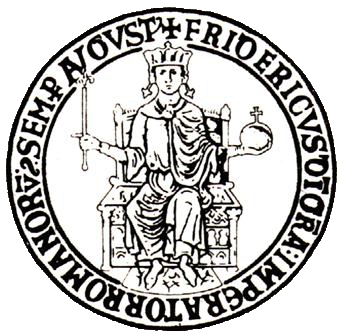
**Università Degli Studi Di Napoli “Federico II”**



**Scuola Politecnica e delle Scienze di Base**

**Dipartimento di Ingegneria Industriale**

**Tesi di Laurea Triennale in Ingegneria Aerospaziale**

**A Java Software for**

**Aircraft Flight Dynamics**

**Calculation**

|  |  |
| --- | --- |
| **Relatore** | **Candidato** |
| **Ch.mo PROF. Ing.**  **Agostino De Marco** | **Giuseppe Scarlatella** |
|  | **N35/678** |

**Anno accademico 2013/2014**

INDEX

[SUMMARY 2](#_Toc433698045)

[1 INTRODUCTION 3](#_Toc433698046)

[1.1 LINEARIZED EQUATIONS SYSTEM 3](#_Toc433698047)

[1.1.1 LONGITUDINAL AND LATER-DIRECTIONAL EQUATIONS SYSTEMS 5](#_Toc433698048)

[1.1.2 STABILITY AND CONTROL DERIVATIVES 8](#_Toc433698049)

[1.2 AIRCRAFT DYNAMICS 10](#_Toc433698050)

[1.2.1 LONGITUDINAL DYNAMICS 10](#_Toc433698051)

[1.2.2 LATERAL-DIRECTIONAL DYNAMICS 11](#_Toc433698052)

[2 SOFTWARE 12](#_Toc433698053)

[2.1 STABILITY DERIVATIVES CALCULATOR CLASS 13](#_Toc433698054)

[2.1.1 STABILITY AND CONTROL DERIVATIVES 13](#_Toc433698055)

[2.1.2 LONGITUDINAL AND LATERAL-DIRECTIONAL MATRICES 13](#_Toc433698056)

[2.1.3 FUNCTIONS AND SUBROUTINES 15](#_Toc433698057)

[2.2 DYNAMIC STABILITY CALCULATOR CLASS 16](#_Toc433698058)

[2.2.1 EIGENVALUES AND EIGENVECTORS 16](#_Toc433698059)

[2.2.2 DYNAMIC CHARACTERISTICS 17](#_Toc433698060)

[2.3 FLIGHT DYNAMICS MANAGER CLASS 18](#_Toc433698061)

[2.3.1 GLOBAL ENVIRONMENT 18](#_Toc433698062)

[2.3.2 INPUT MANAGEMENT CLASSES 20](#_Toc433698063)

[2.3.3 CONSTRUCTOR METHOD 23](#_Toc433698064)

[2.3.4 CALCULATOR METHOD 23](#_Toc433698065)

[2.3.5 MAIN METHOD 26](#_Toc433698066)

[3 BOEING 747 – TEST 27](#_Toc433698067)

[3.1 CONDITIONS OF FLIGHT 27](#_Toc433698068)

[3.2 OUTPUT 29](#_Toc433698069)

[3.3 FINAL REMARKS 34](#_Toc433698070)

[BIBLIOGRAPHY 36](#_Toc433698071)

[NOTES 36](#_Toc433698072)

# SUMMARY

This Java program is able to determine the characteristics of dynamic stability for small perturbations in longitudinal and lateral-directional motion of an aircraft, from a set of input data supplied by the manufacturer or previously estimated.

Small perturbations allow to linearize the equations of motion system, generating a system of linear equations. This linear system can be decomposed into two subsystems: longitudinal dynamic equations and lateral-directional dynamic equations.

The program consists of three main *classes*: one for the stability and control derivatives calculation and the related matrices generation, another one is for eigenvalues and eigenvectors management, as well as the free response to small perturbations characteristics, and the last one is a *calculator* class, executed in the *main*. After reading from file, the calculator class estimates derivatives and free response characteristics, saves all results in *member variables* and prints them on the screen simultaneously.

This program was built in a “bottom*-up*" perspective. Thanks to its setting from *Object-Oriented Programming*, it can be easily implemented within a more complex code. Its *methods* and its *attributes* can be accessed at any time and can interact with other classes mostly oriented to the complete aircraft study.

# INTRODUCTION

An aircraft is a deformable solid indeed and is not allowed to ignore the effects of its articulated subsystems, but is to be considered acceptable the hypothesis of rigid body for the study of flight dynamics.

In addition, the external forces and moments acting on the vehicle are not simple configuration and motion functions, especially aerodynamic actions.

At any given moment, they can be calculated if you know the current values of speed of the relative wind and angles of trim, but only with a certain approximation.

The level of detail necessary to determine the entity is the dominant theme of any formulation of the equations of motion.

The complete motion equations of a rigid aircraft in the atmosphere, subject to the aerodynamic actions, propulsion and weight force, are nonlinear and coupled. They can only be solved numerically and do not conceptually lend themselves to illustrate the dependence of stability and controllability characteristics of the aircraft from its geometric, inertial and aerodynamic properties.

## LINEARIZED EQUATIONS SYSTEM

Mostly we can learn about aircraft behavior by analyzing the linear approximations of the complete motion equations. The solutions of the linearized equations are valid for small perturbations around to a given flight condition of reference, and they can be analyzed with tools well known by mathematical theory of dynamical systems.

The particular formulation of linear problem will depend on the reference condition choice. The nominal condition adopted lends itself particularly to the linearization procedure. It is a longitudinally symmetrical not accelerated movement along a straight line, at constant speed and rate of steady climb (in particular, zero), assuming the Earth as flat, namely a balanced flight condition, that perfectly meets the complete equations of motion.

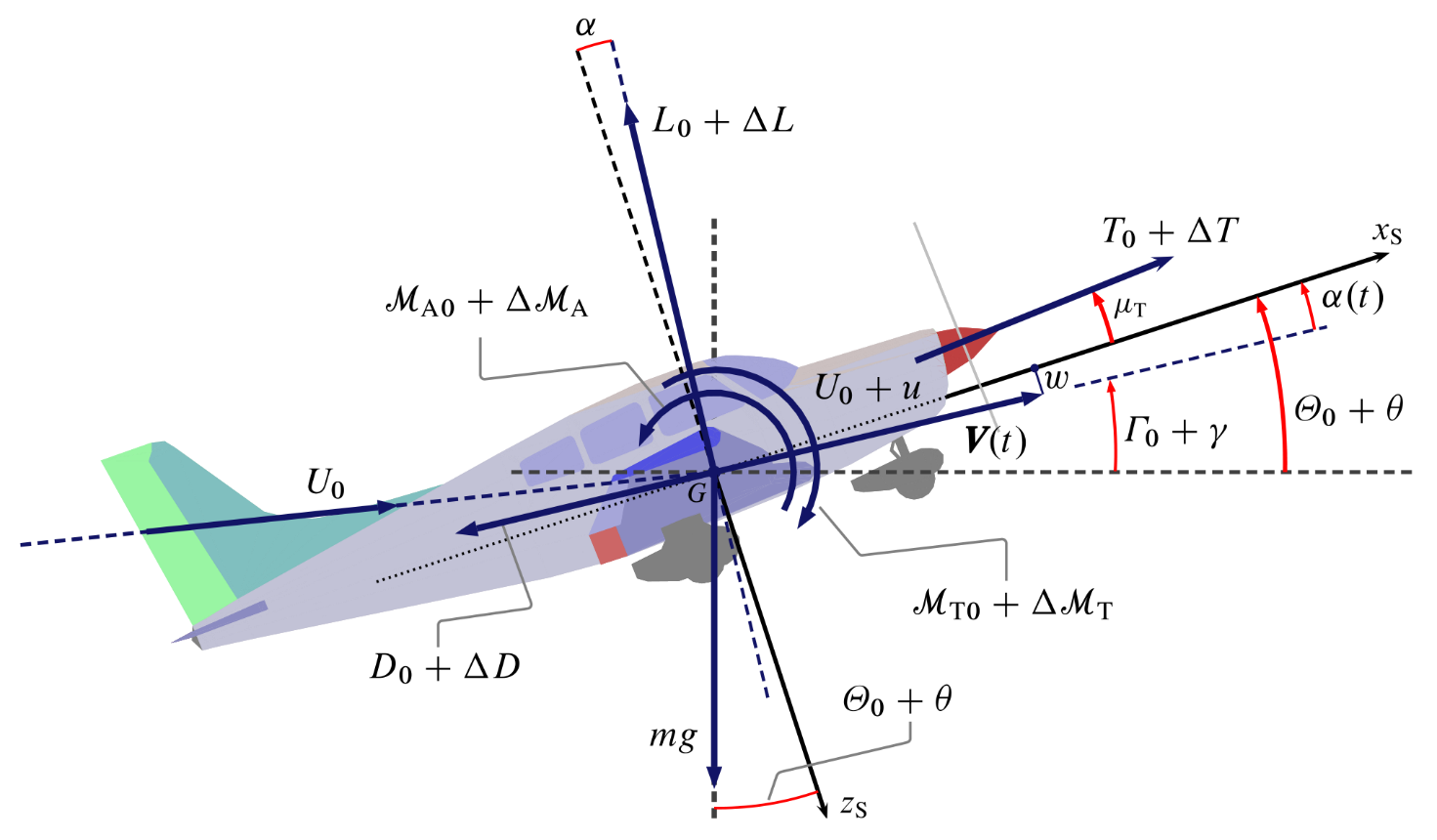
Obviously, there are situations in which this approach has limitations, which occurs for all those motions characterized by large changes of the state variables or from not small assets and nonlinear aerodynamic effects.

