------------------------------------------------------------------
// Angle of attack

Amount<Angle> alphaBody = theConditions.get_alphaCurrent();


// --------------------------------------------------------------------------
// LIFT CHARACTERISTICS
// --------------------------------------------------------------------------

LSAerodynamicsManager.CalcCLAtAlpha theCLWingCalculator = theLSAnalysis
                     .new CalcCLAtAlpha();

double cLIsolatedWing = theCLWingCalculator
                      .nasaBlackwellCompleteCurve(alphaBody);



// ------------------Downwash---------------


System.out.println("\n-----Start␣of␣downwash␣calculation-----\n" );

DownwashCalculator theDownwashCalculator = new DownwashCalculator(aircraft);

theDownwashCalculator.calculateDownwashNonLinearDelft();

theDownwashCalculator.plotDownwashDelftWithPath(subfolderPath);
```

```
        theDownwashCalculator.plotDownwashGradientDelftWithPath(subfolderPath);
        theDownwashCalculator.plotZDistanceWithPath(subfolderPath);
        System.out.println(" DONE PLOTTING DOWNWASH ANGLE vs ALPHA BODY");

        double downwash = theDownwashCalculator.getDownwashAtAlphaBody(alphaBody);
        Amount<Angle> downwashAmountRadiant = Amount
                                .valueOf(Math.toRadians(downwash), SI.RADIAN);
                System.out.println( "At alpha " + alphaBody
                                .to(NonSI.DEGREE_ANGLE)
            .getEstimatedValue() + " (deg) the downwash angle is (deg) = " + downwash );




