POLITECNICO DI MILANO

Facoltà di Ingegneria Industriale

Corso di Laurea in Ingegneria Aeronautica


# Multidisciplinary Multirate Co-simulations in Multibody Dynamics

Relatore: Prof. Pierangelo Masarati

Tesi di Laurea di:

Tommaso Solcia, matr. 706895

Anno Accademico 2008 – 2009

# Contents

# List of Figures

# List of Tables

**Abstract**

Simulations of mechatronic systems quite always imply interaction between numerous and multidisciplinary components. In addition, it is common that the involved components and subsystems show different time scales and frequencies.

An efficient way to build computer simulation models is to use a multidisciplinary environment, in which co-simulation setups are available, in order to simplify the definition of the computer model and improve the computational efficiency. Multirate methods can be used to further improve the efficiency.

Most multirate methods are designed to be applied to monoblock equation systems, and used in single disciplinary simulations. The application of these methods in co-simulations setups may be not straightforward.

The main concern of this thesis is to study properties and performances of multirate co-simulations, and to demonstrate it is possible to build a free software environment capable of integrating a Multibody simulator with a general purpose block diagram simulator, exploiting also the multirate layout. In particular, the co-simulation setup is formed by the multibody simulator MBDyn and the general purpose block diagram simulator Scicos.
The tool derived from this integration is finally applied to simulate the behavior of a controlled wind turbine and a helicopter, to show the computational costs and the effort needed in building the model for possible real cases.

A new multirate scheme is also presented, for which stability and accuracy results are given.

**Keywords**: Co-simulations, Multirate, Multibody, Block Diagram, Free Software.

**Sommario**

La simulazione dei sistemi meccatronici quasi sempre implica interazioni tra numerosi componenti, spesso studiati da discipline diverse. Inoltre, è comune che componenti e sottosistemi coinvolti mostrino diverse scale temporali.

Un metodo efficiente per costruire modelli computazionali è utilizzare ambienti multidisciplinari, in cui sia possibile eseguire co-simulazioni, in modo da semplificare la descrizione dei modelli e migliorare l'efficienza di calcolo. I metodi Multirate possono essere utilizzati per migliorare ulteriormente le prestazioni

Per la maggior parte, i metodi di tipo Multirate sono stati progettati per essere applicati a sistemi di equazioni monoblocco, risolte da singoli simulatori. L'applicazione di tali metodi alle co-simulazioni pu essere complicata.

Il primo obiettivo di questa tesi è studiare le proprietà e le prestazioni delle co-simulazioni Multirate, e dimostrare che è possibile costruire un ambiente di tipo free software capace di integrare un simulatore multicorpo con uno schema a blocchi generico, sfruttando anche la tecnica multirate.

In particolare, il setup presentato è formato dal simulatore multicorpo $MBDyn$ e dal simulatore basato su schemi a blocchi $Scicos$.
Lo strumento risultante da questa integrazione verrà infine utilizzato per simulare il comportamento di una turbina eolica controllata automaticamente e di un elicottero, per mostrare i costi computazionali e lo sforzo per costruire il modello di calcolo applicazioni reali.

Inoltre, un nuovo metodo multirate viene presentato, di cui sono date le propriet di stabilit e accuratezza.

**Parole chiave**: Co-simulazioni, multirate, multicorpo, schemi a blocchi, software libero.

# Chapter 1

# Introduction

The dynamical behavior of mechanical systems can be analyzed by means of multibody simulators. This kind of tool is developed by researchers since the 1960s, and during the decades lots of improvements have been reached also due to the fast growing of computational capabilities.

Nowadays, the simulation of mechanical systems very often implies a strict integration between different disciplines, such as mechanics, electronics, hydraulics and active controls [1].

This kind of integrated systems usually brings to different approaches to analyze the behavior of the interacting components: for example, Multibody and Finite Elements are supposed to be used for structural, kinematic and dynamic analyses of mechanical systems, while to define the characteristics of controllers, sensors and actuators, designers widely exploit general purpose block scheme simulators.

Using different approaches leads to some advantages, going from the ease in defining the model, to the efficiency of the built-in algorithms, to the fact that every specialist has good skills in using the software related to his own discipline, and everything includes all the benefits coming from many years of development and testing.

In the last years lots of efforts have been spent trying to extend multibody simulators with multidisciplinary tools, in order to keep the benefits coming from using multiple approach techniques.

An interesting example is represented by multidisciplinary co-simulation environments, which keep advantages from both using different softwares and building a single model to run a complete single analysis. By this approach, the multibody simulator and the block scheme coexist, thus allowing the de-

signer to build a model of each component employing one of the two ways.

The block scheme can also be used to simplify the description of some mechanical components, which are not important by themselves, and are only needed to define their main effects on the system.

In addition, different frequencies can be experienced by the subsystems that form the whole layout. In this case also, the co-simulation setup can be helpful, because the two simulators can be driven by different timesteps.

The drawback coming from such a layout, however, is that integration of different systems can be not straightforward.

## 1.1 Thesis Objectives

The main objective of this work is to study the co-simulations integrating multibody and block scheme simulators, both in single-rate and multirate layout. A theoretical study will be developed about the properties and performances of the numerical methods needed. Two softwares (a multibody and a block scheme simulator) will be selected, and an interface will be implemented to build the multidisciplinary environment. Also, to complete the advantages given by a co-simulation setup, the generated environment will support the multirate technique.

The extensions built will be applied on a real case, to check whether the benefits are interesting, both in terms of versatility and computational performances.

An additional concern of this thesis is to demonstrate that it is possible to build such an architecture exploiting free software only (see section 1.2 for an explanation).

The result will give the user the possibility of building models of mechatronic systems in a co-simulation environment, thus offering benefits in both simplifying the model construction and improving the efficiency of the simulations.

## 1.2 Free Software

In contrast to the proprietary software, the free software was born to give the users the opportunity to study, improve and distribute the software itself. The idea is even different from the open-source aims, which are limited to give the source code and have nothing to deal with the freedoms guaranteed by free

software distributions. "Free" is indeed to be intended as *freedom* (liberty, the condition of being free of restraints) and not as a matter of price.
For a software to be defined free it is necessary to guarantee four essential freedoms:

**Freedom 0** The freedom to run the program, for any purpose;

**Freedom 1** The freedom to study how the program works, and change it to make it do what you wish. Access to the source code is a precondition for this;

**Freedom 2** The freedom to redistribute copies so you can help your neighbor

**Freedom 3** The freedom to distribute copies of your modified versions to others. By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.

For sure, a free software implementation running on low cost hardware gives anybody both the opportunity to use advanced simulation techniques and the possibility to even explore the contents of the methods implemented and the computer techniques involved. This is, without any doubt, a great chance for the academic world, being much easier for the users to get involved in the developing of new techniques. Also, it is likely that small enterprises have an easier life if such a setting is available.

## 1.3 Thesis Structure

In chapter 1, a brief description of the topics touched by thesis is given.

Chapter 2 describes the multirate co-simulations and the possible applications in which they can be useful.

Chapter 3 gives the analytical definition of the numerical methods considered, mainly aimed to give a detailed description of the multirate schemes that could be used in the environment presented in chapter 4. Also, a new multirate scheme is presented. For all the most indicative cases, the stability and accuracy analysis are performed, both to show the properties of the methods in general and to give a comparison between them.

Chapter 4 presents the working environment built in this context. The description of the softwares used for the co-simulations is given, together with the definition of the intercommunicating process exploited.

After the description of the computational tool is given, in chapter 5 results of a possible application are given. A controlled wind turbine is modeled to show both the computational cost relative to a real application and to give an idea of the effort needed to build the numerical model.

Finally, chapter 7 summarizes the conclusions and results achieved by this work.

# Chapter 2

# Multirate Co-simulations

## 2.1 Introduction

Simulation of complex mechatronic systems in which mechanical components interact with sensors, actuators and controllers can be achieved exploiting different software tools, each simulating the dynamical behavior of one subsystem. Possible practical cases are automobiles (Anti-lock Braking Systems, Traction Controls, Electronic Stability Controls), robots, airplanes (Stability Augmentation Systems, Flutter Suppression Systems) and many others.

Many multibody system softwares can internally model multidisciplinary components (like hydraulic and electrical actuators, aerodynamic surfaces or control loops). It is often convenient, however, that the multibody simulator interact with an external software, such as a block diagram simulator. In this way many benefits arise: the model definition becomes easier, much more tools are available, algorithms are already optimized for controller simulations, and so on. Nevertheless, the integration of different systems can be not straightforward.

## 2.2 Multirate Systems

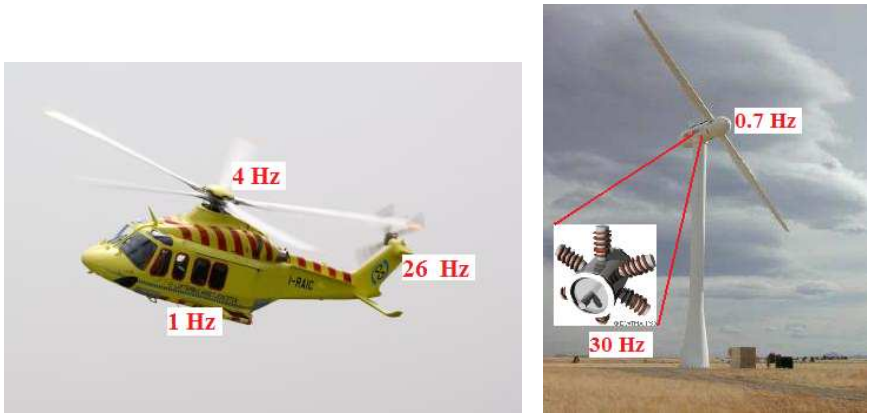
Complex systems, in addition to multidisciplinary components, may show different time scale subsystems. Every time such a situation occurs, it would be convenient to simulate the dynamic behavior integrating the state motion with different frequencies, in order to save computational efforts.

Multirate methods have been studied for many years (since the late 1970s), but their application in co-simulations is not yet developed, mainly because

this kind of methods have always been based on monolithic sets of equations integrated by a single process.

However, the argument is recently showing up in the multibody community, since it happens very often to face with mechatronic systems showing multirate behaviors. In fact, every time electronic components show up in the system, they usually exhibit a faster dynamics than mechanical components. A combination of multibody and block diagram simulators (a common setup in mechatronic industry), working in a multirate layout, has been presented in [2] and [3]. In [2], an in-house developed multibody simulator is coupled to a commercial block diagram simulator (Simulink) in a multirate layout. The setup is applied to the simulation of the dynamics of a kart. The multiphysics model is divided into two subsystems: a multibody model of the mechanical components of the vehicle, including the steering column, tyres and suspensions, and a thermodynamic model of a fourcylinder spark ignition engine, implemented in a Simulink block diagram. The Simulink model is then integrated with a time step of $10^{-4}s$, while the multibody model is integrated with a $10^{-2}s$ time step. In [3], the commercial multibody software SIMPACK is interfaced with the simulation environment Modelica/Dymola. The application is the dynamic simulation of a car with servo-hydraulic steering system. In this case, a multibody model in SIMPACK describes the mechanical components, while the steering system has been described based on the general purpose Modelica language.

Also, it is possible that mechanical subsystems have different frequencies. In a helicopter, for example, the tail rotor usually rotates 6/7 times faster than the main rotor, and this last is 3/4 times faster than the flight dynamic (figure 2.1(a) shows characteristic values). Systems where mechanical and electrical components are coupled together quite always show multirate behavior, as the electrical subsystems are always faster: figure 2.1(b) shows a wind turbine, as example: the gearbox ratio is about 40, making the electrical generator shaft much faster than the turbine rotor. The possibility of handling a multirate setup in all these situations leads to a series of benefits, such as saving of computational time, respecting real-time constraints in complex model and enhancing the productivity when numerous analyses have to be run (for example if different configurations have to be considered).




(a) Helicopters: different frequencies between the main and tail rotors and flight dynamics

(b) Wind turbines: electrical generators rotate much faster than turbine rotors

Figure 2.1: Mutlirate systems examples

## 2.3 Co-simulation Architecture

Simulation of multiphysic systems can be carried out splitting the main model into different subsystems, each of those is defined by means of different model, and maybe in different languages or softwares. This kind of layout has been defined as the optimal one to perform multidisciplinary simulations [4]. In co-simulation layouts, two or more simulators are exploited to integrate the motion of the states of the subsystems. For this architecture to work, the tools involved need to communicate during the execution of the simulation, in order to emulate the real physical interactions.