Figure 1.1- perturbed longitudinal motion (De Marco & Coiro, 2015)

Despite everything, the accuracy with which we can apply the small perturbations theory proves itself nevertheless acceptable in a wide spectrum of possible situations. This is essentially due to two circumstances:

* for a wide range of flight conditions of practical importance, aerodynamic forces and moments retain effectively a linear dependence with status and control variables;
* normal flight situations are actually perturbed motions of a certain amplitude, not infinitesimal, which correspond, indeed, to the combined effect of small linear and angular speed perturbations;

In fact, relatively small perturbation of the state variables can lead to flight conditions particularly 'violent' and this should normally be avoided.

### LONGITUDINAL AND LATER-DIRECTIONAL EQUATIONS SYSTEMS

The longitudinal and lateral-directional linearized equations systems can be put in the following simplified form:

***x'****LON =* **A***LON* ***x****LON +* **B***LON* ***u****LON*

***x'****LD =* **A***LD* ***x****LD +* **B***LD* ***u****LD*

where **x**LON and **x**LD are perturbations of the state variables vectors, while **u**LON and **u**LD are perturbations of the control parameters vectors.

To better highlight the decoupling of longitudinal and lateral-directional motions, we rewrite the equations provided in matrix form:

≈ ∙ + ∙

Usually **x** vectors are composed by dynamic and cinematic state variables such as: *u*, *v*, *w*, *q*, *p*, *r*, *X*E,G, *Y*E,G, *Z*E,G, *θ*, *ϕ*, *ψ*:

***x*** = = ; ***u*** ==

As anticipated, we can work in simplified hypothesis, keeping a general validity for many disparate cases. In particular we assume our nominal condition as a longitudinally symmetrical (*v*0=0) not accelerated movement along a straight line (*p*0 = *q*0 = *r*0 = 0), at constant speed and rate of steady climb (in particular, null), null angular velocity (*ϕ*0 = 0), null initial prow angle (*ψ*0 = 0) and assuming the Earth as flat.

Working under these hypotheses, our state vectors and coefficient matrices seem very simplified:

***x****LON* =  ; ***u****LON*  =

***x****LD* =  ; ***u****LD*  =

Matrices **A** and **B** are usually an articulated stability and control derivatives assembling but, adopting our simplified nominal condition, many coefficients get lightened. In particular, matrices **A** and **B** get reduced to (4×4) and (4×2) order matrices, assuming their well-known structure:

**A**LON  =

**A**LD  =

**B**LON  =

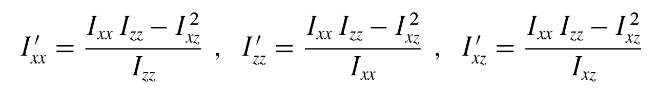
**B**LD  =

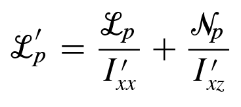
### STABILITY AND CONTROL DERIVATIVES

The stability and control derivatives are the matrices **A** and **B** constituents. They derive directly from theories of aerodynamics and flight mechanics, and we generally will estimate them respectively to aircraft mass (in case of forces) or moments of inertia (in case of moments of forces). They are all reported in the following lists.

**N.B.**

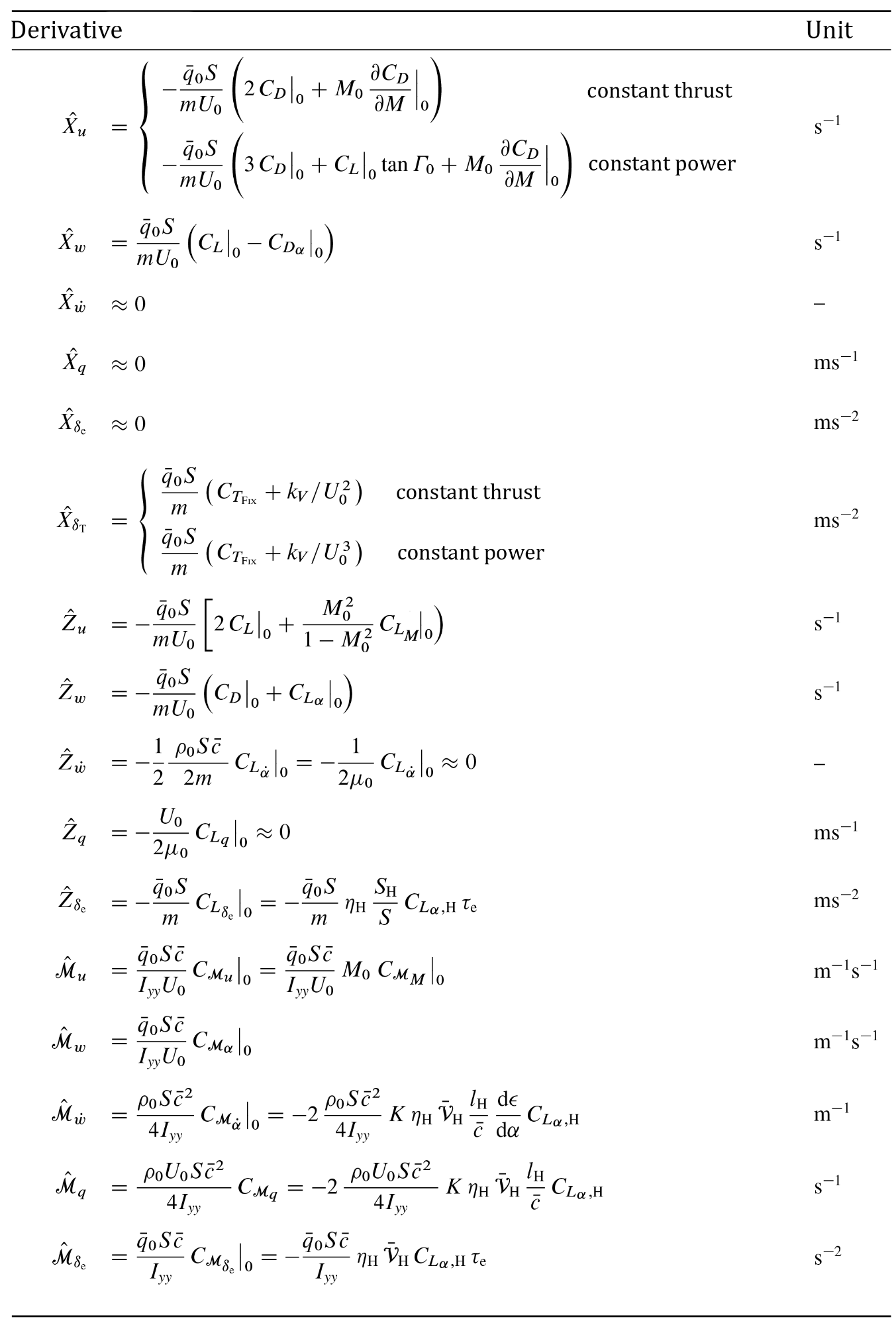
The **A**LD and **B**LD coefficients are also known as “*primed derivatives*”, and they result quite different from dimensional derivatives reported in the list on the left. *Primed derivatives* are obtained dividing dimensional derivatives for a linear combination of their moments and  
product of inertia:



 **i.e.**

Primed derivatives consider combined effect between Roll and Yaw.

Figure 1.2 lateral-directional stability and control derivatives (De Marco & Coiro, 2015)

Figure 1.3 longitudinal stability and control derivatives (De Marco & Coiro, 2015)

## AIRCRAFT DYNAMICS

The linear equations systems can be studied with classical methods of the *linear time-invariant* (LTI) theory. Study the system response will not be in our interest, rather we will just evaluate the dynamic *open-loop* characteristics, whose validity is extended to all specific cases in the presence of external forcing.

### LONGITUDINAL DYNAMICS

It’s our interest to analyze longitudinal *open-loop response* to estimate basic characteristics such as *damping coefficient* and *natural frequency*. To do so, we have to determine the characteristic polynomial of longitudinally symmetrical motion, which is the characteristic polynomial of dynamic matrix **A**LON indeed, and, in terms of the eigenvalues ​​of the matrix, can be expressed in the form:

Δ LON(s)= (s – λ SP)(s – λ\*SP)(s – λ PH)(s – λ\* PH)

λ SP , λ\*SP } = σ SP ± i ω SP ; λ PH , λ\*PH } = σ PH ± i ω PH

its roots are two pairs of complex and conjugated eigenvalues (λ PH , λ\*PH) and  
(λ SP , λ\*SP), which constitute two modal components of the dynamic, respectively known as ***phugoid*** (or *long-term*) and ***short-period*** modes. [[1]](#endnote-1)

Introducing *natural frequency* and *damping coefficient*:

;

;

where ξ = SP, PH.

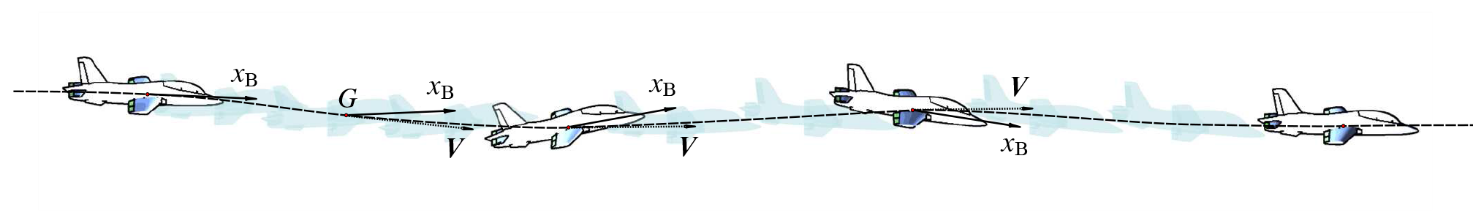


Figure 1.4 typical **short-period** oscillation, with changes on both **α** and **θ** (De Marco & Coiro, 2015)

### LATERAL-DIRECTIONAL DYNAMICS

Similarly, we will analyze lateral-directional *open-loop response*. The matrix **A**LD characteristic equation has the following roots:

λ ROLL = σ ROLL ; λ SPIRAL = σ SPIRAL

λ DR , λ\* DR } = σ DR ± i ω DR

They are two real roots and a complex and conjugated one (λ DR , λ\* DR), which constitute three modal components of the dynamic, respectively known as ***roll***, ***spiral*** and ***dutch-roll*** modes.