        }

}
```

Chapter 9

Aircraft Longitudinal Static Stability

Citazione
citazione
– Autore

9.1 Aerodynamic Lift

9.1.1 Wing

9.1.2 Fuselage

9.1.3 Horizontal Tail

Elevator index of effectiveness

In order to evaluate the contribution to the longitudinal stability of horizontal tail it's necessary to consider the deflection of the elevator.

The variation of zero lift angle is not constant with the angle of deflection. So it's necessary to evaluate the tau factor which is defined as follows:

$$\tau_e = \frac{d\alpha_{0l}}{d\delta_e} \tag{9.1}$$

Introducing this parameter the lift coefficient of the horizontal tail can be rated as follows:

$$C_{L_H} = C_{L_0} + C_{L_{\alpha_H}}\alpha_H + C_{L_{\alpha_H}}\tau_e\delta_e \tag{9.2}$$

Considering a symmetrical horizontal tail, the term C_{L_0} is zero, so it's possible to express the lift coefficient in the following form:

$$C_{L_H} = C_{L_{\alpha_H}}\left(\alpha_H + \tau_e\delta_e\right) \tag{9.3}$$

In general the value of τ is constant until about 15 deg; after this value, due to the flow separation, the effectiveness of elevator decrease and consequently the product $\tau_e\delta_e$ that appears in the equation of lift coefficient.

The evaluation of tau is made by reading of external database, considering the following graphs.

$$\tau = \alpha_\delta\eta_\delta = \frac{\alpha_{\delta_{c_L}}}{\alpha_{\delta_{c_l}}}\alpha_{\delta_{c_l}}\eta_\delta \tag{9.4}$$

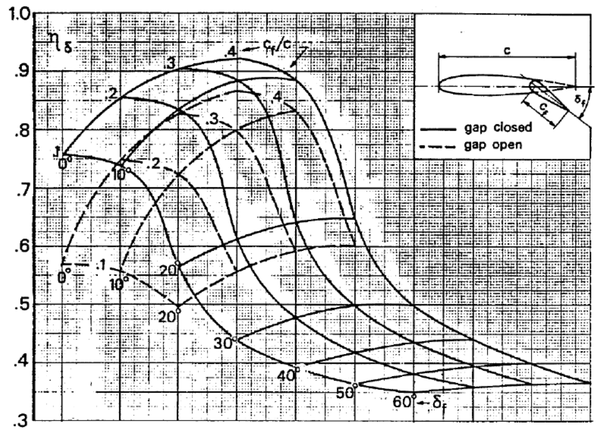


Figure 8.1: 2D efficiency correction for elevator.

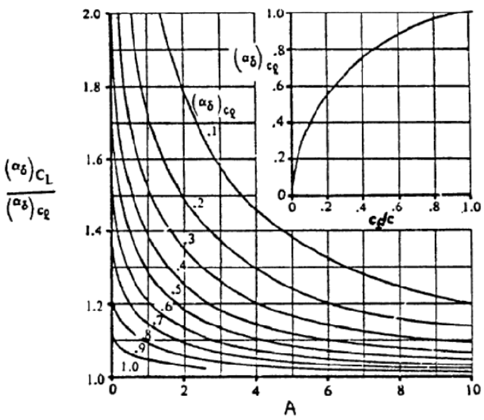


Figure 8.2: $\frac{d\alpha_{0l}}{d\delta_e}$ 2D and 3D correction.

9.1.4 Complete Aircraft

In order to evaluate the lift coefficient of the entire airplane it's possible to consider it as consisting of the following parts[56]:

- wing and fuselage
- horizontal Tail
- canard

It's important to consider the effectivness angles of attack in which the surfaces work. This is made considering the angles of incidence of the lifting surfaces and the downwash angle aft of the wing. An horizontal tail and a canard may be equiped with a trailing edge control surface. So in order to evaluate these contributes it' s important to know the angle of deflection δ of these control surfaces.
The calculation of the individual contributions it's reported in the relevant sections. In this section will be shown the method to evaluate the aircraft lift coefficient, known the single contributes.
For an aircraft with no cadard, the formula is the following:

$$C_L = C_{L_{wb}} + \frac{S_t}{S_w}\eta_t C_{L_t} \tag{9.5}$$

Where η_t is the ratio of dynamic pressure. In fact the dynamic pressure seen by horizontal tail differ from the free stream dynamic pressure due to two main reasons: the combination wing-fuselage and the presence of the propeller. The dynamic pressure of the tail depends on the location of the tail. If the tail is in the wake of the wing-body, the local dynamic pressure will be less than the freestream because the flow gradually looses its kinetic energy. While if the tail is in the slipstream of propeller, the local dynamic pressure may increase due to the power absorbed by the propeller.

9.2 Aerodynamic Drag

9.2.1 Wing

9.2.2 Fuselage

9.2.3 Horizontal Tail

9.3 Pitching Moments

9.3.1 Wing

9.3.2 Fuselage

9.3.3 Horizontal Tail

9.3.4 Propulsors

9.3.5 Stability Calculation

9.4 Java Class Architecture

9.5 User's Guide

9.6 Analysis Results

Chapter 10

Minor Works

Appendices

Appendix A

Java Reference

A.1 Inheritance, Overriding and Polymorphism

An extensive usage of inheritance, overriding and polymorphism has been made throughout the application.

Java inheritance can be defined as the process where one object acquires the properties of another. With the use of inheritance the information is made manageable in a hierarchical order. Inheritance has been used several times to define superclass-subclass relationship in order to ease the management of the instances of the subclasses and to group together a set of properties that each subclass must have. Some of the methods defined in the superclass are overridden by the subclasses to execute the proper action accordingly to the run-time object type.