As already stated, even though multirate algorithms have been studied for many years, their application in co-simulations is still an open field of research, especially for the interaction between multibody and block scheme simulators.

The already mentioned wind turbine example (figure 2.1(b)) is also a test bench for co-simulation architectures. In figure 2.2 it is shown how the wind turbine can be split in different subsystems, each of those is modeled in a different environment: the mechanical and aerodynamic behavior of the turbine are simulated as a multibody system, while the controller and the electrical generator are described in a block diagram.

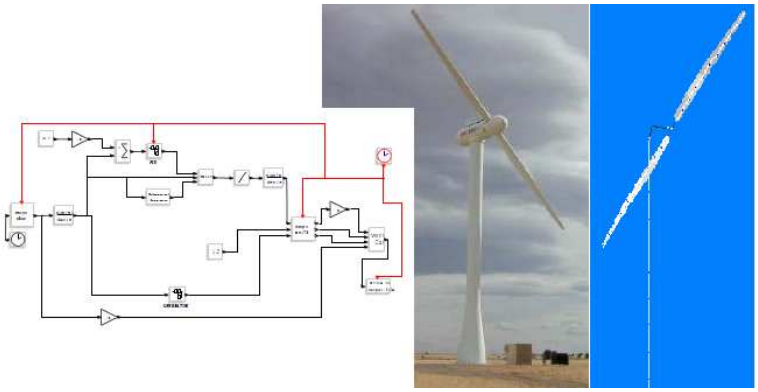


Figure 2.2: A co-simulation setup: the wind turbine in the center is split in a multibody system (on the right) and a block diagram (on the left)

# Chapter 3

# Multirate Algorithms

The idea behind Multirate Methods was born in the Seventies. The first try in studying the stability is presented in [5], one of the first works about the Multirate Methods.
The basics consist in a numerical method that uses more than one time grid to integrate the ODE system. Once the system is split in more subsystems presenting different time scales, the fast subsystems are integrated with finer time grid than the slow ones. The terms defining the coupling between the subsystems are evaluated by means of interpolation or extrapolation.
Since Gear and Wells [5], a lot of effort has been spent in upgrading and extending the possible applications, as well as in studying the mathematical properties of the methods. However, a complete study on this kind of algorithms for co-simulations has been not proposed yet, mainly because multirate methods are usually applied to monolithic sets of equations, solved by a single simulator.

In this chapter, the analytical description of the implemented algorithms is presented, together with a stability and accuracy performances description.

## 3.1 Introduction

Let the system of ODE be partitioned into two subsystems

$$\begin{cases} y' = f(t, y, z) & y(t_0) = y_0; \quad (3.1.1a) \\ z' = g(t, y, z) & z(t_0) = z_0; \quad (3.1.1b) \end{cases}$$

The equation (3.1.1a) represents the fast subsystem and the equation (3.1.1b)

the slow subsystem.

A linear multirate formula which uses $k$ steps is defined by the operator $L_k$

$$L_k[y(t);h] = \sum_{r=0}^{k} \alpha_r y(t-rh) + h\beta_r y'(t-rh) \tag{3.1.2}$$

If the local truncation errors satisfy the relation in (3.1.3)

$$\|L_k[y(t);h]\| >> \|L_k[z(t);h]\| \qquad \forall t \in [t_0, t_f] \tag{3.1.3}$$

then the system (3.1.1) can be integrated using larger steps in (3.1.1b) than in (3.1.1a), without increasing the maximum local truncation error.

A numerical method exploiting this possibility is called a Multirate Method.

In order to achieve good stability properties, it is desirable to integrate the system by an implicit method, such as the Backward Difference Formula (BDF).

## 3.2 Multirate Formulae

A quick recall to some traditional implicit schemes is given below.

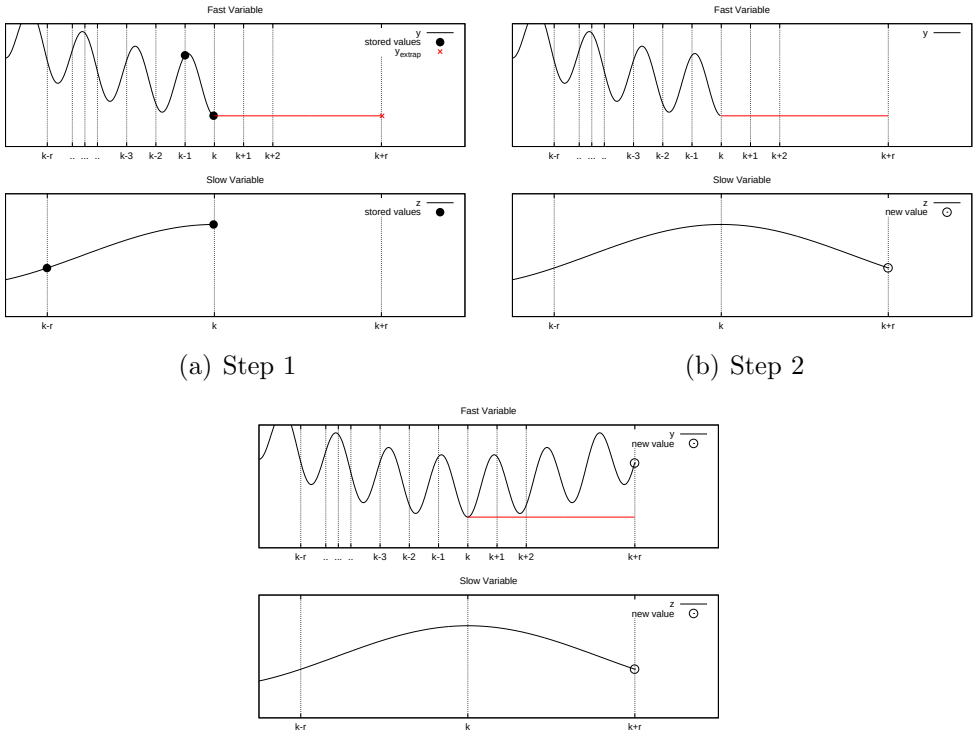
### 3.2.1 Slowest first

With this approach, the slow variable is integrated first (with a large time step $H$). Since the fast variable values are not computed yet, an extrapolation is used to predict them. Then the fast variable is integrated with a small step size $h$, exploiting interpolation to compute the slow variable values on the finer grid.

Often, a zero order extrapolation is used to predict the fast variable extrapolation. This is due to two main reasons: firstly, an accurate prediction is impossible to compute, due to the fact that the time step is incompatible (too large) with the fast variable rate of change. Moreover, only the slow components of the fast subsystem matter in the slow simulator, because of the multirate behavior of the system.

A graphical representation of this scheme is shown in figure 3.1: in step 1 (figure 3.1(a)), the fast variable extrapolation ($\times$) is computed and communicated to the slow simulator, which in step 2 (figure 3.1(b)) computes the new slow variable value; based on this last value, in step 3 (figure 3.1(c)) the fast simulator finally computes all the new fast variable values, for time steps from

$k+1$ to $k+r$ (the last being the new value used for the next extrapolation), where $r = \frac{H}{h}$.




(a) Step 1

(b) Step 2

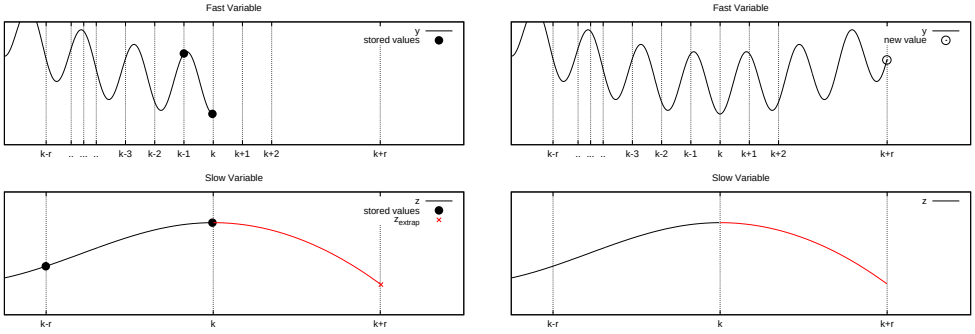
(c) Step3

Figure 3.1: Slowest First chronological sequence

### 3.2.2 Fastest first

The fast variable is integrated, based on extrapolated slow variable values. Then the slow variable is computed by means of an implicit step. This approach can exploit higher order prediction algorithms, and the relative error related to them lowers when the fast components of the fast variable are less important in the slow variable motion integration. The drawback is that if the larger time step is reduced (due to unsatisfactory tolerances in the slow integrator), and the procedure has to be repeated, previous solutions of the fast subsystem are required to cover the whole large time step, thus implying more memory resources. Figure 3.2 gives a representation of this method: in step 1 3.2(a),

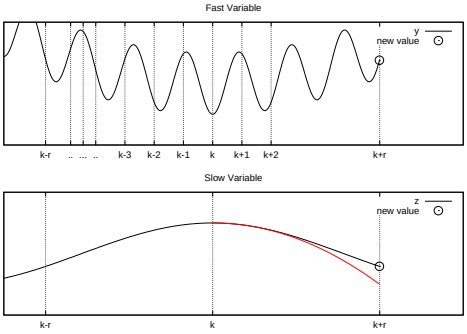
the slow variable extrapolation ($\times$) is computed and communicated to the fast simulator, which in step 2 3.2(b) computes all the new fast variable values, for time steps from $k+1$ to $k+r$; based on the last value, in step 3 3.2(c) the slow simulator finally computes the new slow variable value.




(a) Step 1




(b) Step 2




(c) Step 3

Figure 3.2: Fastest First chronological sequence

### 3.2.3 Compound - Fast

In this approach, the integration is carried once (implicitly) for the whole system (3.1.1), with the large step $H$, so that $y_{k+r}$ and $z_{k+r}$ are computed. Afterwards, only equation (3.1.1a) is integrated with the small step $h$, using interpolated values of $z_{k+1} \dots z_{k+r-1}$.
It has to be noted that $y_{k+r}$ is updated at the end of the time step computation, thus is computed twice. This is used to improve stability with respect to the

previous methods, where extrapolation brings unstable behavior. A more precise description of this algorithm can be found in [7].

### 3.2.4 Generalized Compound-Fast

As for the $Compound-Fast$ scheme, the integration is carried once for both the fast and slow components, but computing $y_{k+\alpha r}$ together with $z_{k+r}$, where $\alpha r \in \mathbb{N}$ and $1 < \alpha r < r$. Afterwards, $y_{k+1} \dots y_{k+r-1}$ are computed using $z_{k+1} \dots z_{k+r-1}$ from interpolation. This time it is $y_{k+\alpha r}$ which is computed twice. A detailed description can be found in [8].

### 3.2.5 Mixed Compound-Fast

This method is obtained as a derivation from the $Generalized\ Compound-Fast$, using $\alpha = \frac{1}{r}$. In this way, $y_{k+1}$ and $z_{k+r}$ are computed first, then $y_{k+2} \dots y_{k+r-1}$ are determined based on interpolated values of $z$. With this algorithm, there are no values that have to be computed twice.

When a co-simulation setup is considered, the subsystems (3.1.1a) and (3.1.1b) are integrated by independent processes. This means that performing an implicit integration for the whole system is not possible in one iteration only. In fact, the fast simulator solving (3.1.1a) for the time step $k + r$ (thus computing $y_{k+r}$) would need $f$ at time $t_{k+r}$, and so needs $z_{k+r}$. In the same way, the slow simulator needs $y_{k+r}$ to compute $z_{k+r}$, and this brings to an inconsistency. The $Compound - Fast$-like methods (the last three presented), once introduced in a co-simulation setup, have exactly this problem.

The only way to compute a completely implicit step is to use a Predictor-Corrector method, thus iterating until convergence at every time step. This setup would be heavily time consuming.