Introducing *natural frequency* and *damping coefficient* for dutch-roll mode:

;

;

# SOFTWARE

As a software for *Flight Dynamics Calculation*, our program consists of a series of routines able to handle sufficiently all the information related to the aircraft dynamics, which includes the stability derivatives calculation and the system response characteristics.

The program has been written in Java programming language, that is *concurrent*, *class-based* but, mostly, *object-oriented*: “*In the object-oriented (OO) paradigm, a program consists of interacting objects. An object encapsulates data and algorithms. Data defines the state of an object. Algorithms define the behavior of an object. An object communicates with other objects by sending messages to them*.” (Sharan, 2014).

This is exactly the philosophy that our program embraces. In fact, it is organized in three main *classes*, each of which treats a particular aspect of the global analysis:

* *StabilityDerivativesCalc.java*: calculator of stability derivatives and matrices **A**LON, **B**LON, **A**LD, **B**LD;
* *DynamicStabilityCalculator.java*: calculator of eigenvalues, eigenvectors and response characteristics;
* *FlightDynamicsManager.java*: *Java Main Method* (*JVM*), *I/O* operator and *CalculateAll* method.

Our approach is to describe briefly the function of each class and their structure, showing some excerpts of code to better contextualize.

## STABILITY DERIVATIVES CALCULATOR CLASS

The class *StabilityDerivativesCalc.java* employs more than forty different routines to estimate stability (and control) derivatives and to build longitudinal and lateral-directional matrices.

### STABILITY AND CONTROL DERIVATIVES

The following type of *procedures* is based on aerodynamics theorems and considerations on flight mechanics. We will now see an example.

**EXAMPLE 2.1**

*Xªu\_CT:*

**public** **static** **double** calcXªu\_CT (**double** rho0, **double** surf, **double** mass, **double** u0,

**double** q0, **double** cd0, **double** m0, **double** cdM0) {

**return** -q0\*surf\*(2\*cd0 + m0\*cdM0)/(mass\*u0);

}

Figure 2.1 - "Xªu\_CT" calculation procedure

This type of method estimates all our stability and control derivatives.

### LONGITUDINAL AND LATERAL-DIRECTIONAL MATRICES

This type of *procedures* is designed for longitudinal and lateral-directional matrices assembling.

**EXAMPLE 2.2**

*Building ALON matrix*:

**A**LON  =

**public** **static** **double**[][] build\_A\_Lon\_matrix (Propulsion propulsion\_system,

**double** rho0, **double** surf, **double** mass, **double** cbar, **double** u0, **double** q0,

**double** cd0, **double** m0, **double** cdM0, **double** cl0, **double** cdAlpha0, **double** gamma0,

**double** theta0\_rad, **double** clAlpha0, **double** clAlpha\_dot0, **double** cMAlpha0,

**double** cMAlpha0\_dot, **double** clQ0, **double** iYY, **double** cM\_m0, **double** cMq) {

**double** [][] aLon = **new** **double** [4][4];

// Propulsion type in the Xªu calculation

**switch** (propulsion\_system)

{

**case** ***CONSTANT\_TRUST***:

aLon [0][0] = *calcX\_u\_CT* (rho0, surf, mass, u0, q0, cd0, m0,

cdM0);

**break**;

**case** ***CONSTANT\_POWER***:

aLon [0][0] = *calcX\_u\_CP* (rho0, surf, mass, u0, q0, cd0, m0,

cdM0, cl0, gamma0);

**break**;

**default**:

aLon [0][0] = *calcX\_u\_CT* (rho0, surf, mass, u0, q0, cd0, m0,

cdM0);

**break**;

}

**double** k = *calcM\_w\_dot* (rho0, mass, surf, cbar, iYY, cMAlpha0\_dot)/(1 –

*calcZ\_w\_dot* (rho0, surf, mass, cbar, clAlpha\_dot0));

// Construction of the Matrix [A Lon]

aLon [0][1] = *calcX\_w*(rho0, surf, mass, u0, q0, cl0, cdAlpha0);

aLon [0][2] = 0;

aLon [0][3] = -(9.8100)\*Math.*cos*(theta0\_rad);

aLon [1][0] = *calcZ\_u* (rho0, surf, mass, u0, q0, m0, cl0)/(1 –

*calcZ\_w\_dot* (rho0, surf, mass, cbar, clAlpha\_dot0));

aLon [1][1] = *calcZ\_w* (rho0, surf, mass, u0, q0, m0, cd0, clAlpha0)/(1 –

*calcZ\_w\_dot* (rho0, surf, mass, cbar, clAlpha\_dot0));

*[…]*

aLon [2][3] = -k\*(9.8100)\*Math.*sin*(theta0\_rad);

aLon [3][0] = 0;

aLon [3][1] = 0;

aLon [3][2] = 1;

aLon [3][3] = 0;

**return** aLon;

}

Figure 2.2 - "[A\_Lon]" building procedure

**EXAMPLE 2.3**

*Primed derivatives Calculation*:

Figure 2.3 - "Primed Derivative" calculation for [A\_Ld] and [B\_Ld] assembling

// Inertia coefficient calculation

**double** i1 = iXZ/iXX;

**double** i2 = iXZ/iZZ;

// Primed Derivatives calculation

**double** Y\_beta\_1 = *calcY\_beta* (rho0, surf, mass, u0, q0, cyBeta);

**double** Y\_p\_1 = *calcY\_p* (rho0, surf, mass, u0, q0, bbar, cyP);

**double** Y\_r\_1 = *calcY\_r* (rho0, surf, mass, u0, q0, bbar, cyR);

**double** L\_beta\_1 = (*calcL\_beta* (rho0, surf, bbar, iXX, u0, q0, cLBeta) +

i1\**calcN\_beta* (rho0, surf, bbar, iZZ, u0, q0, cNBeta))/(1-i1\*i2);

*[…]*

### FUNCTIONS AND SUBROUTINES

In this category, we have many *subroutines* designed to estimate some flight characteristics, such as the *dynamic pressure*, recalled inside our main routines.

**EXAMPLE 2.4**

*Dynamic Pressure*:

**public** **static** **double** calcDynamicPressure(**double** rho0, **double** u0) {

**return** 0.5\*rho0\*Math.*pow*(u0,2);

}

Figure 2.4 - "Dynamic Pressure" calculation

**EXAMPLE 2.5**

*Propulsive Regime*:

**public** **enum** Propulsion { ***CONSTANT\_TRUST***, ***CONSTANT\_POWER***, ***CONSTANT\_MASS\_FLOW***,  
 ***RAMJET*** }

Figure 2.5 - "Propulsive Regime" Enum type declaration

We have declared a new enumeration type called *Propulsion*, which describes our propulsive regime. This choose influences and calculation by a *switch case* during **A**LON and **B**LON calculation (*see pag.14*).

## DYNAMIC STABILITY CALCULATOR CLASS

The class *DynamicStabilityCalculator.java* employs two routines to build the eigenvectors and an eigenvalues matrix and other five different routines to estimate dynamic characteristics, such as *damping coefficient* or *natural frequency*.

### EIGENVALUES AND EIGENVECTORS

The eigenvalues calculation is fundamental to get the dynamic characteristics. We obtain them by using a routine named *buildEigenValuesMatrix*, that includes many other *routines* imported from a package, *org.apache.commons.math3* (Apache Commons, s.d.), which allows you to generate two vectors, from matrices as **A**LON and **B**LON, each one containing respectively the real and imaginary roots parts from the characteristic polynomial. Finally, these are reported in a matrix (in our case 4x2) which contains on each line the real and imaginary part of each eigenvector.

**EXAMPLE 2.6**

*buildEigenValuesMatrix*:

Figure 2.6 - Eigenvalues Matrix composition procedure

**public** **static** **double**[][] buildEigenValuesMatrix (**double** aMatrix[][]) {

RealMatrix aLonRM = MatrixUtils.*createRealMatrix*(aMatrix);

EigenDecomposition aLonDecomposition = **new** EigenDecomposition(aLonRM);

**double**[] reEigen = aLonDecomposition.getRealEigenvalues();

**double**[] imgEigen = aLonDecomposition.getImagEigenvalues();

**double** [][] lambda\_Matrix = **new** **double** [4][2];

**for** (**int** i=0 ; i < 4 ; i++) {

lambda\_Matrix[i][0] = reEigen [i];

lambda\_Matrix[i][1] = imgEigen [i];

}

**return** lambda\_Matrix;

}

On each row of the matrix we have *real part* and *imaginary part* (separated).

Equally important it is to get the eigenvectors obtained through a procedure called *buildEigenVector*, which is recalled as many times as the number of the eigenvectors to calculate, specifying each time the index for the ith eigenvector.

**EXAMPLE 2.7**

*buildEigenVector:*

Figure 2.7 – An Eigenvector composition procedure

**public** **static** RealVector buildEigenVector (**double** aMatrix[][], **int** index) {

RealMatrix aRM = **new** Array2DRowRealMatrix(aMatrix);

EigenDecomposition eigDec = **new** EigenDecomposition(aRM);

RealVector eigVec = eigDec.getEigenvector(index);

**return** eigVec;

}

### DYNAMIC CHARACTERISTICS

These *procedures* are based on many classical methods from the *linear time-invariant* (LTI) theory. They are designed to calculate all the dynamic characteristics concerning an *open-loop* response and equally valid in all the other specific cases. They use as inputs all the elements from the ith line of the eigenvalues matrix previously calculated.