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object. Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object. Polymorphism has been exploited in the design of methods which had to perform an action that could be valid for more than one object type. In such a case, the method signature contains a parameter which is of the superclass type; when invoking the method elsewhere in the application with a subclass instance as an argument at compile time, the compiler uses the method in the superclass to validate the statement in the "client" method. At run time, however, the JVM invokes the method which is defined in the subclass which overrides the superclass method. This behaviour is referred to as virtual method invocation, and the methods are referred to as virtual methods.

In this way it was possible to define a single method which was suitable for several objects types instead of defining a different method for each component or using the instanceof operator.

A.2 Packages

A Java package is a mechanism for organizing Java classes into namespaces, providing modular programming. Java packages can be stored in compressed files called JAR files, allowing classes to be downloaded faster as groups rather than individually. Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality.

A package provides a unique namespace for the types it contains. In general, a namespace is a container for a set of identifiers (also known as symbols, names). Namespaces provide a level of direction to specific identifiers, thus making it possible to distinguish between identifiers with the same exact name. For example, a surname could be thought of as a namespace that makes it possible to distinguish people who have the same given name. In computer programming, namespaces are typically employed for the purpose of grouping symbols and identifiers around a particular functionality.

A.3 Documentation

The doc comments, which are a particular type of comments provided by Java, are automatically recognized by the most popular IDE programs; this helps the developer to quickly get an idea of the actions performed by each method, the parameter which has to be passed to it and the what it returns to the user.

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.

client code the code where the code in question will be effectively exploited.

Graphical User Interface In computing, a Graphical User Interface is a type of interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels or text navigation.

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.

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.

reflection In computer science, reflection is the ability of a computer program to examine (see type introspection) and modify the structure and behavior (specifically the values, meta-data, properties and functions) of the program at runtime.

serialization In computer science, in the context of data storage, serialization is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection link) and reconstructed later in the same or another computer environment.

Table a collection that associates an ordered pair of keys, called a row key and a column key, with a single value. A table may be sparse, with only a small fraction of row key / column key pairs possessing a corresponding value.

Unified Modeling Language The Unified Modeling Language (UML) is a general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system.

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.

wrapper function A wrapper function is a subroutine in a software library or a computer program whose main purpose is to call a second subroutine or a system call with little or no additional computation..

Acronyms

ACRF Aicraft Construction Reference Frame.

AIAA American Institute of Aeronautics and Astronautics.

BRF Body Reference Frame.

FAA Federal Aviation Administration.

GC Gravity Center.

GNC Guidance Navigation and Control.

GUI Graphical User Interface.

JPAD Java Aircraft Design.

MAC Mean Aerodynamic Chord.

MAPE Mean Absolute Percentage Error.

MLW Maximum Landing Weight.

MSL Mean Sea Level.

MTOW Maximum Take Off Weight.

MZFW Maximum Zero Fuel Weight.

UML Unified Modeling Language.

List of symbols

$(\,)_{\mathbf{H}}$ quantity related to the horizontal tail.

$(\,)_{\mathbf{LG}}$ quantity related to the landing gear.

$(\,)_{\mathbf{N}}$ quantity related to the nacelle.

$(\,)_{\mathbf{S}}$ quantity related to systems.

$(\,)_{\mathbf{V}}$ quantity related to the vertical tail.