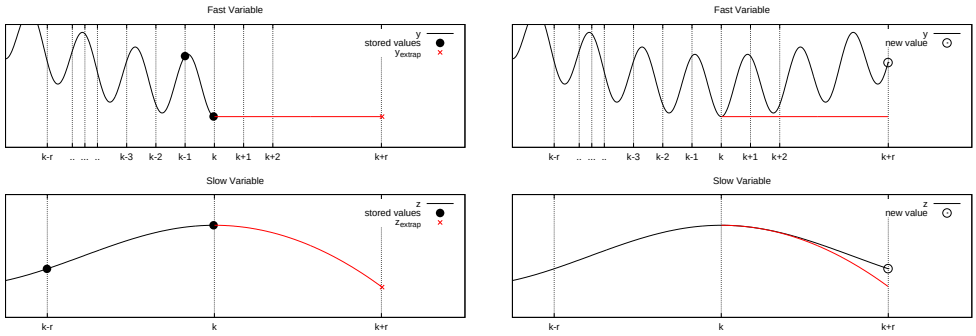
A workaround to this problem is to use a prediction (extrapolation) for the unknown external variable, based on old values. The first two methods ($Slowest\ First$ and $Fastest\ First$) are based on this kind of approach. If, for example, a prediction $\hat{y_{k+r}}$ of the fast variable $y$ is computed based on old values

$y_k$ $y_{k-1}\ldots$ and passed to the slow simulator, the slow simulator can compute $z_{k+r}$, which would be then passed to the fast simulator, so that $y_{k+r}$ can be finally computed. In this case (it is nothing but the *Slowest First* method) only one iteration would be performed at every time step, and the same would happen if the prediction of the slow variable were given to be used in the fast simulator (*Fastest First* method).
Such an organization implies that the extrapolation introduces an explicit behavior in the method, and, very likely, the stability properties will be lower than in fully implicit methods. Together with this goes the fact that also the accuracy of such methods will be poorer, and should be analyzed before implementing these kind of algorithms in a software.

Both the *Slowest First* and the *Fastest First* methods, moreover, imply dead times between the iterations of each subsystem: for such schemes, one simulator needs to wait the other to give results before proceeding with the computations, so that every time one simulator is working, the other is idle.
This might be not important for serial simulation, in which only one processor is used to run both the slow and the fast simulators, because if this was the case, the CPU would always be working. But if the processes were run on distinct machines or on multiple processors, the CPUs would advance one at a time, and the simulation would require more time than it was actually necessary.
An application in a co-simulation setup of these two methods is given in [9], where a simple application is described as test case.

The dead time issue would not disappear neither with the $Compound-Fast$-like algorithm. Thus the defect about computational costs of the last three methods would be further exaggerated in a parallel co-simulation layout.

The method presented in this work uses a setup similar to the *Slowest First* and *Fastest First* methods, but aims to overcome the dead time issue. The solution is to give predictions for both the fast and slow variables (the fast process sees only an extrapolation of the slow variable, and vice versa). By doing this it is not necessary that one process waits for the other to give results, but only needs to wait for a prediction (internally or externally computed), and both simulators can advance at the same time. The figures 3.3 represents the chronological sequence of such a layout: step 1 (figure 3.3(a)), both the slow and fast variables extrapolations are computed and exchanged between the simulators; step 2 3.3(b), the fast and slow simulators perform the integrations at the same time. In what follows, this method will be referred to as **Double Extrapolation Method**.




(a) Step 1 (b) Step 2

Figure 3.3: Double Extrapolation chronological sequence.

## 3.3 New Algorithm Definition

As already stated, in the case the two subsystems are integrated by different processes, it is convenient to carry out the integration using predicted values of the external variable, and then advance by an implicit scheme.

The method described in this work makes use of a linear multistep implicit method. In contrast to a single-rate method (defined in (3.1.2)), the solution of (3.1.1) in a multirate layout is given by (3.3.1).

$$y_{i+1} = \sum_{j=0}^{p-1} (a_j y_{i-j} + h b_j \hat{f}_{i-j}) + h b_p \hat{f}_{i+1} \tag{3.3.1a}$$

$$z_{(i+1)r} = \sum_{j=0}^{p-1} (a_j z_{(i-j)r} + H b_j \hat{g}_{(i-j)r}) + H b_p \hat{g}_{i+1} \tag{3.3.1b}$$

where $\hat{f}_i = f(t_i, y_i, \hat{z}_i)$, $\hat{g}_i = g(t_i, \hat{y}_i, z_i)$ and $\hat{y}$ and $\hat{z}$ are approximation to $y$ and $z$ respectively. Both methods have order $p$. Note that both methods have $b_p \neq 0$, i.e. both are implicit. In addition, the two subsystems are integrated with different timesteps ($h$ and $H$), thus giving a multirate scheme. The ratio $r = \frac{h}{H}$ is called the multirate ratio.

In general, different strategies could be adopted to approximate $y$ and $z$, and even if the same Multistep Method is adopted as internal solver, the approximation strategies may generate even extremely different performances, especially in terms of stability.

As already stated, the method proposed in this work makes use of extrapolations (based on explicit Linear Multistep Methods) to predict both the fast and the slow variable. Moreover, a 2 steps BDF will be selected as internal solver for both processes.

### 3.3.1 Extrapolation method

Extrapolation is given based on state variable and state derivative old values, as for the explicit Multistep Methods.

Considerations about accuracy, efficiency, memory usage and computational costs bring to the conclusion that the best choice is to use a maximum of 2 steps for predictions, thus relating the computation on already stored old variables values (being the integrators built on a BDF 2 steps).

The slow integrator uses a constant (zero-order) extrapolation of the fast variable, because it is not possible to determine a higher order formula showing good behavior in every situations, being the extrapolation step always too large compared to the frequency of the signal.

For use in the fast integrator, the slow variable can be predicted by a higher order scheme (linear or quadratic), using the values of the most recent steps available. Increasing the order to a cubic extrapolation will decrease the stability performance if only 2 steps are used, and will increase the memory requirements if more than 2 steps are required. Thus, no cubic extrapolation is considered in this work.
Even if both linear and quadratic extrapolations will be implemented in the software, the rest of the thesis will focus only on quadratic extrapolation, since this algorithm shows better performances.

## 3.4 The Single-rate Layout

The case in which the two subsystems are integrated with the same time step is of course a special case of the general layout described in the previous sections.

This case is referred to as single-rate (multirate ratio $r = 1$). If this were the case, the distinction between fast and slow variables does not exist any more, and the considerations in section 3.3.1 are no longer valid: in a sinlge-rate layout the extrapolation of both variables can be carried out by the same higher order method.

## 3.5 Stability Analysis

Traditionally, Multistep Methods applied to test problems generate difference equations [11]. Once the difference equation is determined, the stability condition is translated in the roots condition: the method is said to be absolutely stable if and only if all the roots of the characteristic polynomial have an absolute value less than or equal to one.

In this work, it is not possible to select a single differential equation as a test problem, since the aim is to analyze the interaction between different systems. A system of ODEs is instead to be used, for which the application of a Multistep Method gives a system of difference equations. This system can be written by means of a matrix, called the Compound Matrix. The stability analysis is thus carried out through the study of the compound matrix, as in [7], [10]. In particular, the roots condition becomes the spectral radius condition, and the method is said to be absolutely stable if and only if the spectral radius of the compound matrix is less than or equal to one.

### 3.5.1 Compound Matrix

Here below a method is given to build the compound matrix for the general cases of Multistep Linear Multirate Methods, where the coupled terms are approximated either in terms of interpolation or extrapolation, considering both implicit and explicit schemes. After this, such a method will be applied to the particular case of interest. The reason why no restrictions are imposed at this level is to build an instrument giving the possibility to evaluate, also in the future, possible modifications and improvements.

The compound matrix is built for the $(N_f+N_s)\times(N_f+N_s)$ system in (3.5.1), where, without loss of generality, the system is partitioned into two subsystems (the procedure can be easily extended to the case of more subsystems).

$$\begin{pmatrix} \mathbf{y}' \\ \mathbf{z}' \end{pmatrix} = \begin{pmatrix} \boldsymbol{\Lambda}_1 & \boldsymbol{\mu} \\ \boldsymbol{\epsilon} & \boldsymbol{\Lambda}_2 \end{pmatrix} \begin{pmatrix} \mathbf{y} \\ \mathbf{z} \end{pmatrix} \tag{3.5.1}$$

where $\mathbf{y}$ contains the $N_f$ fast variables, and $\mathbf{z}$ contains the $N_s$ slow variables.

Once the relations below are defined

$$\boldsymbol{\alpha_f} = ({}^f a_0,\ {}^f b_0, \ldots,\ {}^f a_{p-1},\ {}^f b_{p-1},\ {}^f b_p) \tag{3.5.2a}$$

$$\boldsymbol{\alpha_s} = ({}^s a_0,\ {}^s b_0, \ldots,\ {}^s a_{q-1},\ {}^s b_{q-1},\ {}^s b_q) \tag{3.5.2b}$$

$$\boldsymbol{A_f} = \boldsymbol{A}(\boldsymbol{\alpha_f}) = \left[\begin{array}{cccc|cc} {}^f a_0\mathbf{I} & {}^f b_0\mathbf{I} & \cdots\cdots & {}^f a_{p-1}\mathbf{I} & {}^f b_{p-1}\mathbf{I} \\ 0 & 0 & \cdots\cdots & 0 & 0 \\ \multicolumn{4}{c|}{\multirow{2}{*}{$\mathbf{I}$}} & \vdots & \vdots \\ \multicolumn{4}{c|}{} & & \end{array}\right] \tag{3.5.3a}$$

$$\boldsymbol{A_s} = \boldsymbol{A}(\boldsymbol{\alpha_s}) = \left[\begin{array}{cccc|cc} {}^s a_0\mathbf{I} & {}^s b_0\mathbf{I} & \cdots\cdots & {}^s a_{q-1}\mathbf{I} & {}^s b_{q-1}\mathbf{I} \\ 0 & 0 & \cdots\cdots & 0 & 0 \\ \multicolumn{4}{c|}{\multirow{2}{*}{$\mathbf{I}$}} & \vdots & \vdots \\ \multicolumn{4}{c|}{} & & \end{array}\right] \tag{3.5.3b}$$

$$\boldsymbol{\beta_f} = \boldsymbol{\beta}(\boldsymbol{\alpha_f}) = ({}^f b_p\mathbf{I}, \mathbf{I}, 0, \ldots, 0)^T \tag{3.5.4a}$$

$$\boldsymbol{\beta_s} = \boldsymbol{\beta}(\boldsymbol{\alpha_s}) = ({}^s b_q\mathbf{I}, \mathbf{I}, 0, \ldots, 0)^T \tag{3.5.4b}$$

$$\mathbf{Y_i} = (\mathbf{y}_i^T, \hat{\mathbf{f}}_i^T, \ldots, \mathbf{y}_{i-p+1}^T, \hat{\mathbf{f}}_{i-p+1}^T)^T \tag{3.5.5a}$$

$$\mathbf{Z_{nk}} = (\mathbf{z}_{nk}^T, \hat{\mathbf{g}}_{nk}^T, \ldots, \mathbf{z}_{(n-q+1)k}^T, \hat{\mathbf{g}}_{(n-q+1)k}^T)^T \tag{3.5.5b}$$

the multistep method (3.3.1) applied to (3.5.1) gives the following relations

$$\mathbf{Y_{i+1}} = \boldsymbol{A_f}\mathbf{Y}_i + h\boldsymbol{\beta_f}\hat{\mathbf{f}}_{i+1} \tag{3.5.6a}$$

$$\mathbf{Z_{(n+1)k}} = \boldsymbol{A_s}\mathbf{Z}_{nk} + H\boldsymbol{\beta_s}\hat{\mathbf{g}}_{(n+1)k} \tag{3.5.6b}$$

where $i = nk, \ldots, (n+1)k$, and

$$\hat{\mathbf{f}}_{i+1} = \boldsymbol{\Lambda_1}\mathbf{y}_{i+1} + \boldsymbol{\mu}\hat{\mathbf{z}}_{i+1} \tag{3.5.7a}$$

$$\hat{\mathbf{g}}_{(n+1)k} = \boldsymbol{\Lambda_2}\mathbf{z}_{(n+1)k} + \boldsymbol{\epsilon}\hat{\mathbf{y}}_{(n+1)k} \tag{3.5.7b}$$

A general way to define approximated values used in the simulation, considering both extrapolation-based and implicit methods, is the following.

$$\hat{\mathbf{y}}_{(n+1)k} = \boldsymbol{\phi_f}\mathbf{Y}_{(n+1)k} \tag{3.5.8}$$

$$\hat{\mathbf{z}}_{i+1} = \boldsymbol{\phi}_j\mathbf{Z}_{(n+1)k} \tag{3.5.9}$$

with $j = i - nk + 1$. Details about $\boldsymbol{\phi_f}$ and $\boldsymbol{\phi}_j$ will be given in (3.5.13)