**EXAMPLE 2.8**

*Damping coefficient:*

**public** **static** **double** calcZeta (**double** sigma, **double** omega) {

**return** Math.*sqrt*( 1 / ( 1 + Math.*pow*( omega/sigma , 2 )));

}

Figure 2.8 - "Damping Coefficient" calculation procedure

This is an *exact* value of mode characteristics (not *approximate*).

## FLIGHT DYNAMICS MANAGER CLASS

The class *FlightDynamicsManager.java* contains all the necessary *procedures* to perform the dynamic stability calculation of the aircraft, reading from file and storing the read data as *global variables*, performing the procedures imported from the classes previously treated and printing their results.

### GLOBAL ENVIRONMENT

A *global variable* is visible (hence accessible) throughout the program, unless [shadowed](https://en.wikipedia.org/wiki/Variable_shadowing). The set of all global variables is known as the *global environment* or *global state.* In compiled languages, global variables are generally [static variables](https://en.wikipedia.org/wiki/Static_variable), whose [*extent*](https://en.wikipedia.org/wiki/Variable_(programming)#Scope_and_extent) (lifetime) is the entire runtime of the program. They are generally dynamically allocated when declared, since they are not known ahead of time but, once we will read our file, they will be reallocated with a specific value. Here is shown our *global variables list*.

**EXAMPLE 2.9**

*Global Variables List:*

Propulsion propulsion\_system = Propulsion.***CONSTANT\_TRUST***; // propulsion

regime type

**double** rho0; // air density

**double** surf; // wing area

**double** mass; // total mass

**double** cbar; // mean aerodynamic chord

**double** bbar; // wingspan

**double** u0; // speed of the aircraft

**double** q0; // dynamic pressure

**double** m0; // Mach number

**double** gamma0; // ramp angle

**double** theta0\_rad = Math.*toRadians*(gamma0); // Euler angle [rad] (assuming gamma0 = theta0)

**double** iXX; // lateral-directional moment of inertia (IXX)

**double** iYY; // longitudinal moment of inertia (IYY)

**double** iZZ; // lateral-directional moment of inertia (IZZ)

**double** iXZ; // lateral-directional product of inertia (IXZ)

**double** cd0; // drag coefficient at null incidence (Cdº) of the aircraft

Figure 2.9 – “Global Variables” list (part 1 of 3)

**double** cdAlpha0; // linear drag gradient (CdAlphaº) of the aircraft

**double** cdM0; // drag coefficient with respect to Mach (ClMº) of the aircraft

**double** cl0; // lift coefficient at null incidence (Clº) of the aircraft

**double** clAlpha0; // linear lift gradient (ClAlphaº) of the aircraft

**double** clAlpha\_dot0; // linear lift gradient time derivative (ClAlpha\_dotº) of the aircraft

**double** clQ0; // lift coefficient with respect to q (ClQº) of the aircraft

**double** clM0; // lift coefficient with respect to Mach (ClMº) of the aircraft

**double** clDelta\_T; // lift coefficient with respect to delta\_T (ClDelta\_Tº) of the aircraft

**double** clDelta\_E; // lift coefficient with respect to delta\_E (ClDelta\_Eº) of the aircraft

**double** cMAlpha0; // pitching moment coefficient with respect to Alpha (CmAlphaº) of the aircraft

**double** cMAlpha\_dot0; // pitching moment coefficient time derivative (CmAlpha\_dotº) of the aircraft

**double** cM\_m0; // pitching moment coefficient with respect to Mach number

**double** cMq; // pitching moment coefficient with respect to q

**double** cMDelta\_T; // pitching moment coefficient with respect to delta\_T (CMDelta\_Tº) of the aircraft

**double** cMDelta\_E; // pitching moment coefficient with respect to delta\_E (CMDelta\_Eº) of the aircraft

**double** cTfix; // thrust coefficient at a fixed point ( U0 = u , delta\_T = 1 )

**double** kv; // scale factor of the effect on the propulsion due to the speed

**double** cyBeta; // lateral force coefficient with respect to beta (CyBeta) of the aircraft

**double** cyP; // lateral force coefficient with respect to p (CyP) of the aircraft

**double** cyR; // lateral force coefficient with respect to r (CyR) of the aircraft

**double** cyDelta\_A; // lateral force coefficient with respect to delta\_A (CyDelta\_A) of the aircraft

**double** cyDelta\_R; // lateral force coefficient with respect to delta\_R (CyDelta\_R) of the aircraft

**double** cLBeta; // rolling moment coefficient with respect to beta (CLBeta) of the aircraft

**double** cLP; // rolling moment coefficient with respect to a p (CLP) of the aircraft

**double** cLR; // rolling moment coefficient with respect to a r (CLR) of the aircraft

**double** cLDelta\_A; // rolling moment coefficient with respect to a delta\_A (CLDelta\_A) of the aircraft

**double** cLDelta\_R; // rolling moment coefficient with respect to a delta\_R (CLDelta\_R) of the aircraft

**double** cNBeta; // yawing moment coefficient with respect to a beta (CNBeta) of the aircraft

**double** cNP; // yawing moment coefficient with respect to p (CNP) of the aircraft

**double** cNR; // yawing moment coefficient with respect to r (CNR) of the aircraft

**double** cNDelta\_A; // yawing moment coefficient with respect to delta\_A (CNDelta\_A) of the aircraft

**double** cNDelta\_R; // yawing moment coefficient with respect to delta\_R (CNDelta\_R) of the aircraft

**double** x\_u\_CT; // dimensional derivative of force component X with respect to "u" for Constant Thrust

**double** x\_u\_CP; // dimensional derivative of force component X with respect to "u" for Constant Power

Figure 2.10 - “global variables” list (part 2 of 3)

**double** x\_w; // dimensional derivative of force component X with respect to "w"

**double** x\_w\_dot; // dimensional derivative of force component X with respect to "w\_dot"

**double** x\_q; // dimensional derivative of force component X with respect to "q"

**double** z\_u; // dimensional derivative of force component Z with respect to "u"

**double** z\_w; // dimensional derivative of force component Z with respect to "w"

**double** z\_w\_dot; // dimensional derivative of force component Z with respect to "w\_dot"

**double** z\_q; // dimensional derivative of force component Z with respect to "q"

**double** m\_u; // dimensional derivative of pitching moment M with respect to "u"

**double** m\_w; // dimensional derivative of pitching moment M with respect to "w"

**double** m\_w\_dot; // dimensional derivative of pitching moment M with respect to "w\_dot"

**double** m\_q; // dimensional derivative of pitching moment M with respect to "q"

**double** x\_delta\_T\_CT; // dimensional control derivative of force component X with respect to "delta\_T" for Constant Thrust

**double** x\_delta\_T\_CP; // dimensional control derivative of force component X with respect to "delta\_T" for Constant Power

**double** x\_delta\_T\_CMF; // dimensional control derivative of force component X with respect to "delta\_T" for Constant Mass Flow

**double** x\_delta\_T\_RJ; // dimensional control derivative of force component X with respect to "delta\_T" for RamJet

**double** x\_delta\_E; // dimensional control derivative of force component X with respect to "delta\_E"

**double** z\_delta\_T; // dimensional control derivative of force component Z with respect to "delta\_T"

**double** z\_delta\_E; // dimensional control derivative of force component Z with respect to "delta\_E"

**double** m\_delta\_T; // dimensional control derivative of pitching moment M with respect to "delta\_T"

**double** m\_delta\_E; // dimensional control derivative of pitching moment M with respect to "delta\_E"

**double** y\_beta; // dimensional derivative of force component Y with respect to "beta"

**double** y\_p ; // dimensional derivative of force component Y with respect to "p"

**double** y\_r ; // dimensional derivative of force component Y with respect to "r"

**double** l\_beta; // dimensional derivative of rolling moment L with respect to "beta"

**double** l\_p ; // dimensional derivative of rolling moment L with respect to "p"

**double** l\_r ; // dimensional derivative of rolling moment L with respect to "r"

**double** n\_beta; // dimensional derivative of yawing moment N with respect to "beta"

**double** n\_p ; // dimensional derivative of yawing moment N with respect to "p"

**double** n\_r ; // dimensional derivative of yawing moment N with respect to "r"

**double** y\_delta\_A; // dimensional control derivative of force component Y with respect to "delta\_A"

**double** y\_delta\_R; // dimensional control derivative of force component Y with respect to "delta\_R"

**double** l\_delta\_A; // dimensional control derivative of rolling moment L with respect to "delta\_A"

**double** l\_delta\_R; // dimensional control derivative of rolling moment L with respect to "delta\_R"

**double** n\_delta\_A; // dimensional control derivative of yawing moment N with respect to "delta\_A"

**double** n\_delta\_R; // dimensional control derivative of yawing moment N with respect to "delta\_R"

**double** [][] aLon = **new** **double** [4][4]; // longitudinal coefficients [A\_Lon] matrix

**double** [][] bLon = **new** **double** [4][2]; // longitudinal control coefficients [B\_Lon] matrix

**double** [][] aLD = **new** **double** [4][4]; // lateral-directional coefficients [A\_LD] matrix

**double** [][] bLD = **new** **double** [4][2]; // lateral-directional control coefficients [B\_LD] matrix

**double**[][] lonEigenvaluesMatrix = **new** **double** [4][2]; // longitudinal eigenvalues matrix

**double**[][] ldEigenvaluesMatrix = **new** **double** [4][2]; // lateral-directional eigenvalues matrix

RealVector eigLonVec1; // longitudinal 1st eigenvector

RealVector eigLonVec2; // longitudinal 2nd eigenvector

RealVector eigLonVec3; // longitudinal 3rd eigenvector

RealVector eigLonVec4; // longitudinal 4th eigenvector

RealVector eigLDVec1; // lateral-directional 1st eigenvector

RealVector eigLDVec2; // lateral-directional 2nd eigenvector

RealVector eigLDVec3; // lateral-directional 3rd eigenvector

RealVector eigLDVec4; // lateral-directional 4th eigenvector

**double** zeta\_SP; // Short Period mode damping coefficient

**double** zeta\_PH; // Phugoid mode damping coefficient

**double** omega\_n\_SP; // Short Period mode natural frequency

**double** omega\_n\_PH; // Phugoid mode natural frequency

**double** period\_SP; // Short Period mode period

**double** period\_PH; // Phugoid mode period

**double** t\_half\_SP; // Short Period mode halving time

**double** t\_half\_PH; // Phugoid mode halving time

**double** N\_half\_SP; // Short Period mode number of cycles to halving time

**double** N\_half\_PH; // Phugoid mode number of cycles to halving time

**double** zeta\_DR; // Dutch-Roll mode damping coefficient

**double** omega\_n\_DR; // Dutch-Roll mode natural frequency

**double** period\_DR; // Dutch-Roll mode period

**double** t\_half\_DR; // Dutch-Roll mode halving time

**double** N\_half\_DR; // Dutch-Roll mode number of cycles to halving time

Figure 2.11 - “Global Variables” list (part 3 of 3)