$(\,)_{\mathbf{F}}$ quantity related to the fuselage.

$(\,)_{\mathbf{WF}}$ quantity related to the wing-fuselage configuration.

D aerodynamic drag.

CG Center of Gravity.

$\vec{g}$ gravitational acceleration.

$i_{\mathbf{H}}$ the angle between the horizontal tail root chord and the ACRF x-axis.

$i_{\mathbf{W}}$ the angle between the wing root chord and the ACRF x-axis.

L aerodynamic lift.

T Thrust.

M Mach number.

$\boldsymbol{A\!\!\!R}$ aspect ratio.

c chord.

d diameter.

l length.

m mass, in kg or lb.

$n_{\mathbf{lim}}$ limit load factor.

$n_{\mathbf{ult}}$ ultimate load factor.

b span.

S surface.

Λ sweep.

λ taper ratio.

t thickness.

V scalar velocity.

W weight, in Newtons.

m_{W} wing mass.

q dynamic pressure.

Re Reynolds number (evaluated with respect to $\bar{c}$).

$\alpha_{\mathbf{W}}$ angolo d'attacco riferito alla corda di radice dell'ala.

β sideslip angle.

ρ air density.

Bibliography

[1] I. H. Abbott and A. E. von Doenhoff. *Theory of Airfoil Sections*. Dover, 1959.

[2] Joe Walnes et al. *XStream*. URL: http://xstream.codehaus.org/.

[3] Guillaume Alleon. *occjava*. URL: http://jcae.sourceforge.net/occjava.html.

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

[5] Various Authors. *ATR 72. Aircraft description*. URL: http://www.atraircraft.com/products/list.html.

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

[7] Various Authors. *Boeing 747. Aircraft description*. URL: http://www.boeing.com.

[8] Various Authors. *ESDU-76003. Geometrical properties of cranked and straight tapered wing planforms*. London, 2001.

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

[10] Various Authors. *Guava*. URL: https://github.com/google/guava.

[11] Various Authors. *JFace*. URL: https://wiki.eclipse.org/JFace.

[12] Various Authors. *Model View Controller*. URL: https://en.wikipedia.org/wiki/Model%C3%A2%C2%80%C2%93view%C3%A2%C2%80%C2%93controller.

[13] Various Authors. *The Standard Widget Toolkit*. URL: https://www.eclipse.org/swt/.

[14] Various Authors. *Window Builder*. URL: https://eclipse.org/windowbuilder/.

[15] Azeckoski. *ReflectUtils*. URL: https://code.google.com/p/reflectutils/.

[16] A. et al. Bilgin. *Graphviz*. URL: http://www.graphviz.org/.

[17] J. A. Jr Blackwell. *A Finite-Step Method for Calculation of Theoretical Load Distributions for Arbitrary Lifting-Surface Arrangements at Subsonic Speeds*. Washington, D.C., 1969.

[18] M. Calcara. *Elementi di dinamica del velivolo*. Napoli: Edizioni CUEN, 1988. ISBN: 9788871460062. URL: http://books.google.it/books?id=HqBoAQAACAAJ.

[19] Jean-Marie Dautelle. *Jscience*. URL: http://jscience.org/.

[20] M. Drela and H. Youngren. *XFoil, Subsonic Airfoil Development System*. 2008. URL: http://web.mit.edu/drela/Public/web/xfoil/.

[21] The Apache Software Foundation. *Apache Commons Math*. URL: http://commons.apache.org/proper/commons-math/.

[22] The Apache Software Foundation. *Apache POI*. URL: http://poi.apache.org/.

[23] The Apache Software Foundation. *Apache projects*. URL: http://www.apache.org/.

[24] «Full-configuration drag estimation». In: Journal of Aircraft 47 (4) (2010).

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

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

[27] R.R Gilruth and M.D. White. *NACA TR 711 - Analysis And Prediction Of Longitudinal Stability of Airplanes*.

[28] The HDF Group. *HDF*. URL: http://www.hdfgroup.org/HDF5/.

[29] M. Hepperle. *Applet* JavaFoil. 2006. URL: http://www.mh-aerotools.de/airfoils/javafoil.htm.

[30] W.F. Hilton. *High-speed Aerodynamics*. Longmans, Green, 1951. URL: https://books.google.it/books?id=%5C_RRFAAAAMAAJ.

[31] D. E. Hoak et al. *The USAF Stability and Control DATCOM*. Tech. rep. (Revised 1978). 1960.

[32] D. Howe. *Aircraft conceptual design synthesis*. Aerospace Series. Professional Engineering Publishing, 2000. ISBN: 9781860583018. URL: https://books.google.it/books?id=QJZTAAAAMAAJ.

[33] D. Howe. *Aircraft Loading and Structural Layout*. AIAA Education Series, 2004.

[34] J. Huwaldt. *1976 Standard Atmosphere Program*. URL: http://thehuwaldtfamily.org/java/Applications/StdAtmosphere/StdAtmosphere.html.

[35] E. N. Jacobs and K. E. Ward. *Interference of wing and fuselage from tests of 209 combinations in the NACA variable-density tunnel*. Tech. rep. 1936.