Defining the rectangular matrices

$$\begin{aligned}\mathbf{E_{11}} &= (h\mathbf{\Lambda_1}, 0, \ldots, 0)\\ \mathbf{E_{22}} &= (H\mathbf{\Lambda_2}, 0, \ldots, 0)\\ \mathbf{E_{12}} &= (h\boldsymbol{\mu}, 0, \ldots, 0)\\ \mathbf{E_{21}} &= (H\boldsymbol{\epsilon}, 0, \ldots, 0)\end{aligned}$$

leads to

$$\mathbf{Y_{i+1}} = \mathbf{A_f}\mathbf{Y}_i + \mathbf{B_{11}}\mathbf{Y}_{i+1} + \mathbf{B_{12}}\boldsymbol{\phi}_j\mathbf{Z}_{(n+1)k} \tag{3.5.10a}$$

$$\mathbf{Z_{(n+1)k}} = \mathbf{A_s}\mathbf{Z}_{nk} + \mathbf{B_{22}}\mathbf{Z}_{(n+1)k} + \mathbf{B_{21}}\boldsymbol{\phi_f}\mathbf{Y}_{(n+1)k} \tag{3.5.10b}$$

where:

$$\begin{aligned}\mathbf{B_{11}} &= \boldsymbol{\beta_f}\mathbf{E_{11}}\\ \mathbf{B_{12}} &= \boldsymbol{\beta_f}\mathbf{E_{12}}\\ \mathbf{B_{21}} &= \boldsymbol{\beta_s}\mathbf{E_{21}}\\ \mathbf{B_{22}} &= \boldsymbol{\beta_s}\mathbf{E_{22}}\end{aligned}$$

Defining the new matrices

$$\begin{aligned}\mathbf{\Lambda_f} &= (\mathbf{I} + \mathbf{B_{11}})^{-1}\mathbf{A_f}\\ \mathbf{\Lambda_s} &= (\mathbf{I} + \mathbf{B_{22}})^{-1}\mathbf{A_s}\\ \mathbf{\Gamma}_{\mathbf{f}i} &= (\mathbf{I} + \mathbf{B_{11}})^{-1}\mathbf{B_{12}}\mathbf{\Phi}_j\\ \mathbf{\Gamma_s} &= (\mathbf{I} + \mathbf{B_{22}})^{-1}\mathbf{B_{21}}\mathbf{\Phi_f}\end{aligned}$$

the system (3.5.10) can be written using a more compact notation:

$$\mathbf{Y_{i+1}} = \mathbf{\Lambda_f}\mathbf{Y_i} + \mathbf{\Gamma}_{\mathbf{f}i}\mathbf{Z_{(n+1)k}} \tag{3.5.11a}$$

$$\mathbf{Z_{(n+1)k}} = \mathbf{\Lambda_s}\mathbf{Z_{nk}} + \mathbf{\Gamma_s}\mathbf{Y_{(n+1)k}} \tag{3.5.11b}$$

and the compound step is finally given by

$$\mathbf{Y_{(n+1)k}} = \mathbf{\Lambda_f}^k\mathbf{Y_{nk}} + \sum_{i=0}^{k-1}\mathbf{\Lambda_f}^{(k-1-i)}\mathbf{\Gamma}_{\mathbf{f}i}\mathbf{Z_{nk}} \tag{3.5.12a}$$

$$\mathbf{Z_{(n+1)k}} = \mathbf{\Lambda_s}\mathbf{Z_{nk}} + \mathbf{\Gamma_s}\mathbf{Y_{nk}} \tag{3.5.12b}$$

**Extrapolation algorithm**

The matrices $\boldsymbol{\phi_f}$ and $\boldsymbol{\phi}_i$ define the extrapolation algorithm. Changing the extrapolation technique will carry to different methods: fully implicit, Slowest First, Fastest First, and so on.

Once the algorithm is chosen, building such matrices is straightforward. The example leading to the new method presented in this work is given here below.

$$\boldsymbol{\phi_f} = (\mathbf{0}, \mathbf{0}, \mathbf{I}, \mathbf{0}, \ldots, \mathbf{0}) \tag{3.5.13a}$$

$$\boldsymbol{\phi}_i = (\mathbf{0}, \mathbf{0}, {}^i v_0 \mathbf{I}, {}^i d_0 \mathbf{I}, \ldots, {}^i v_{q-1} \mathbf{I}, {}^i d_{q-1} \mathbf{I}) \tag{3.5.13b}$$

where the coefficients ${}^i v_k$ ${}^i d_k$ define the extrapolation method for the slow variable.

In the same way, it is easy to show how the traditional algorithm can be defined in this context. For the Slowest First method:

$$\boldsymbol{\phi_f} = (\mathbf{0}, \mathbf{0}, \mathbf{I}, \mathbf{0}, \ldots, \mathbf{0}) \tag{3.5.14a}$$

$$\boldsymbol{\phi}_i = ({}^i m_0 \mathbf{I}, {}^i n_0 \mathbf{I}, {}^i m_{-1} \mathbf{I}, {}^i n_{-1} \mathbf{I}, \mathbf{0}, \ldots, \mathbf{0}) \tag{3.5.14b}$$

where the coefficients ${}^i m_k$ ${}^i n_k$ define the interpolation method for the slow variable.

For the Fastest First method:

$$\boldsymbol{\phi_f} = (\mathbf{I}, \mathbf{0}, \ldots, \mathbf{0}) \tag{3.5.15a}$$

$$\boldsymbol{\phi}_i = (\mathbf{0}, \mathbf{0}, {}^i v_0 \mathbf{I}, {}^i d_0 \mathbf{I}, \ldots, {}^i v_{q-1} \mathbf{I}, {}^i d_{q-1} \mathbf{I}) \tag{3.5.15b}$$

By such an approach, changing the coefficients in (3.5.2), (3.5.4) and (3.5.13), (3.5.14) or (3.5.15), the stability properties of many linear Multistep Multirate schemes, applied to a general linear ODE system, can be determined.

### 3.5.2 Test Equation and Results

Results in terms of stability performances are given comparing the stability region of the traditional BDF scheme with some multirate methods. In particular, results are presented for the Double Extrapolation Method, for a comparison with the most similar traditional multirate algorithm, the Slowest First and the Fastest First.

To this aim, the linearized two-dimensional ODE system (3.5.16) is used.

$$\begin{pmatrix} y' \\ z' \end{pmatrix} = \begin{pmatrix} \lambda & \mu \\ \epsilon & \alpha\lambda \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} \tag{3.5.16}$$

where $\lambda = p + j\omega$ is a complex number, $j$ is the imaginary unit and $\alpha < 1$ is the frequency ratio.

Now, consider the systems in which $p = 0$. If the stability of the algorithm haves to be determined, it is necessary to select linear systems with simple stability (without positive nor negative damping). This means that the diagonal terms in (3.5.16) must be imaginary, and the real part of the eigenvalues must be set to zero. For the system (3.5.16) with $\lambda$ pure imaginary, it is easy to demonstrate that this condition leads to

$$\frac{\mu\epsilon}{(\frac{\omega_1-\omega_2}{2})^2} < 1 \tag{3.5.17}$$

The ratio in (3.5.17) will be used also in the general case in which $p \neq 0$, to give the measure of the coupling of the subsystems, defined in (3.5.18):

$$\beta = \frac{\mu\epsilon}{(\frac{\omega_1-\omega_2}{2})^2} \tag{3.5.18}$$

$\beta$ is thus the measure of the coupling, and imposing $\beta < 1$ gives simply stable systems for $\lambda$ pure imaginary.

The stability of the Multirate Methods has been analyzed since the beginning of the 1980s, exploiting different ways.

A stability analysis can be found in [10], where, however, damped systems are selected to define stability regions of the methods. Thus, the regions defined in that work are wider than actual regions.

Also in [7] a stability analysis is presented. In that case, for a general case of a Multirate Linear Multistep Method, the spectral radius of the compound matrix is studied indirectly using a $2x2$ matrix $\hat{M}$ for which the spectral radius $\rho(\hat{M})$ is demonstrated to be higher than the actual compound matrix spectral radius $\rho(M)$, thus having $\rho(\hat{M}) < 1$ necessarily leads to $\rho(M) < 1$. Thus, in this case the region determined is narrower than the actual one.

In this work, the stability region bounds are not given in closed form. For each case considered, only the graphical representation of the region is presented. As traditionally done for the numerical integration methods, the stability regions

are determined on the complex plane. For a single equation such as $y' = \lambda y$, the points of the complex plane correspond to $h\lambda$, with $h$ being the dimension of the time step and $\lambda$ a complex number. For the system in (3.5.16), the same convention is adopted, and the points of the complex plane in the figures shown in this work always correspond to $h\lambda$, with $h$ being the time step size. In all the figures, the green area represents the stability region.

The results for the test case with $\alpha = 0.1$ and $\beta = 0.1$ are shown in figures from 3.4 to 3.7. In figure 3.4 the stability region of the traditional BDF method is drawn (it is given only for comparison), while figures 3.5 and 3.6 show the Slowest First and the Fastest First case respectively, and figures 3.7 finally represent the stability results for the Double Extrapolation Method.

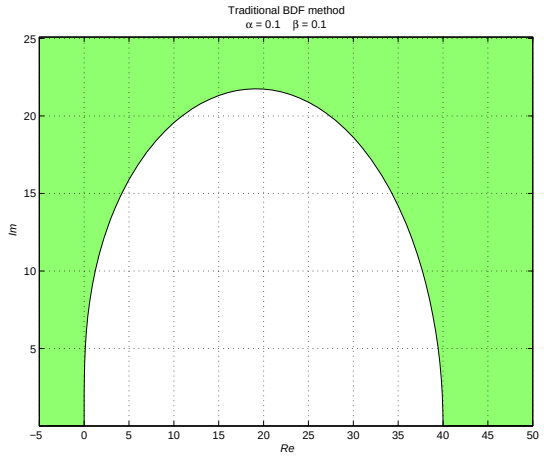
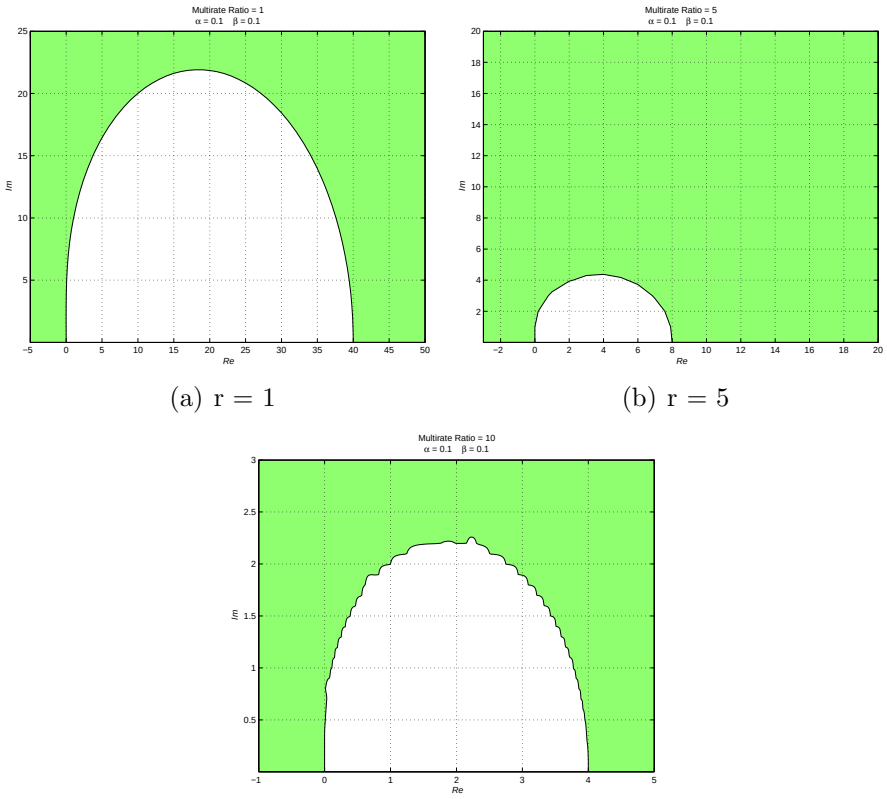



Figure 3.4: Stability region of the BDF Method

The Fastest First and the Double Extrapolation methods loose the A-stability faster than the Slowest First (figures 3.7(c) and 3.6(c)).