### INPUT MANAGEMENT CLASSES

The two *procedures* named *readDataFromExcelFile* and *cellToString* are designed for extracting and reinterpreting any information contained in specific cells of an *excel* file as a *string* value. This file has to follow a particular model or reading procedure will not succeeds. In fact, our reading *procedure* is designed to read only the 2nd column of the current sheet (sheet number can be selected within the *main method*) and to stop once reached the 47th line (as much lines as the data are).

**EXAMPLE 2.10**

*A Correct Excel file output:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **Variable Name** | **Variable Value** | **Unit Label** | **Description** |
| **2** | propulsion\_system | CONSTANT\_TRUST | NONE | propulsion regime type |
| **3** | rho0 | 1,225 | kg\*m3 | air density |
| **4** | surf | 510,97 | m^2 | wing area |
| **5** | mass | 255753 | kg | total mass |
| **6** | cbar | 8,32 | m | mean aerodynamic chord |
| **7** | bbar | 59,64 | m | wingspan |
| **8** | u0 | 85,075 | m\*s | speed of the aircraft |
| **9** | m0 | 0,25 | NONE | Mach number |
| **10** | gamma0 | 0 | deg | ramp angle |
| **11** | theta0\_rad | 0 | rad | Euler angle [rad] (assuming gamma0 = theta0) |
| ***[…]*** | *[…]* | *[…]* | *[…]* | *[…]* |
| **46** | cNDelta\_A | 0,0064 | rad^(-1) | yawing moment coefficient with respect to delta\_A (CNDelta\_A) of the aircraft |
| **47** | cNDelta\_R | -0,109 | rad^(-1) | yawing moment coefficient with respect to delta\_R (CNDelta\_R) of the aircraft |
| **48** | *(empty)* | *(empty)* | *(empty)* | *(empty)* |
| **49** | *(empty)* | *(empty)* | *(empty)* | *(empty)* |

Figure 2.12 – A correct "excel file" output

The last two rows are intentionally left *empty*.

**EXAMPLE 2.11**

*readDataFromExcelFile*:

Figure 2.13 - Reading from excel file (part 1 of 3)

**public** **void** readDataFromExcelFile(File excelFile, **int** sheetNum) {

// Formats numbers up to 4 decimal places

DecimalFormat df = **new** DecimalFormat("#,###,##0.0000");

**try** {

System.***out***.println("Input file: " + excelFile.getAbsolutePath());

FileInputStream fis = **new** FileInputStream(excelFile);

Workbook wb = WorkbookFactory.*create*(fis);

Sheet ws = wb.getSheetAt(sheetNum);

**int** rowNum = ws.getLastRowNum() + 1;

System.***out***.println("Numero di righe: " + rowNum);

**if**(sheetNum == 0){

System.***out***.println("------------------------------------\n");

System.***out***.println("\n BOEING 747 /// Flight Condition (2)”);

System.***out***.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

System.***out***.println("DATA LIST: \n");

}

**else** **if** (sheetNum == 1){

System.***out***.println("------------------------------------\n");

System.***out***.println("\n BOEING 747 /// Flight Condition (5)”);

System.***out***.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

System.***out***.println("DATA LIST: \n");

}

**for** (**int** i = 0 ; i < rowNum ; i++) {

Row row = ws.getRow(i);

**int** colNum = ws.getRow(0).getLastCellNum();

**for** (**int** j = 0 ; j < colNum-2 ; j++) {

Cell cell = row.getCell(j);

String value = *cellToString*(cell);

**switch** (sheetNum){

///////////// 1st sheet /////////////

**case** 0:

**if** ((i == 1) && (j == 1)) {

This *method* extracts the rows number, using it as final value in the *for* cicle. During this cicle, for each row, the program reads the ith row and applies a new *for* cicle to read the jth column value. Then happens a *switch-case* cicle relative to sheet number and, right after, another *switch-case* cicle relative to the cell index. At this point a *sub-routine* saves the values in a *global variable* and prints it. Then the process is repeated for the 2nd sheet. If there’s any need to read a file with more than two sheets, the code can be easily modified. Moreover, code could be even more simplified by using the same routine for 1st and 2nd sheet, but we preferred to keep them separated, in case that the second sheet present a different number or order of elements.

propulsion\_system = Propulsion.*valueOf*(value);

**switch** (propulsion\_system){

**case** ***CONSTANT\_TRUST***:

System.***out***.println(" PROPULSION SYSTEM:

CONSTANT TRUST \n");

**break**;

[…]

**default**:

System.***out***.println(" PROPULSION SYSTEM:

CONSTANT TRUST \n");

**break**;

}

}

**if** ((i == 2) && (j == 1)) {

rho0 = Double.*parseDouble*(value);

System.***out***.println(" rho0 = " + rho0);

}

**if** ((i == 5) && (j == 1)) {

cbar = Double.*parseDouble*(value);

System.***out***.println(" cbar = " + cbar);

}

[…]

**if** ((i == 46) && (j == 1)) {

cNDelta\_R = Double.*parseDouble*(value);

System.***out***.println(" cNDelta\_R = " + cNDelta\_R);

}

**break**;

Figure 2.14 - Reading from excel file (part 2 of 3)

///////////// 2nd sheet /////////////

Figure 2.15 - Reading from excel file (part 3 of 3)

**case** 0:

**if** ((i == 1) && (j == 1)) {

[…]

**break**;

}

} } }

**catch**(Exception ioe) {

ioe.printStackTrace();

} }

The second procedure, *cellToString*, will not be explained in details to be lighter on the discussion but it is important to know that it provides a method for converting almost any kind of variables types into a *String* type.

### CONSTRUCTOR METHOD

*Constructor* in *Java* is a special type of *method* that is used to initialize the *object*.

**public** FlightDynamicsManager() {

}

Figure 2.16 -"Constructor" method

*Java constructor* is invoked at the time of *object* creation. It constructs the values i.e. provides data for the object that is why it is known as *constructor*.

### CALCULATOR METHOD

The *CalculateAll* method is definitely the biggest one. It has the task to recall all the other methods contained in the program and to save their results in the *global variables*, then it has to print them on screen to visualize the efficiency of the execution. Show in detail the entire *CalculateAll* structure would be unproductive, so is preferable to explain its single steps and extract some examples from each one.

*“CalculateAll”* **main structure** (for each *Longitudinal* and *Lateral-Directional dynamics*):

* Stability and Control Derivatives Calculation;
* Prints out the Stability and Control Derivatives List;
* Generates and Prints out the **A** and **B** matrices;
* Generates and Prints out the Eigenvalues matrix of **A** matrix;
* Generates and Prints out the Eigenvectors of **A** matrix;
* Calculates and Prints out the characteristics for *open-loop* modes;

We will now extract some examples for each category of tasks.

**EXAMPLE 2.12**

*Stability and Control Derivatives Calculation (X\_u\_CT)*

x\_u\_CT = StabilityDerivativesCalc.*calcX\_u\_CT*(rho0, surf, mass, u0, q0, cd0, m0, cdM0);

Figure 2.17 -"X\_u\_CT" derivative calculation

**EXAMPLE 2.13**

*Stability and Control Derivatives List (X\_u\_CT)*

// Formats numbers up to 4 decimal places

DecimalFormat df = **new** DecimalFormat("#,###,##0.0000");

System.***out***.println("LONGITUDINAL STABILITY DERIVATIVES: \n");

System.***out***.println(" Xªu\_CT = " + df.format(x\_u\_CT));

Figure 2.18 - "X\_u\_CT" derivative printing out

**EXAMPLE 2.14**

**A** *and* **B** *matrices (***A**LON *matrix)*

Figure 2.19 - "[A\_Lon]" matrix calculation and printing out

aLon = StabilityDerivativesCalc.*build\_A\_Lon\_matrix* (propulsion\_system, rho0, surf,

mass, cbar, u0, q0, cd0, m0, cdM0, cl0, cdAlpha0, gamma0, theta0\_rad,

clAlpha0, clAlpha\_dot0, cMAlpha0, cMAlpha\_dot0, clQ0, iYY, cM\_m0, cMq);

System.***out***.println(df.format(aLon[0][0])+"\t\t"+df.format(aLon[0][1])+"\t\t"+

df.format(aLon[0][2])+"\t\t"+df.format(aLon[0][3])+"\n");

**[…]**

System.***out***.println(aLon[3][0]+"\t\t"+aLon[3][1]+"\t\t"+aLon[3][2]+"\t\t"+aLon [3][3]+"\n");