[36] L.R. Jenkinson, D. Rhodes, and P. Simpkin. *Civil jet aircraft design*. AIAA education series. Arnold, 1999. ISBN: 9780340741528. URL: https://books.google.it/books?id=6tMeAQAAIAAJ.

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

[38] I. Kroo and R. Shevell. *Aircraft Design: Synthesis and Analysis*. 1999.

[39] S. Ledru and C. Denizet. *JLatexMath*. URL: http://forge.scilab.org/index.php/p/jlatexmath/.

[40] W.H. Mason. «Analytic Models for Technology Integration in Aircraft Design». In: AIAA/AHS/ASEE (1990).

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

[42] H. Multhopp. *Aerodynamics of Fuselage*. 1942.

[43] M. M. Munk. *Aerodynamic Forces of Airship Hulls*. Tech. rep. 1924.

[44] Marcello R. Napolitano. *Aircraft Dynamics: From Modeling to Simulation*. John Wiley, 2012. ISBN: 9780470626672. URL: http://www.amazon.com/Aircraft-Dynamics-From-Modeling-Simulation/dp/0470626674.

[45] R. Nelson. *Flight Stability and Automatic Control*. McGraw-Hill, 1989.

[46] R.C. Nelson. *Flight Stability and Automatic Control*. Aerospace Science & Technology. McGraw-Hill International Editions, 1998. ISBN: 9780071158381. URL: https://books.google.it/books?id=Uzs8PgAACAAJ.

[47] L.M. Nicolai and G. Carichner. *Fundamentals of Aircraft and Airship Design*. AIAA education series v. 1. American Institute of Aeronautics and Astronautics, 2010. ISBN: 9781600867514. URL: https://books.google.it/books?id=qA0oAQAAMAAJ.

[48] M. Nita and D. Scholz. «Estimating Oswald Factor From Basic Aircraft Geometrical Parameters». In: *none* (2012).

[49] E. Obert. *Aerodynamic Design of Transport Aircraft*. IOS Press, 2009. ISBN: 9781607504078. URL: https://books.google.it/books?id=axPvAgAAQBAJ.

[50] C. D. Perkins and R. E. Hage. *Airplane Performance Stability and Control*. New York: John Wiley & Sons, 1949. ISBN: 9780471680468. URL: http://books.google.it/books?id=DHxTAAAAMAAJ.

[51] W. F. Phillips. «Lifting-Line Analysis for Twisted Wings and Washout-Optimized Wings». In: *Journal of Aircraft* Vol. 41.No. 1 (Jan.–Feb. 2004).

[52] Warren F. Phillips. *Mechanics of Flight*. John Wiley and Sons, 2004. URL: https://books.google.it/books/about/Mechanics_of_Flight.html?id=6-_iGbJHM-8C&redir_esc=y.

[53] D.P. Raymer. *Aircraft Design: A Conceptual Approach*. 1992.

[54] D.P. Raymer. *Aircraft Design / RDS-Student: A Conceptual Approach*. AIAA Education Series. Amer Inst of Aeronautics &, 2013. ISBN: 9781600869211. URL: https://books.google.it/books?id=bvxllgEACAAJ.

[55] J. Roskam. *Airplane Design*. Airplane Design pt. 5. DARcorporation, 1999. ISBN: 9781884885501. URL: https://books.google.it/books?id=mMU47Ld7yQkC.

[56] J. Roskam. *Airplane Design*. Airplane Design pt. 8. DARcorporation, 2002. ISBN: 9781884885556. URL: https://books.google.it/books?id=GIHHFkd829cC.

[57] J. Roskam. *Airplane Flight Dynamics and Automatic Flight Controls*. DARcorporation, 2001.

[58] J. Roskam. *Airplane Flight Dynamics and Automatic Flight Controls*. Airplane Flight Dynamics and Automatic Flight Controls. Darcorporation, 2003. ISBN: 9781884885181. URL: http://books.google.it/books?id=pWUAzy87zLYC.

[59] M.H. Sadraey. *Aircraft Design: A Systems Engineering Approach*. Aerospace Series. Wiley, 2012. ISBN: 9781118352809. URL: https://books.google.it/books?id=VT-Tc3Tx5aEC.

[60] OPEN CASCADE S.A.S. *OpenCASCADE*. URL: http://www.opencascade.org/.

[61] David Schmidt. *Modern Flight Dynamics*. McGraw-Hill, 2012. ISBN: 97800711339811.

[62] P.M. Sforza. *Commercial Airplane Design Principles*. Elsevier Science, 2014. ISBN: 9780124199774. URL: https://books.google.it/books?id=knhHAgAAQBAJ.

[63] Abe Silverstein and S. Katzoff. *Design Charts for Predicting Downwash Angles and Wake Characteristics behind Plain and Flapped Wings*. Tech. rep. 1939.

[64] E. Torenbeek. *Advanced Aircraft Design: Conceptual Design, Technology and Optimization of Subsonic Civil Airplanes*. Aerospace Series. Wiley, 2013. ISBN: 9781118568095. URL: https://books.google.it/books?id=C0gUyoYewhoC.

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

[66] 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. ISBN: 9789024727247. URL: https://books.google.it/books?id=NG2%5C_qiSjmMEC.

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

[68] S. R. Vukelich and J. E. Williams. *The USAF Stability and Control Digital Datcom*. AFFDL-TR-79-3032. Updated by Public Domain Aeronautical Software 1999. Apr. 1979. Volume I.