The Double Extrapolation Method has worse stability properties compared to the other two methods, for higher multirate ratios. This is obviously due to the fact that the traditional SF and FF methods use only extrapolated values for one subsystem, while for the other the actual solution is used. In the Double Extrapolation Method, instead, two extrapolations are used, leading to a stronger explicit character of the method, which make the formula loose the stability properties a little faster. This kind of layout was the one chosen in defining the new method to avoid dead times (as already mentioned in section 3.2), and this behavior of stability properties was expected.




(a) r = 1

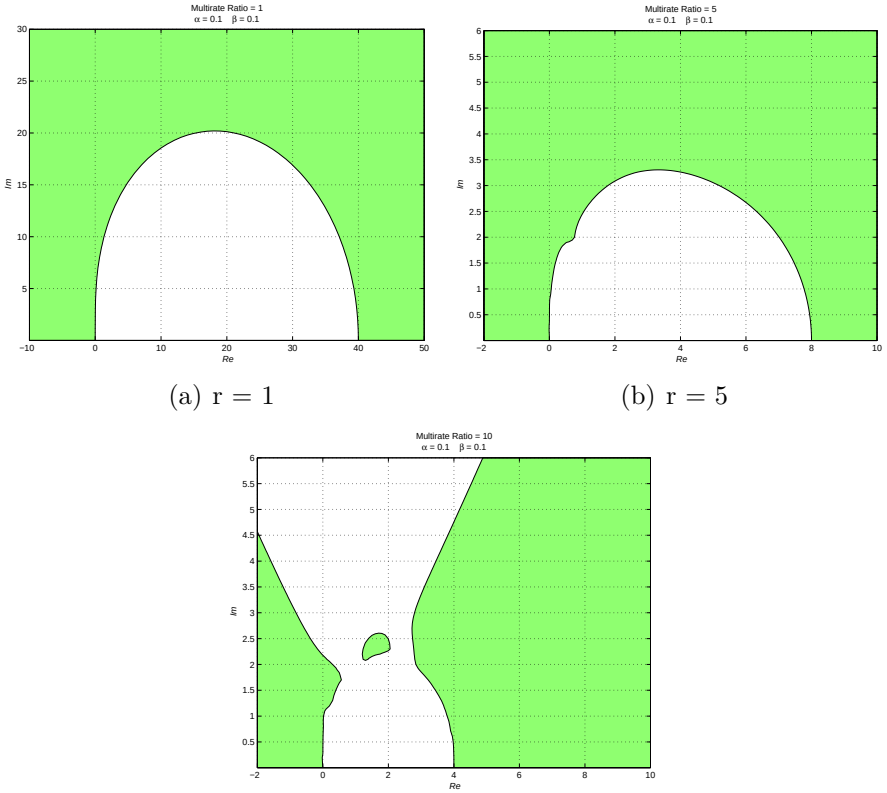
(b) r = 5

(c) r = 10

Figure 3.5: Stability region of the Slowest First Method, for multirate ratio $r$ equal to 1 3.5(a), to 5 3.5(b) and 10 3.5(c)

It must be noted that the Double Extrapolation Method does not loose the stability on the imaginary axis, for multirate ratio equal to 1 (3.7(a)) and to 5 (3.7(b)). For $r = 10$, the stability condition is slightly lost, and the method gives acceptable results only for systems that are at least weakly damped.

It is important to understand how the stability performances change by varying the parameters $\alpha$ and $\beta$. Figures 3.8 show the behavior of the stability region for the Double Extrapolation method when the parameter $\alpha$ changes from 0.001 (figure 3.8(d)) to 0.01 (figure 3.8(a)), and $\beta = 0.05$ Again, it is of interest to observe the stability properties on the imaginary axis: for $\alpha = 0.1$ and 0.05 (figures 3.8(a) 3.8(b)), the stability is unconditioned, and there is no restriction in selecting the time step for undamped systems. Figures 3.8(c) 3.8(d), instead,




(a) r = 1

(b) r = 5

(c) r = 10

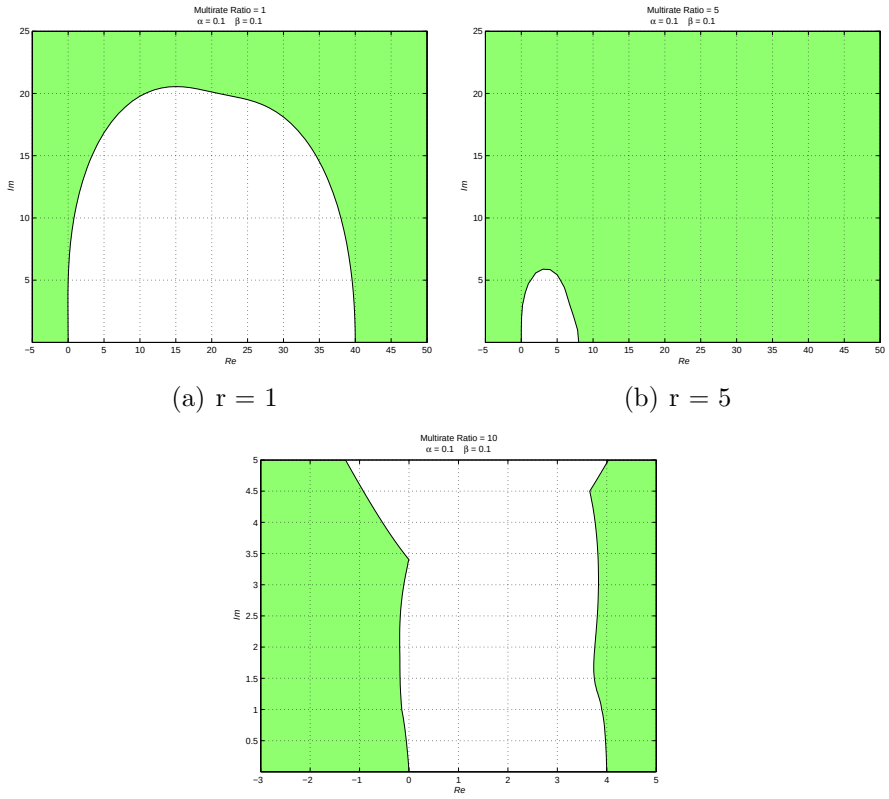
Figure 3.6: Stability region of the Fastest First Method, for multirate ratio $r$ equal to 1 3.6(a), to 5 3.6(b) and 10 3.6(c)

show two conditionally stable methods, where stability is reached only for a finite interval of time step values.

In figures 3.8 the change in the stability region is shown for $\beta$ varying from 0 (meaning no coupling between the susbsystems, figure 3.9(a)) to 0.8 (figure 3.9(d)).

### 3.5.3 Validity of Stability Results

To analyze the stability performances of the methods, a linear ODE system has been selected.
The systems the multibody simulators deal with, however, are nonlinear Differ-




(a) r = 1

(b) r = 5

(c) r = 10

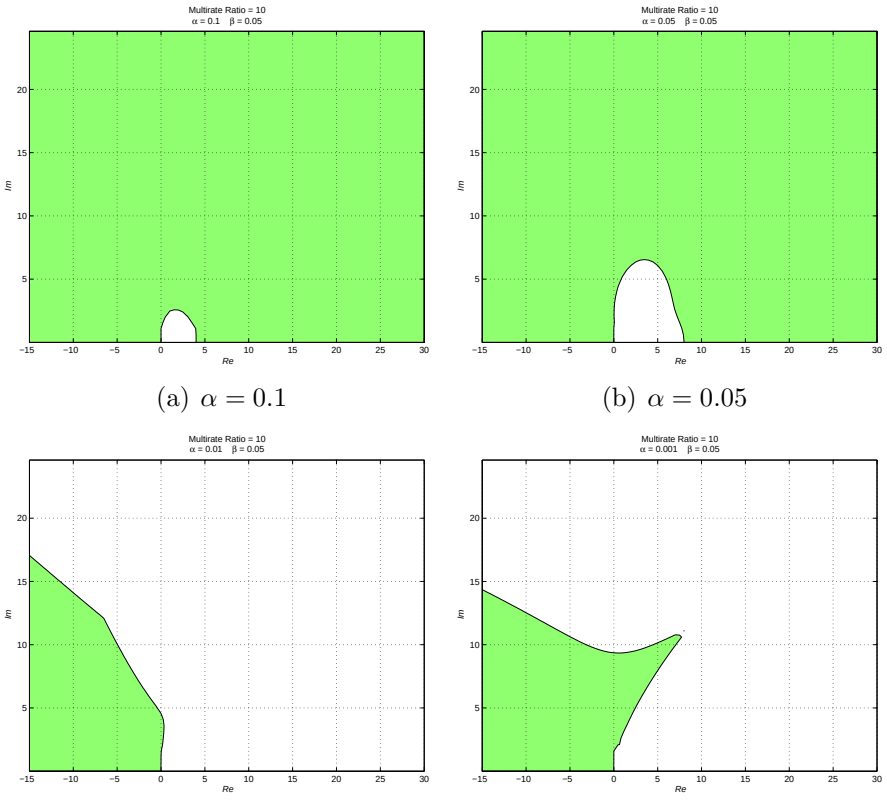
Figure 3.7: Stability region of the Double Extrapolation Method, for multirate ratio $r$ equal to 1 3.7(a), to 5 3.7(b) and 10 3.7(c)

ential Algebraic Equation (DAE) systems.
This means that the stability results presented in this section do not guarantee that the methods analyzed will show the same stability properties, once implemented in the co-simulation environment and used for real mechatronic systems modeling.

## 3.6 Accuracy

In order to determine the accuracy of the method, a comparison with analitical results is carried for the test equation (3.5.16).

Figure 3.10 shows the convergence order of the multirate methods that this




(a) $\alpha = 0.1$ (b) $\alpha = 0.05$

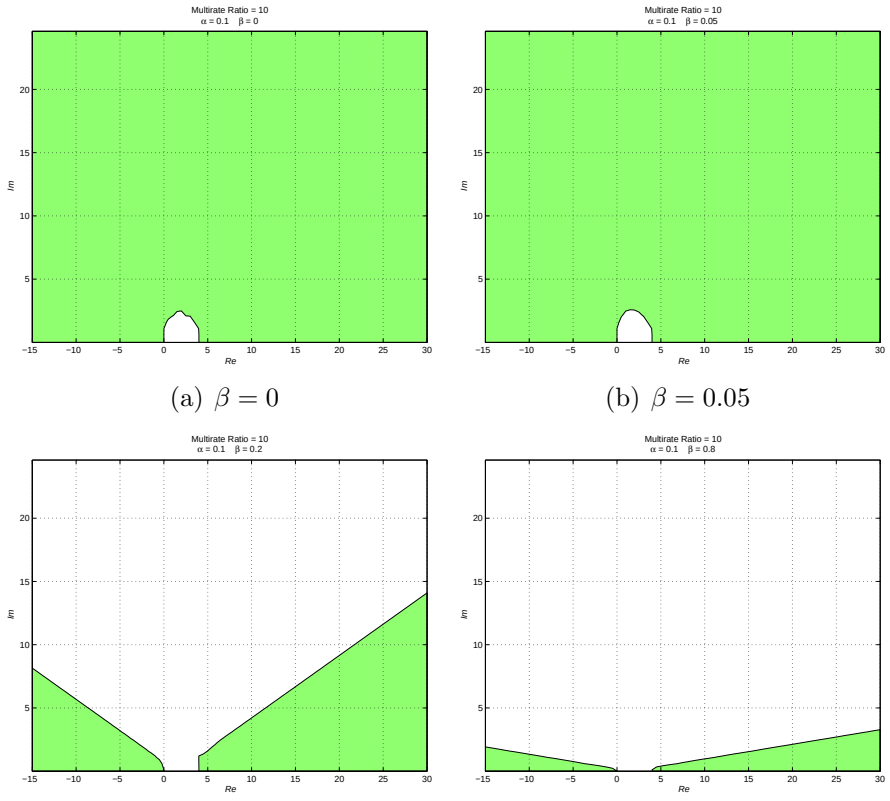
(c) $\alpha = 0.01$ (d) $\alpha = 0.001$

Figure 3.8: Stability region of the Double Extrapolation Method, for multirate ratio $r = 10$, $\beta = 0.05$, and $\alpha$ varying from 0.001 to 0.01

thesis deals with. Error is expressed in $L_{\text{inf}}$ norm. It is easy to note that for larger values of the coupling coefficient $\beta$ the order of convergence lowers from 2 to about 1.