**EXAMPLE 2.15**

*Eigenvalues matrix of* **A** *matrix (***A**LON*)*

lonEigenvaluesMatrix = DynamicStabilityCalculator.*buildEigenValuesMatrix*(aLon);

System.***out***.println(" SHORT PERIOD: "+df.format(lonEigenvaluesMatrix[0][0])+" ± j"+df.format(lonEigenvaluesMatrix[0][1])+"\n");

System.***out***.println(" PHUGOID: "+df.format(lonEigenvaluesMatrix[2][0])+" ± j"+df.format(lonEigenvaluesMatrix[2][1])+"\n");

Figure 2.20 - Longitudinal eigenvalues generation and printing out

**EXAMPLE 2.15**

*Eigenvectors of* **A** *matrix (1st Eigenvector)*

Figure 2.21 - 1st longitudinal eigenvector generation and printing out

System.***out***.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

System.***out***.println("LONGITUDINAL EIGENVECTORS:\n");

eigLonVec1 = DynamicStabilityCalculator.*buildEigenVector*(aLon, 0);

System.***out***.println("EigenVector 1 = " + eigLonVec1);

**EXAMPLE 2.15**

*Open-Loop characteristics (Short Period)*

Figure 2.22 – “Short Period” open-loop characteristics

zeta\_SP = DynamicStabilityCalculator.*calcZeta*(lonEigenvaluesMatrix[0][0],

lonEigenvaluesMatrix[0][1]);

omega\_n\_SP = DynamicStabilityCalculator.*calcOmega\_n*(lonEigenvaluesMatrix[0][0],

lonEigenvaluesMatrix[0][1]);

period\_SP = DynamicStabilityCalculator.*calcT*(lonEigenvaluesMatrix[0][0],

lonEigenvaluesMatrix[0][1]);

t\_half\_SP = DynamicStabilityCalculator.*calct\_half*(lonEigenvaluesMatrix[0][0],

lonEigenvaluesMatrix[0][1]);

N\_half\_SP = DynamicStabilityCalculator.*calcN\_half*(lonEigenvaluesMatrix[0][0],

lonEigenvaluesMatrix[0][1]);

System.***out***.println("SHORT PERIOD MODE CHARACTERISTICS\n");

System.***out***.println("Zeta\_SP = "+df.format(zeta\_SP)+"\n");

System.***out***.println("Omega\_n\_SP = "+df.format(omega\_n\_SP)+"\n");

System.***out***.println("Period = "+df.format(period\_SP)+"\n");

System.***out***.println("Halving Time = "+df.format(t\_half\_SP)+"\n");

System.***out***.println("Number of cycles to Halving Time = "+df.format(N\_half\_SP)+"\n");

All customer examples cited well sums up all the statements that occur within the *method*. To better visualize how exactly does the *Calculator* works, we invite you to visualize the relative *APPENDIX*.

### MAIN METHOD

In the Java language, when you execute a class with the Java *interpreter*, the runtime system starts by calling the class's *main()* method. The *Java Main Method* then calls all the other methods required to run your application.

**EXAMPLE 2.15**

*Java Main Method*

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

FlightDynamicsManager theObj = **new** FlightDynamicsManager();

System.***out***.println("---------------------------------------------------");

System.***out***.println("Reading input data file (excel format)");

String inputFileName = "AIRCRAFT\_DATA.xlsx";

File excelFile = **new** File (inputFileName) ;

////// select the excel sheet you want to read \\\\\\

**int** sheetNumber = 0;

**if** (excelFile.exists()){

System.***out***.println("File " + inputFileName + " found.");

System.***out***.println("\n %%% start reading from file %%% ");

// Read all data from file

theObj.readDataFromExcelFile(excelFile, sheetNumber);

System.***out***.println("\n %%% end of reading from file %%%");

theObj.calculateAll();

}

**else** {

System.***out***.println("File " + inputFileName + " not found.");

}

}

Figure 2.23 - Java Main Method

First step is to initialize the *Constructor*, then start reading from file.

Inside of it, we can select the *sheet number* (condition of flight). After that, *Calculator* starts. The *Main method* has been built as shortest as possible.

# BOEING 747 – TEST

We are now ready to test the program on a complete aircraft model (*Boeing-747*) in two specific conditions of flight, making a comparison with actual results.

## CONDITIONS OF FLIGHT

In the next figure, we will report the characteristics of the *Boeing-747*, a large *airliner* with turbofan engines, in two different flight conditions. The data were taken from the ***NASA report***(Heffley & Jewell, December 1972). Numbering of the conditions considered (‘2’ and ‘5’) corresponds to the same used in the original report. All the configurations appear in a clean configuration (flap not deflected and motors in function) except for condition ‘2’, which presents the aircraft in *powered approach* configuration with a 20 deg flaps deflection.

Figure 3.1 – Flight Conditions (part 1 of 2) (Heffley & Jewell, December 1972)

|  |  |  |  |
| --- | --- | --- | --- |
| Variable | Unit | Condition 2 | Condition 5 |
| *h0* | m | 0 | 6096 |
| *ρ0* | kg/m3 | 1,225 | 0,653 |
| *a0* | m/s | 340,29 | 158,02 |
| *S* | m2 | 510,97 | 510,97 |
| *m* | kg | 255753 | 288676 |
|  | m | 8,32 | 8,32 |
|  | m | 59,64 | 59,64 |
| *U0* | m/s | 85,07 | 158,02 |
| *q0* | kg/m∙s2 | 1098,42 | 8148,65 |
| *M0* | [adim.] | 0,25 | 0,50 |
| *Γ0* | deg | 0 | 0 |
| *Ixx* | kg⋅m2 | 1,94⋅107 | 2,49⋅107 |
| *Iyy* | kg⋅m2 | 4,38⋅107 | 4,49⋅107 |
| *Izz* | kg⋅m2 | 6,14⋅107 | 6,71⋅107 |
| *Ixz* | kg⋅m2 | -3,02⋅106 | -3,74⋅106 |
| *SM = (Xn – Xcg)/* | [adim.] | 0,22 | 0,22 |
| *αB* | deg | 5,70 | 6,80 |
|  | [adim.] | 0,102 | 0,04 |

|  |  |  |  |
| --- | --- | --- | --- |
| Variable | Unit | Condition 2 | Condition 5 |
|  | rad-1 | 0,66 | 0,37 |
|  | [adim.] | 0 | 0 |
|  | [adim.] | 1,108 | 0,68 |
|  | rad-1 | 5,70 | 4,67 |
|  | rad-1 | 6,70 | 6,53 |
|  |  | 0 | -0,09 |
|  | rad-1 | 5,40 | 5,13 |
|  | [adim.] | 0 | 0 |
|  | rad-1 | 0,338 | 0,356 |
|  | rad-1 | -1,26 | -1,15 |
|  | rad-1 | -3,20 | -3,35 |
|  | [adim.] | 0 | 0,12 |
|  | rad-1 | -20,80 | -20,7 |
|  | rad-1 | 0 | 0 |
|  | rad-1 | -1,34 | -1,43 |
|  | [adim.] | 0 | 0 |
|  | [adim.] | 0 | 0 |
|  | [adim.] | -0,96 | -0,9 |
|  | [adim.] | 0 | 0 |
|  | [adim.] | 0 | 0 |
|  | [adim.] | 0 | 0 |
|  | [adim.] | 0,175 | 0,1448 |
|  | [adim.] | -0,22 | -0,193 |
|  | [adim.] | -0,45 | -0,323 |
|  | [adim.] | 0,10 | 0,212 |
|  | [adim.] | 0,046 | 0,0129 |
|  | [adim.] | 0,007 | 0,0039 |
|  | [adim.] | 0,15 | 0,147 |
|  | [adim.] | -0,12 | -0,0687 |
|  | [adim.] | -0,30 | -0,278 |
|  | [adim.] | 0,0064 | 0,0015 |
|  | [adim.] | -0,109 | -0,1081 |

Figure 3.2 – Flight Conditions (part 2 of 2) (Heffley & Jewell, December 1972)

## OUTPUT

We will now see a typical output for flight conditions ‘2’ and ‘5’.

Figure 3.3 - Conditions '2' and '5' Output (part 1 of 5)