## 3.7 Overcoming the Synchronization Restriction

An additional improvement regards the synchronization between the distinct processes. In what it has been discussed so far, it is implied that the processes exchange data at every macro step, at the time instants of the coarse time mesh



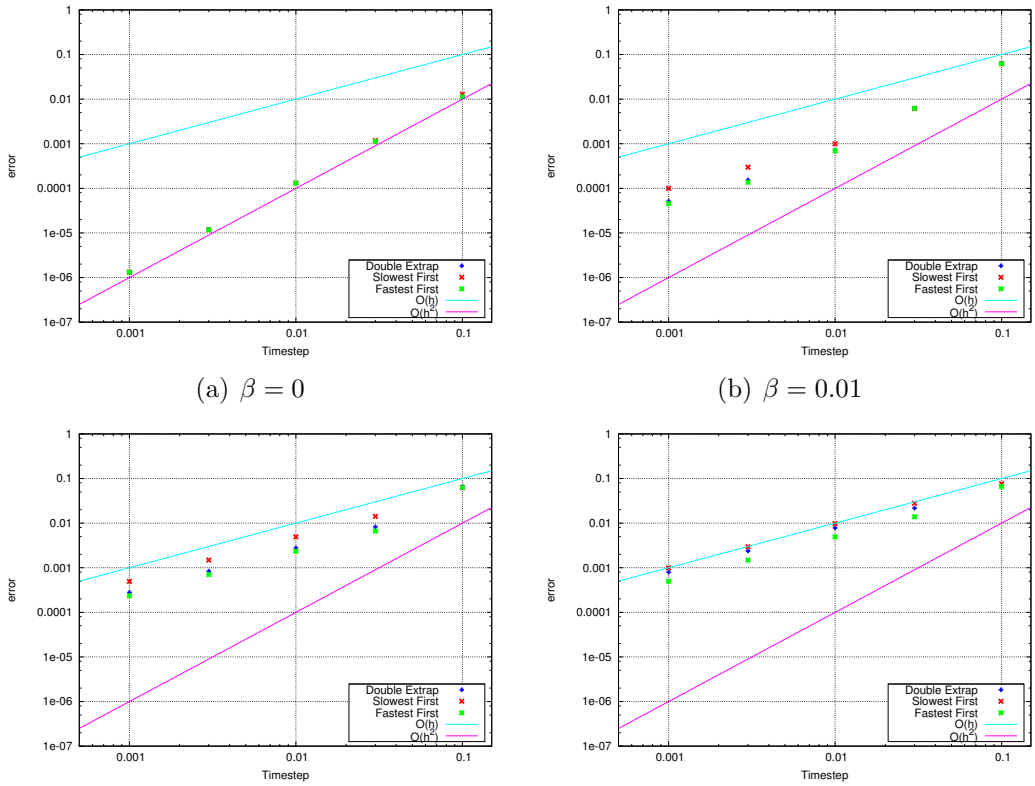

(a) $\beta = 0$ (b) $\beta = 0.05$

(c) $\beta = 0.2$ (d) $\beta = 0.8$

Figure 3.9: Stability region of the Double Extrapolation Method, for multirate ratio $r = 10$, $\alpha = 0.1$, and $\beta$ varying from 0 to 0.8

(the grid of the slow simulator), as shown in figure 3.11. Such a constraint leads to a restriction in the simulation that might be left apart.

It is natural that if the simulators follow their own internal numerical method, they advance at every step of a time length defined internally by the method, and it is not guaranteed that every node of the slow time grid has a correspondence with a node of the fast simulator grid, i.e. it is not guaranteed that the slow time step is an integer multiple of fast time step.
Therefore, if the synchronization is required, it has to be imposed. Otherwise, a workaround needs to be made up such that no synchronism is required. This is simply achieved by means of interpolation [9]. In fact, when the value, for instance, of the fast variable, is required outside the fast time grid, an interpo-




(a) $\beta = 0$ (b) $\beta = 0.01$

(c) $\beta = 0.05$ (d) $\beta = 0.1$

Figure 3.10: Convergence plot for the Double Extrapolation, Slowest First and Fastest First methods, for $\alpha = 0.1$ and different values of $\beta$.

lation is executed using the two or more nearest grid nodes, as shown in figure 3.12.

It must be noted that with the Double Extrapolation scheme coupled with a 2 steps $BDF$, (which is the method proposed in this work, this workaround is automatically implemented in the method itself, because each subsystem only sees an extrapolated value of the external variable, as can be easily understood by looking at (3.5.2), and considering that, for the $BDF$ methods, ${}^f b_0, \ldots, {}^f b_{p-1}$ and ${}^s b_0, \ldots, {}^s b_{q-1}$ are all zeros.
Thus, for both the subsystems (both use $BDF$), when the extrapolation algorithm is called, this is automatically set to calculate the values of the external variable at the nodes of the internal time grid.

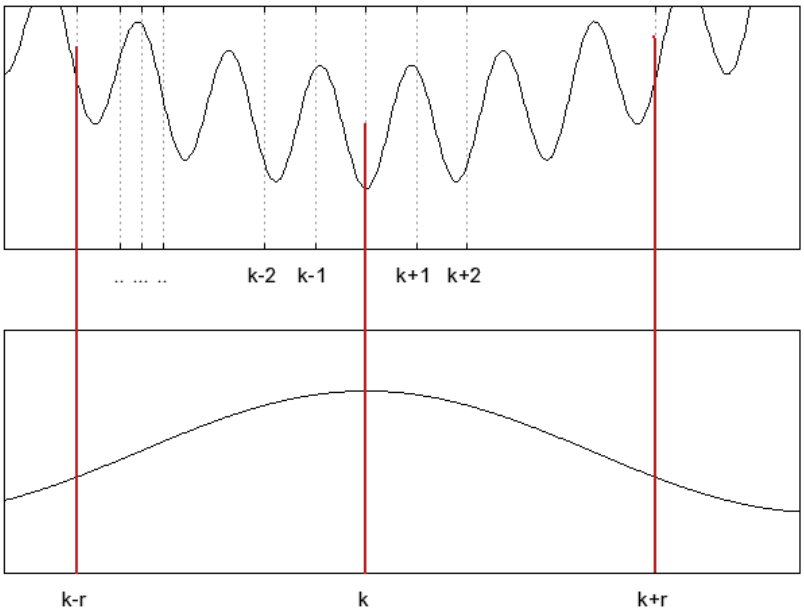



Figure 3.11: Synchronized time grids setup

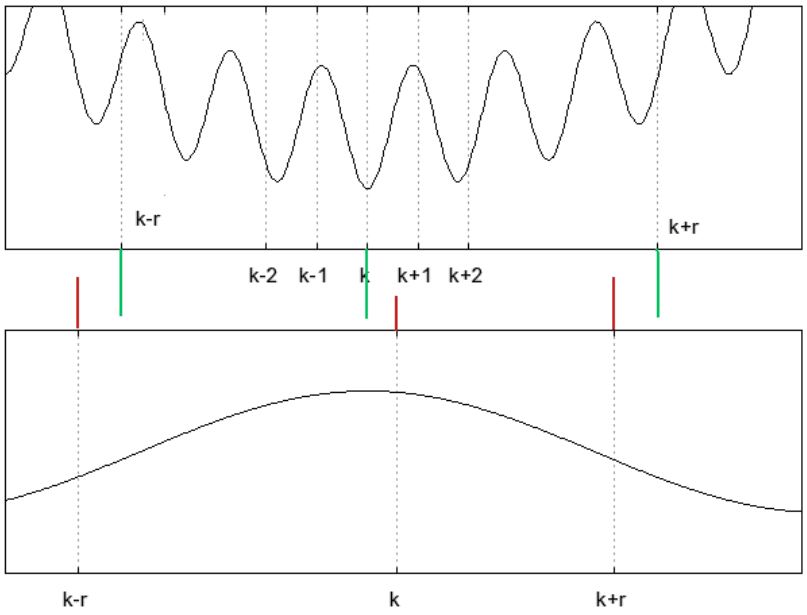



Figure 3.12: Non-synchronized time grids

# Chapter 4

# Software Environment Design

The software environment is constructed by basically choosing a Multibody and a block scheme simulators, and the way the two tools will communicate. As the general intent of this work is to begin a longer term developing, directly involving as many users as possible, free softwares will be selected. It is likely that, during the developing of a layout such as the one proposed in this work, some extensions will have to be added to the software tools initially selected. This is another causal factor addressing the choice on free softwares, as this category is intrinsically proposed to the users as a base to construct new tools, rather than just out of the box instruments.

## 4.1 Software Tools Selection

### 4.1.1 Multibody Simulator

The multibody simulator selected is MBDyn [20], a free software developed at Dipartimento di Ingegneria Aerospaziale of Politecnico di Milano (DIAPM), Italy.

MBDyn is already predisposed for external communication, both in local and network layout. The functions implemented use Internet sockets and Unix domain sockets, which basically drive endpoints of bidirectional communication on the kernel ports. The *Stream* drive is the function making the communication available (see MBDyn input manual for details [17]).

The analyses this software performs are based on an original formulation for the direct time integration of Initial Value Problems (IVP), which is written as

a system of first-order Differential-Algebraic Equations (DAE), using implicit (nearly) L-stable integration algorithms [12].

### Unconstrained Dynamics

The equations of motion (EOMs) of a general mechanical system are usually derived by the Newton-Euler approach. For a system of unconstrained bodies, EOMs are described by

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} = \mathbf{l}(\mathbf{q}, \dot{\mathbf{q}}, t) \tag{4.1.1}$$

If $n_b$ is the number of nodes forming the system, $n = 6n_b$ is the dimension of the system (the number of equations and degrees of freedom).
$\mathbf{q} \in \mathbb{R}^n$ contains the kinematic variables of the nodes (or the generalized coordinates), and $\mathbf{M}(\mathbf{q})$ is the mass matrix, which may depend on $\mathbf{q}$.
$\mathbf{l}(\mathbf{q}, \dot{\mathbf{q}}, t)$ is the function describing the external forces acting on nodes, and may include structural deformability contributions.
Written in the form of a first order system, equation (4.1.1) becomes

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{q}} = \mathbf{p} \tag{4.1.2a}$$

$$\dot{\mathbf{p}} = \mathbf{l}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{p}, t) \tag{4.1.2b}$$

where $\mathbf{p} \in \mathbb{R}^n$ is the momenta and momentum moments vector.

### Constrained Dynamics

There exist two types of constraints, holonomic and non-holonomic. Both are modeled by adding explicit algebraic relations

$$\mathbf{0} = \mathbf{\Phi}(\mathbf{q}, t) \tag{4.1.3}$$

in the case of holonomic constraints, while for non-holonomic constraints

$$\mathbf{0} = \mathbf{\Psi}(\mathbf{q}, \dot{\mathbf{q}}, t) \tag{4.1.4}$$

By using Lagrange's multipliers formalism, system (4.1.2b) becomes

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{q}} = \mathbf{p} \tag{4.1.5a}$$

$$\dot{\mathbf{p}} + \mathbf{\Phi}^{\mathbf{T}}_{/\mathbf{q}}\lambda + \mathbf{\Psi}^{\mathbf{T}}_{/\dot{\mathbf{q}}}\mu = \mathbf{l}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{p}, t) \tag{4.1.5b}$$

$$\mathbf{\Phi}(\mathbf{q}, t) = \mathbf{0} \tag{4.1.5c}$$

$$\mathbf{\Psi}(\mathbf{q}, \dot{\mathbf{q}}, t) = \mathbf{0} \tag{4.1.5d}$$

where $\lambda$ and $\mu$ are the Lagrange's multipliers.

### Numerical Integration

The Differential Algebraic Equation system (4.1.5d) has the general form

$$\mathbf{h}(\dot{\mathbf{x}}, \mathbf{x}, t) = \mathbf{0} \tag{4.1.6}$$

where $\mathbf{x} = (\mathbf{q}, \mathbf{p}, \lambda, \mu)^{\mathbf{T}}$.