|  |  |
| --- | --- |
| **Condition 2** | **Condition 5** |
| -----------------------------------------Reading input data file (excel format)  File AIRCRAFT\_DATA.xlsx found.  %%% start reading from file %%%  Input file: C:\workspace\newproj\AIRCRAFT\_DATA.xlsx  rows number: 50  -----------------------------------------  BOEING 747 /// Flight Condition (2)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  DATA LIST:  PROPULSION SYSTEM: CONSTANT TRUST  rho0 = 1.225  surf = 510.9667  mass = 255753  **[…]**  cNR = -0.3  cNDelta\_A = 0.0064  cNDelta\_R = -0.109  %%% end of reading from file %%%  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  LONGITUDINAL STABILITY DERIVATIVES:  Xªu\_CT = -0,0212  Xªu\_CP = -0,0319  Xªw = 0,0466  Xªw\_dot = 0,0000  Xªq = 0,0000  Zªu = -0,2307  Zªw = -0,6040  Zªw\_dot = -0,0341  Zªq = -2,3389  Mªu = 0,0000  Mªw = -0,0064  Mªw\_dot = -0,0008  Mªq = -0,4378  LONGITUDINAL CONTROL DERIVATIVES:  Xªdelta\_T\_CT = -0,0000  Xªdelta\_T\_CP = -0,0000  **[…]**  Zªdelta\_E = -2,9935  Mªdelta\_T = 0,0000  Mªdelta\_E = -0,5767  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  MATRIX [A\_LON]:  -0,0212 0,0466 0,0000 -9,8100  -0,2231 -0,5841 80,0055 -0,0000  0,0002 -0,0059 -0,5011 0,0000  0.0 0.0 1.0 0.0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  MATRIX [B\_LON]:  -0,0000 0,0000  0,0000 -2,8948  0,000 -0,5744   1. 0.0   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  LONGITUDINAL EIGENVALUES  SHORT PERIOD: -0,5515 ± j0,6879  PHUGOID: -0,0018 ± j0,1340  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  LONGITUDINAL EIGENVECTORS:  EigenVector 1 = {0,0062; -3,4144; -0,0940; 0,0468}  EigenVector 2 = {0,8950; 10,7820; -0,0224; 0,0991}  EigenVector 3 = {-41,9631; 6,24644; -0,0622; 0,7458}  EigenVector 4 = {46,3490; -5,5770; 0,0991; 0,4545}  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  SHORT PERIOD MODE CHARACTERISTICS  Zeta\_SP = 0,6255  Omega\_n\_SP = 0,8816  Period = 9,1341  Halving Time = 1,2569  Number of cycles to Halving Time = 0,1376  Figure 3.4 - Conditions '2' and '5' Output (part 2 of 5)  PHUGOID MODE CHARACTERISTICS  Zeta\_PH = 0,0132  Omega\_n\_PH = 0,1340  Period = 46,905  Halving Time = 391,14  Number of cycles to Halving Time = 8,3390 | -----------------------------------------  Reading input data file (excel format)  File AIRCRAFT\_DATA.xlsx found.  %%% start reading from file %%%  Input file: C:\workspace\newproj\AIRCRAFT\_DATA.xlsx  rows number: 50  -----------------------------------------  BOEING 747 /// Flight Condition (5)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  DATA LIST:  PROPULSION SYSTEM: CONSTANT TRUST  rho0 = 0.6527  surf = 510.9667  mass = 288676  **[…]**  cNR = -0.278  cNDelta\_A = 0.0015  cNDelta\_R = -0.1081  %%% end of reading from file %%%  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  LONGITUDINAL STABILITY DERIVATIVES:  Xªu\_CT = -0,0073  Xªu\_CP = -0,0110  Xªw = 0,0283  Xªw\_dot = 0,0000  Xªq = 0,0000  Zªu = -0,1240  Zªw = -0,4299  Zªw\_dot = -0,0157  Zªq = -1,9482  Mªu = 0,0000  Mªw = -0,0056  Mªw\_dot = -0,0004  Mªq = -0,4208  LONGITUDINAL CONTROL DERIVATIVES:  Xªdelta\_T\_CT = -0,0000  Xªdelta\_T\_CP = -0,0000  **[…]**  Zªdelta\_E = -5,1347  Mªdelta\_T = 0,0000  Mªdelta\_E = -1,1040  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  MATRIX [A\_LON]:  -0,0073 0,0283 0,0000 -9,8100  -0,1195 -0,4233 153,65 -0,0000  0,0003 -0,0054 -0,4870 0,0000   1. 0.0 1.0 0.0   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  MATRIX [B\_LON]:  -0,0000 0,0000  0,0000 -5,0554  0,0000 -1,1018   1. 0.0   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  LONGITUDINAL EIGENVALUES  SHORT PERIOD: -0,4567 ± j0,9119  PHUGOID: -0,0021 ± j0,0866  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  LONGITUDINAL EIGENVECTORS:  EigenVector 1 = {0,0137; -0,0158; 0,0080; -0,0034}  EigenVector 2 = {-0,0429; -1,3620; 0,0001; -0,0071}  EigenVector 3 = {-89,2840; 1,0188; -0,0669; 0,5293}  EigenVector 4 = {54,2225; 0,5551; 0,0442; 0,7607}  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  SHORT PERIOD MODE CHARACTERISTICS  Zeta\_SP = 0,4478  Omega\_n\_SP = 1,0199  Period = 6,8901  Halving Time = 1,5177  Number of cycles to Halving Time = 0,2203  PHUGOID MODE CHARACTERISTICS  Zeta\_PH = 0,0238  Omega\_n\_PH = 0,0866  Period = 72,555  Halving Time = 336,90  Number of cycles to Halving Time = 4,6435 |

Figure 3.5 - Conditions '2' and '5' Output (part 3 of 5)

As we can see from the *free response* characteristics and from the image below, it is easy to visualize the main differences between longitudinal response modes.

In particular, *short period mode* is characterized by a stronger damping coefficient and a higher natural frequency, which lead to a shorter period than *phugoid mode*.

It is also interesting to compare their values from condition ‘2’ to ‘5’, in dependence from altitude and Mach value.

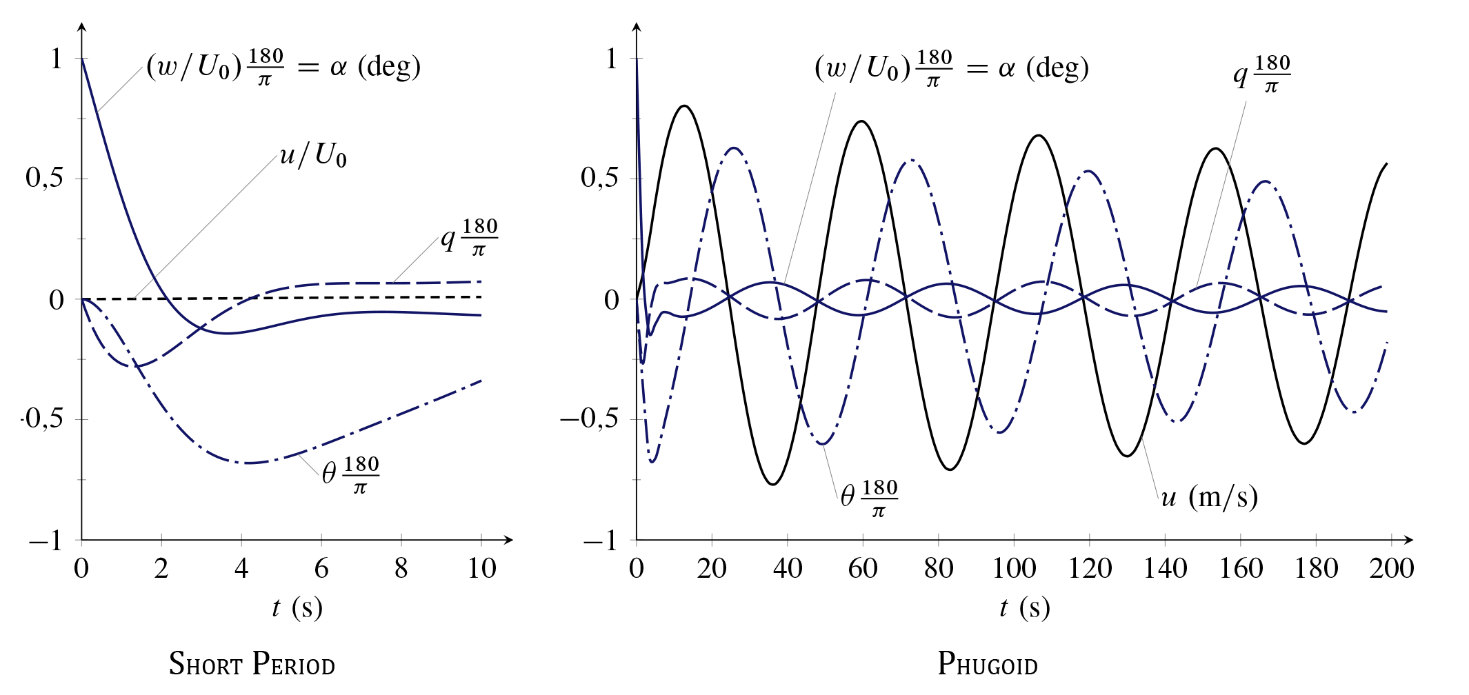


Figure 3.6 - Free responses of a Boeing-747 in the longitudinal disturbances of initial condition. The two responses were obtained exciting, respectively, only the Short Period mode (left) and the Phugoid mode (right) (De Marco & Coiro, 2015)

|  |  |
| --- | --- |
| **Condition 2** | **Condition 5** |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  LATERAL-DIRECTIONAL STABILITY DERIVATIVES:  Yªbeta = -8,5023  Yªp = 0,0000  Yªr = 0,0000  Lªbeta = -1,5399  Lªp = -1,0992  Lªr = 0,2467  Nªbeta = 0,3299  Nªp = -0,0933  Nªr = -0,2313  LATERAL-DIRECTIONAL CONTROL DERIVATIVES:  Yªdelta\_A = 0,0000  Yªdelta\_R = 1,5499  Lªdelta\_A = 0,3212  Lªdelta\_R = 0,0488  Nªdelta\_A = 0,0141  Nªdelta\_R = -0,2398  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  MATRIX [A\_LD]:  -0,2453 0,4089 -0,0395 0,0000  -1,0000 -0,0999 0,0000 0,1153  0,2850 -1,6037 -1,0930 0,0000   1. 0.0 1.0 0.0   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  MATRIX [B\_LD]:  -0,0017 -0,2440  0,0000 0,0182  0,3215 0,0868   1. 0.0   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  LATERAL-DIRECTIONAL EIGENVALUES  ROLL: -1,2306  DUTCH-ROLL: -0,0806 ± j0,7433  SPIRAL: -0,0464  Figure 3.7 - Conditions '2' and '5' Output (part 4 of 5)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  LATERAL-DIRECTIONAL EIGENVECTORS:  EigenVector 1 = {-0,1100; -0,6107; 0,7215; -0,4934}  EigenVector 2 = {0,3499; -0,0873; -0,2928; -0,9171}  EigenVector 3 = {0,0052; 0,1090; 1,2600; -1,0238}  EigenVector 4 = {-0,3019; -0,1348; 0,1244; -2,6813}  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  DUTCH-ROLL MODE CHARACTERISTICS  Zeta\_DR = 0,1078  Omega\_n\_DR = 0,7477  Period = 8,4529  Halving Time = 8,5975  Number of cycles to Halving Time = 1,0171 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  LATERAL-DIRECTIONAL STABILITY DERIVATIVES:  Yªbeta = -12,9810  Yªp = 0,0000  Yªr = 0,0000  Lªbeta = -1,9212  Lªp = -0,6068  Lªr = 0,3983  Nªbeta = 0,5439  Nªp = -0,0480  Nªr = -0,1941  LATERAL-DIRECTIONAL CONTROL DERIVATIVES:  Yªdelta\_A = 0,0000  Yªdelta\_R = 2,0885  Lªdelta\_A = 0,1284  Lªdelta\_R = 0,0388  Nªdelta\_A = 0,0056  Nªdelta\_R = -0,4000  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  MATRIX [A\_LD]:  -0,2182 0,6566 -0,0143 0,0000  -1,0000 -0,0822 0,0000 0,0621  0,4310 -2,0197 -0,6047 0,0000   1. 0.0 1.0 0.0   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  MATRIX [B\_LD]:  -0,0016 -0,4056  0,0000 0,0132  0,1287 0,0997   1. 0.0   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  LATERAL-DIRECTIONAL EIGENVALUES  ROLL: -0,7414  DUTCH-ROLL: -0,0729 ± j0,8562  SPIRAL: -0,0179  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  LATERAL-DIRECTIONAL EIGENVECTORS:  EigenVector 1 = {0,2651; 0,2718; -0,8138; 0,0830}  EigenVector 2 = {-0,1770; 0,3066; 0,0023; 0,9435}  EigenVector 3 = {0,0623; -0,0794; -1,3701; 1,8480}  EigenVector 4 = {-0,2514; -0,0751; 0,0738; -4,1280}  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  DUTCH-ROLL MODE CHARACTERISTICS  Zeta\_DR = 0,0848  Omega\_n\_DR = 0,8593  Period = 7,3387  Halving Time = 9,5143  Number of cycles to Halving Time = 1,2964 |
|  |  |