The numerical solution at time $t_{k+1}$ by means of the general multistep method (3.1.2) is obtained by solving (4.1.6) for $\dot{\mathbf{x}}_{k+1}$ with

$$\mathbf{x}_{k+1} = \sum_{j=0}^{p-1}(a_j\mathbf{x}_{k-j} + hb_j\dot{\mathbf{x}}_{k-j}) + hb_p\dot{\mathbf{x}}_{k+1} \tag{4.1.7}$$

Using a Newton-Raphson scheme, the solution is given by

$$\mathbf{h}_{/\dot{\mathbf{x}}}\delta\dot{\mathbf{x}}_{\mathbf{k+1}} + \mathbf{h}_{/\mathbf{x}}\delta\mathbf{x}_{\mathbf{k+1}} = -\mathbf{h} \tag{4.1.8}$$

Inserting (4.1.7) in (4.1.8) yields

$$(\mathbf{h}_{/\dot{\mathbf{x}}} + hb_p\mathbf{h}_{/\mathbf{x}})\delta\dot{\mathbf{x}}_{\mathbf{k+1}} = -\mathbf{h} \tag{4.1.9}$$

because

$$\delta\mathbf{x}_{\mathbf{k+1}} = hb_p\delta\dot{\mathbf{x}}_{\mathbf{k+1}} \tag{4.1.10}$$

For each time step (that is, for each instant $t_k$ of the time grid), iterations are performed to solve (4.1.9), until convergence is reached.

**Multirate Numerical Integration**

If the system is divided into two subsystems, equation (4.1.6) becomes

$$\mathbf{f}(\dot{\mathbf{y}}, \mathbf{y}, \mathbf{z}, t) = \mathbf{0} \tag{4.1.11}$$

$$\mathbf{g}(\dot{\mathbf{z}}, \mathbf{z}, \mathbf{y}, t) = \mathbf{0} \tag{4.1.12}$$

and if a generic multirate multistep scheme is used, as the one in (3.3.1), time derivatives for the new time step are computed based on interpolations or extrapolations of old values. Equation (4.1.7) is transformed into

$$\mathbf{y}_{k+1} = \sum_{j=0}^{p-1}(a_j \mathbf{y}_{k-j} + h b_j \dot{\mathbf{y}}_{k-j}) + h b_p \dot{\hat{\mathbf{y}}}_{k+1} \tag{4.1.13a}$$

$$\mathbf{z}_{(k+1)r} = \sum_{j=0}^{p-1}(a_j \mathbf{z}_{(k-j)r} + H b_j \dot{\mathbf{z}}_{(k-j)r}) + H b_p \dot{\hat{\mathbf{z}}}_{(k+1)r} \tag{4.1.13b}$$

As described in section 3.3, $\dot{\hat{\mathbf{y}}}$ and $\dot{\hat{\mathbf{z}}}$ are approximations of $\dot{\mathbf{y}}$ and $\dot{\mathbf{z}}$, respectively, $h$ is the small time step and $H$ is the large time step.
If extrapolation from old values is used to compute approximations

$$\dot{\hat{\mathbf{y}}}_{k+1} = \alpha(\mathbf{y_{k+1}}, \mathbf{z_k}, \ldots, \mathbf{z_{(k-q)r}}) \tag{4.1.14a}$$

$$\dot{\hat{\mathbf{z}}}_{(k+1)r} = \beta(\mathbf{z_{(k+1)r}}, \mathbf{y_{kr}}, \ldots, \mathbf{y_{kr-o}}) \tag{4.1.14b}$$

where $q$ and $o$ are the order of slow variable and fast variable extrapolations, respectively.

Equations (4.1.8) and (4.1.9) formally do not change, except for the fact that approximated values are used for time derivatives.

If both equations in (4.1.14) are followed, the Double Extrapolation Method presented in section 3.2 is obtained.

Depending on the approximation technique, several multirate methods can be carried out following this procedure:

- If the fast variable derivative is computed by (4.1.14a) and the slow variable derivative is the actual solution (thus $\dot{\hat{\mathbf{z}}}_{(k+1)r} = \dot{\mathbf{z}}_{(k+1)r}$), the Fastest First Method is derived.

- If the slow variable derivative is computed by (4.1.14b) and the fast variable derivative is computed by interpolation, the Slowest First Method is

derived.

- If the slow variable derivative is given in terms of actual value and the fast variable derivative is computed by interpolation, a fully implicit method is constructed, such as the Compound-Fast, the General Compound-Fast and the Mixed Compound-Fast methods.

### 4.1.2 Block Scheme Simulator

This kind of simulators have been developed for many years by different groups: examples are Simulink [21] by Mathworks, SystemBuild [22] by National Instruments and the Scicos [23] based ScicosLab [24] by INRIA and Xcos [25] by the Scilab Consortium.

Considering the efficiency and the versatility, Simulink is probably the best simulator available. On the other side, if the user freedom (see section 1.2) related to the software is the argument driving the comparison, ScicosLab becomes the best candidate. All the indicated simulators, though, are widely recognized by the industry and the academic world to be valid instruments.

The choice has been directed to ScicosLab. Since the Scicos based simulators have very similar properties and function calls, the developed MBDyn interface has been tested both with Scilab (Scicos until the 5.1.x versions and Xcos since 5.2.x) and ScicosLab, and both will be proposed to the MBDyn and Scicos/ScicosLab users. Minor differences distinguish the two versions.

Even though no particular communication functions are implemented in Scicos (other that text and audio file interactions) to talk to external processes, it is possible to define new blocks by compiling and linking user defined functions written in C, Fortran or Scilab language. This possibility has been exploited to construct socket based communications (both in local and network setup).

## 4.2 Inter-process Communication

The inter-process communication is constructed based on TCP/IP Internet sockets (with the local communication being a special subcase of the network-based data exchange). This kind of data transmission is very efficient (way more than, for example, employing text file read/write functions), and it is naturally directed to multiple machines layout, which, in many cases, can sensitively decrease the time-to-solution.

In the layout used for the cases presented here, the multibody software MBDyn and the block diagram simulator Scicos communicate through *blocking* sockets (the process reading informations waits until all the expected data are available). In this way, the communication process acts also as synchronizer.

## 4.3 A Simple Test

An inverse pendulum has been used as a test bench of the software tool designed. The problem consists in controlling the angular position of the pendulum by means of a PD controller. This case is used to show the way the simulator operates. A single-rate layout is used.

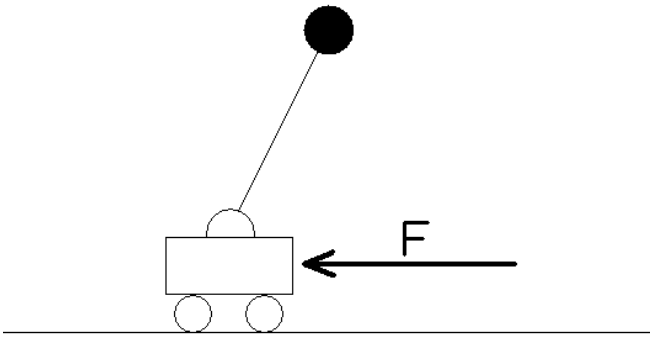



Figure 4.1: Inverse pendulum multibody model: the force F is used to control the angular position of the pendulum

In this case, MBDyn creates a server socket on a kernel port, and waits until Scicos creates the client socket on the same port. After the connection succeeds, the simulation starts: at each time step, MBDyn solves the nonlinear dynamics of the pendulum, and is asked to output the angular position and the angular velocity of the pendulum, which are written as doubles on the kernel port. Scicos reads the data on the port, computes the control force, and write it on the same port as before. At the next step, MBDyn computes the motion of the pendulum reading the input force from the socket, and so on until the simulation ends.

Angular position and input force resulting form the co-simulation are shown in figures 4.3

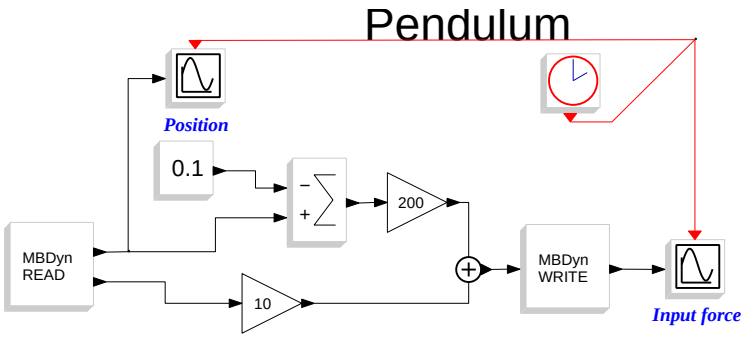
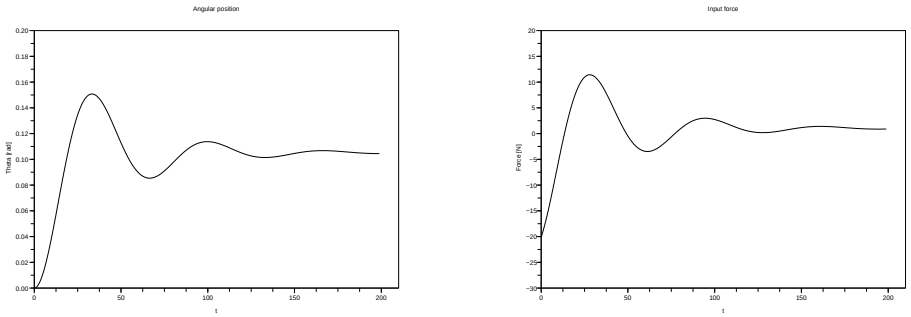



Figure 4.2: PD controller: position and angular velocity are used to compute the control force




(a) Angular position vs time

(b) Input force vs time

Figure 4.3: Inverse pendulum results

## 4.4 Future Enhancement

A possible advantage might be introduced with a Predictor-Corrector setup. For such a layout, the method exposed here (Double Extrapolation) could be used for both the prediction and the correction (with the difference that in the correction iteration, the predicted values are used instead of the extrapolations). Improvement either in terms of accuracy and stability should be expected for this extension.

Another possibility is to integrate the system (3.1.1) by an implicit step, implementing different methods, such as those described in sections 3.2.3, 3.2.4 and 3.2.5. This kind of implementation, as already stated in section 3.2, could result in an inefficient setup. However, enhanced stability and accuracy proper-

ties should be expected. It is possible that such a layout would be useful only for batch simulation, while real-time simulations would be carried out in other ways.

# Chapter 5

# Wind Energy Application

The aim of this chapter is to illustrate the feasibility of the working environment developed.

In order to test the performances of the working environment, a model of a real plant has been built, in which the coupling between active control, dynamics, and structural dynamics has a great relevance.

The plant considered between all the possible applications to test the co-simulation setup is a controlled wind turbine. Specifically, the CART (Controls Advanced Research Turbine) has been selected.

Wind turbines are usually operated in different working regions (start-up, maximum power, constant power, shutdown, ...), and active controls are used to maximize power production, maintain safe operation and limiting structural (static and fatigue) loads in all the working conditions [14].

The multidisciplinary environment is used to analyze the coupling between the wind turbine dynamics and a simple baseline controller, managing the blade pitch and the electrical generator power production.

The CART being the object of this chapter, it will be briefly described in the next section.

## 5.1 CART Description

The CART is a two-bladed, upwind, variable speed wind turbine, rated at 600 KW. It is located at the National Wind Technology Center (NWTC) and was installed as a test-bench to design new control schemes in wind power generation. The rotor has a diameter of 43.3 m and hub height is 36.6 m. The electrical

Figure 5.1: The Control Advanced Research Turbine at NWTC, Colorado

generator is rated at 1800 rpm, and connected to the Low Speed Shaft through a two-state gearbox with a ratio of 43.165. The turbine rotor is thus rated at 41.7 rpm.

## 5.2 Multibody Model

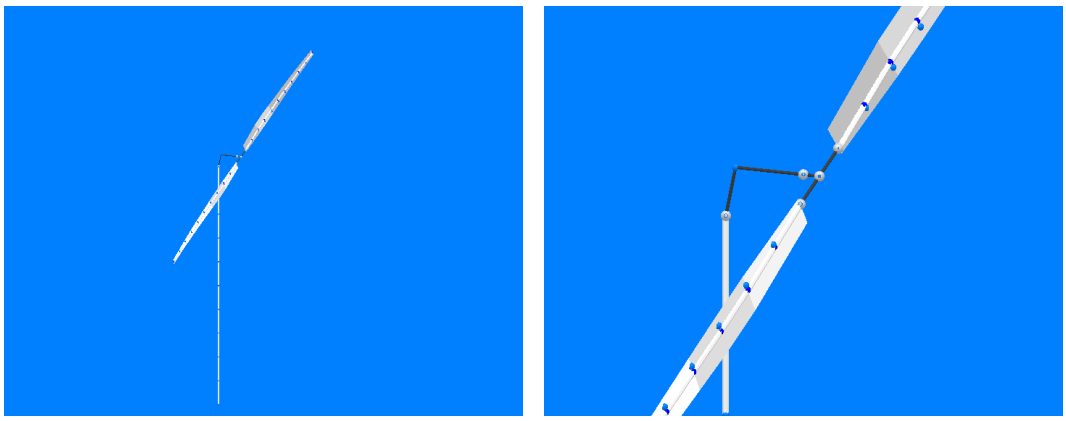
The computational model of the turbine is the same used in [15], freely available for download at [16] (under the name *cart*0) thank to the author, Luca Cavagna.