Figure 3.8 - Conditions '2' and '5' Output (part 5 of 5)

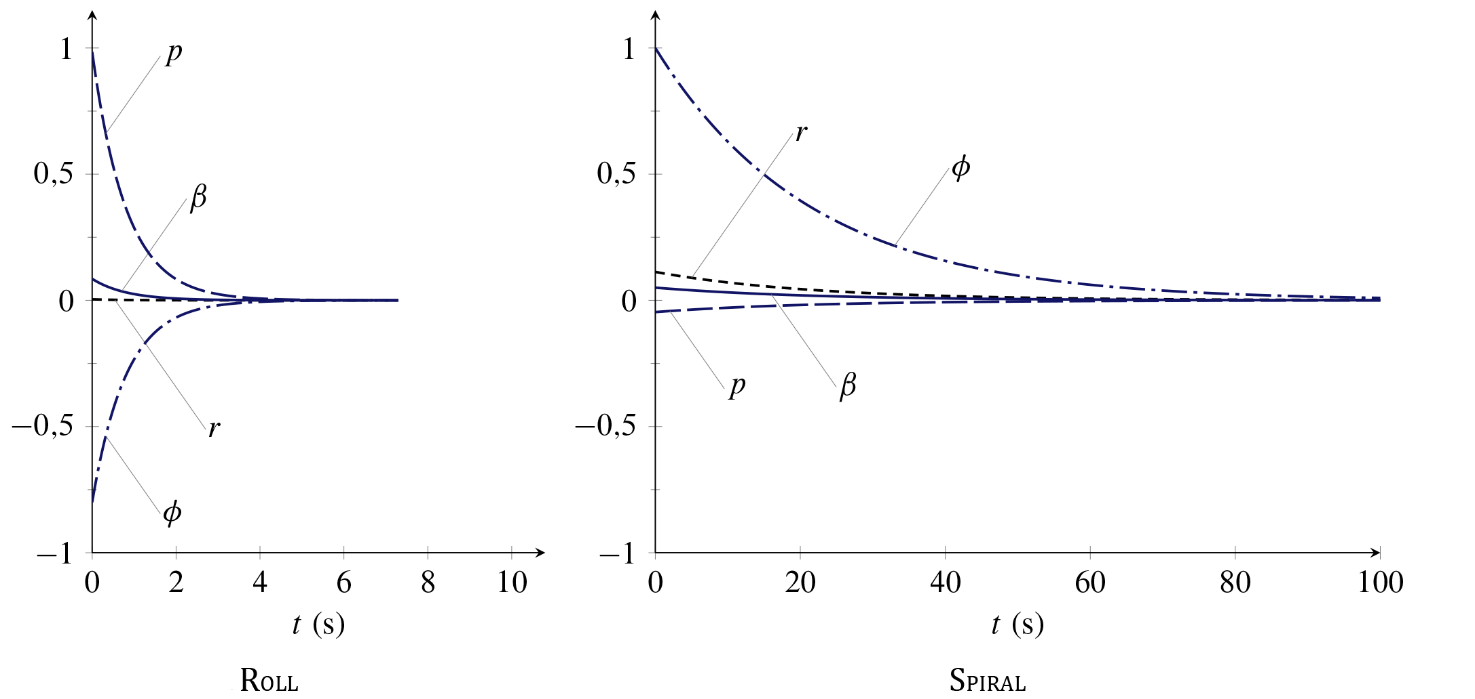


Figure 3.9 - Free responses of a Boeing-747 in the lateral-directional disturbances of initial condition. The two responses were obtained exciting, respectively, only the Roll mode (left) and the Spiral mode (right) (De Marco & Coiro, 2015)

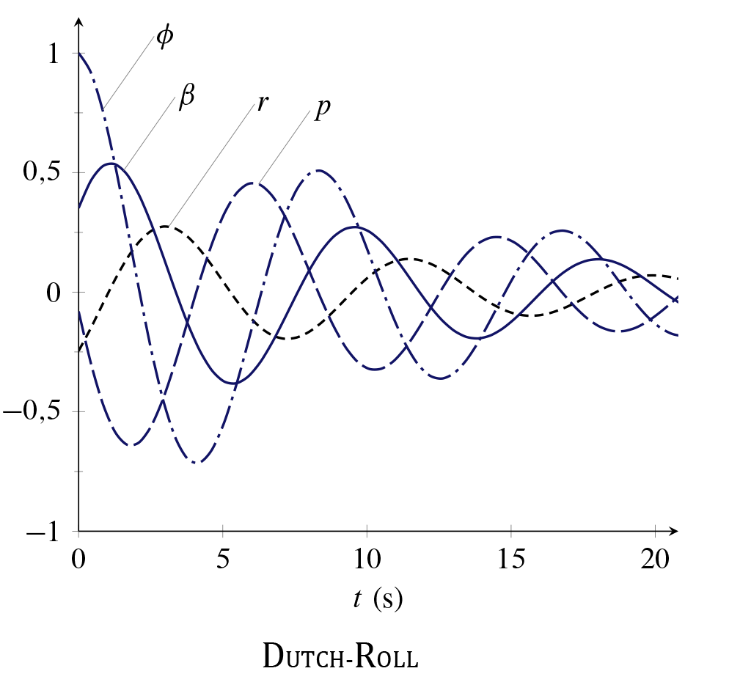
The free responses shown in Figures 3.6, 3.9 and 3.10 are obtained resolving the problem for an initial condition coincident, respectively, with the (left) eigenvector of free response modes.

Figure 3.10 - Free response of a Boeing 747 in lateral-directional perturbation, obtained exciting just the Dutch-Roll mode (De Marco & Coiro, 2015)FINAL REMARKS

## FINAL REMARKS

Stability analysis transcends our discussion, in fact our program simply extrapolates dynamics key values without analyzing them in a system response optics. Nonetheless, it is immediate to note how all the eigenvectors are characterized by a negative real part, which supports the stability condition for small perturbations hypothesis. It is also clear how the values between condition '2' and '5' are comparable in order of magnitude. In addition, the differences between damping coefficient and natural frequency in longitudinal response modes explains why we easily distinguish the short period and phugoid modes according to their dynamic characteristics. Moreover, plotting damping coefficient in terms of natural frequency is of theoretical interest to determine a zone of response optimization, according to the Thumb Print criterion (De Marco & Coiro, 2015).

Finally, we recall that all the results achieved are in exact form, obtained directly from the eigenvalues. There are also approximation formulas obtained instead manipulating the stability derivatives, which are more or less valid depending on the conditions of motion. Some cases, in particular the dutch-roll mode, are difficult to approximate, since they induce variations on all four state variables. That is why, for the purpose of a more appropriate and applicable analysis, we do not use these formulas to derive the characteristics of response modes but we prefer, instead, to obtain them from a more complex and accurate study on eigenvalues.

# BIBLIOGRAPHY

Apache Commons. (s.d.). *Commons Math: The Apache Commons Mathematics Library*, 3.5. Tratto da https://commons.apache.org/proper/commons-math/

De Marco, A., & Coiro, D. P. (2015, Maggio). *Elementi di Dinamica e Simulazione di Volo - Quaderno 16 (Piccole Perturbazioni del Moto di un Velivolo).* Tratto da http://wpage.unina.it/agodemar/DSV-DQV/#materiale-didattico: http://wpage.unina.it/agodemar/DSV-DQV/DSV-DQV\_Quaderno\_16.pdf

Heffley, R. K., & Jewell, W. F. (December 1972). *Aircraft Handling Qualities Data.* NASA-CR-2144.

Sharan, K. (2014). *Beginning Java 8 Fundamentals: Language Syntax, Arrays, Data Types, Objects, and Regular Expressions.* Apress.

# NOTES

1. Real roots λ HEIGHT and λ RANGE do not appear in our characteristic polynomial because matrix **A**LON order has been reduced due to our simplified nominal condition. Their weight would have been negligible, anyway. [↑](#endnote-ref-1)