Deformable beam elements are used to model the tower and the blades structural behavior. Rigid elements model the nacelle, the low-speed shaft and the teetering hub, while the generator is idealized as a torque acting between the nacelle and the low-speed shaft. Aerodynamic elements are used to model the aerodynamical properties of the blades. Table 5.1 quickly describes the multibody model properties.

A graphical representation of the model is given in figure 5.2

| Components | Nodes | Joints | Bodies | Beams | Aero | Forces | DoFs |
|---|---|---|---|---|---|---|---|
| Tower | 11 | 1 | 10 | 5 | | | 138 |
| Nacelle | 1 | 2 | 1 | | | | 17 |
| Shaft | 1 | 1 | 1 | | | | 17 |
| Generator | | | | | | 1 | |
| Teeter | 1 | 1 | 1 | | | | 17 |
| Blade 2x | 11 | 1 | 11 | 5 | 5 | | 138 |
| Total | 36 | 7 | 35 | 15 | 10 | 1 | 465 |

Table 5.1: Summary of CART multibody model



(a) General view (b) Rotor to tower connection detail

Figure 5.2: Graphical representation of the CART multibody model

## 5.3 Controller

A simple example for a variable-speed wind turbine controller is given in [14]. Three working regions are defined:

**Startup** The rotor is starting, and no torque from the generator is applied

**Region 2** The rotor is below rated speed. The generator torque is controlled to get the optimal Tip Speed Ratio (TSR), thus the maximum power generation

**Transition** Transition region to connect Region 2 and Region 3, to let the turbine reach the rated torque at rated speed

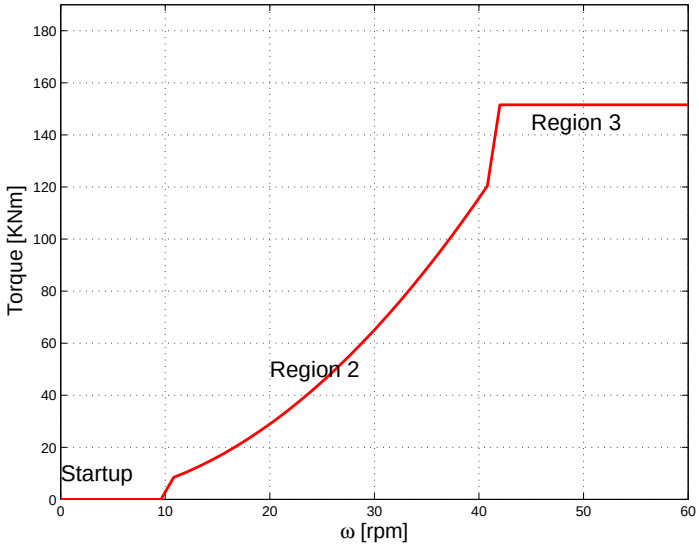



Figure 5.3: Electrical generator working function: the electrical generator torque is plotted as a function of turbine rotor speed

**Region 3** The rotor is above rated speed. The blade pitch is automatically controlled to maintain constant rotor speed, while the generator torque is held constant at rated torque

The turbine shutdown will not be considered. Even if the CART can handle a different pitch command for each blade, only the collective pitch control will be considered in this work.

During the startup, the blade pitch is regulated to maintain the maximum-lift angle of attack, thus giving the maximum acceleration to the rotor. Until a speed rotor of about 10 rpm, no electrical power is extracted, thus no torque is applied by the generator to the turbine rotor.
The Region 2 control model is quite simple to describe. To maintain the maximum power coefficient, the generator torque must be proportional to the square of the rotor speed: this condition is practically achieved controlling the generator internal resistance, and keeping the blade pitch at zero.
In Region 3, a PID controller, acting on the blade collective pitch, is activated to maintain the rotor at constant speed (the rated speed). The electrical generator piecewise working function is represented in figure 5.3

The complete layout of the controller is given in figures 5.4, 5.5 and 5.6. In the block scheme it is possible to recognize the blade pitch controller, the electric

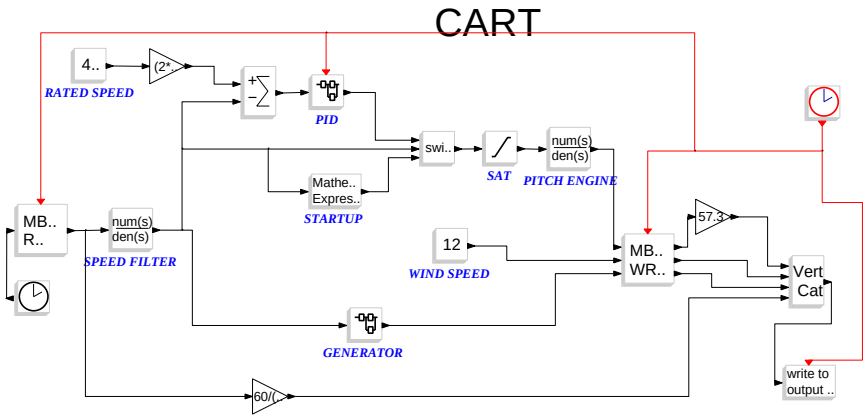



Figure 5.4: Block Scheme of the controller implemented in Scicos

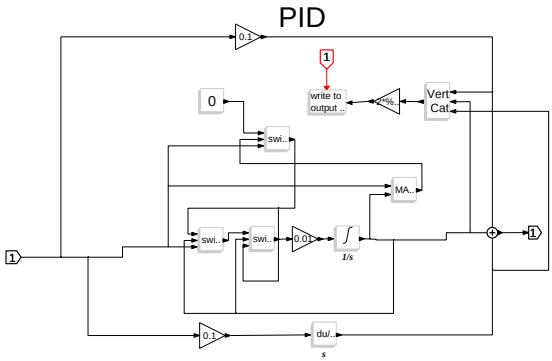



Figure 5.5: PID controller superblock

generator torque and the wind magnitude definitions.

The *PITCH ENGINE* block in figure 5.4 represents the blade pitch actuator transfer function, a second order Butterworth filter.

The *STARTUP* block acts in the first part of the simulation, to startup the turbine rotor. In this condition, the PID controller has to be bypassed because the blades need to point the airfoil nose into the wind. The PID controller, in normal conditions, basically increases the angle of attack (by incrementing the blade pitch) to increase the rotational speed. When the angle of attack is over the stall limit (this occurs during the startup) this condition has to be inverted. The *STARTUP* block takes care of this inversion.

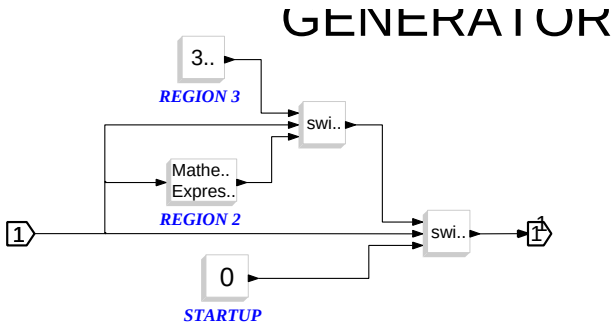



Figure 5.6: Electrical generator superblock

## 5.4 Wind Turbine General Behavior

It is important, before beginning with the design of the blade pitch controller, to analyze the general turbine behavior. For example, it must be determined how the available output power change versus the wind speed and the blade pitch angle. Figure 5.7 shows this result. It is possible to note that, until about a wind magnitude equal to $12\frac{m}{s}$, it is correct to work at maximum efficiency, in order to extract the maximum electrical power: in this region, the blade pitch must be kept constant at a value of $1deg$. As the wind velocity increases, extracting the maximum available power would mean exceeding the rated power, and this must be avoided. To this aim, it is obvious from figure 5.7 that the blade pitch must be controlled

## 5.5 Results and Costs

The results in terms of rotor speed and pitch command, for the given wind magnitude, are shown in figure 5.8. The action of the PID controller, in terms of proportional, derivative and integral components, is given in figure 5.9

From figure 5.8 it is possible to note how the $STARTUP$ block works: until the speed rotor does not reach a significant value, the blade pitch is saturated at the lower limit (for the first 20 seconds). Afterward, the maximum angle of attack is kept to accelerate the rotor (until about 35 seconds). Finally, the PID output is used to control the blade pitch and set the rotor speed at the rated value.

Results in terms of internal forces are shown in figures 5.10 and 5.11.

All the simulation ran on a single AMD Athlon 64 Processor 3000, whose

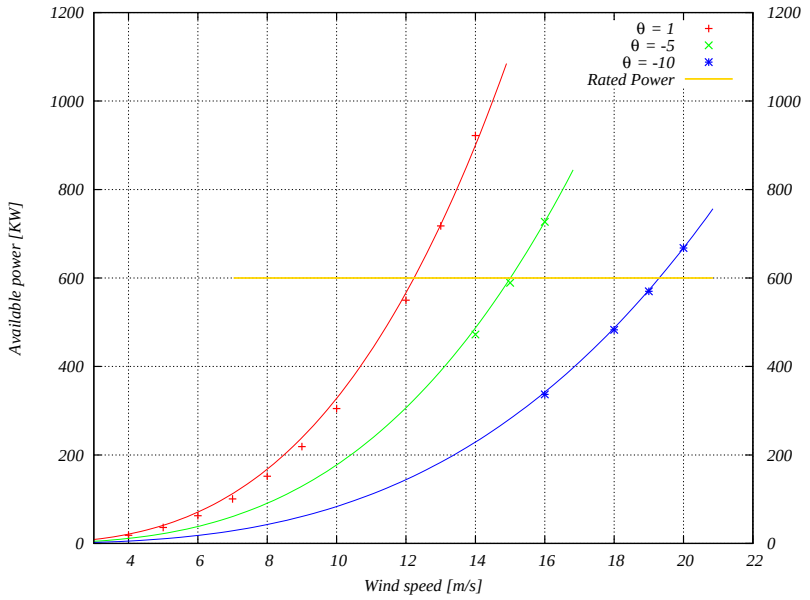



Figure 5.7: CART available power diagram: points represent data from simulations, continuous functions are the theoretical curves

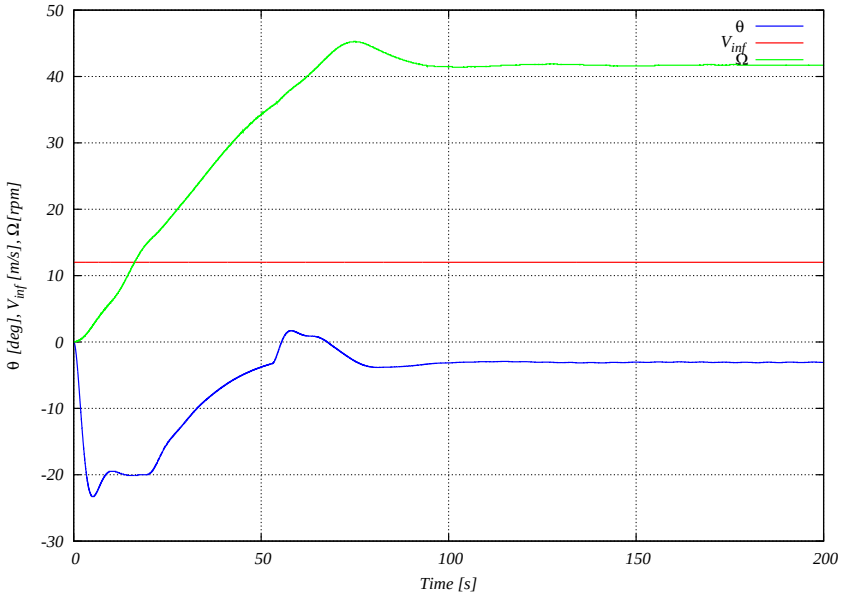



Figure 5.8: Blade pitch, wind magnitude and rotor speed as functions of time

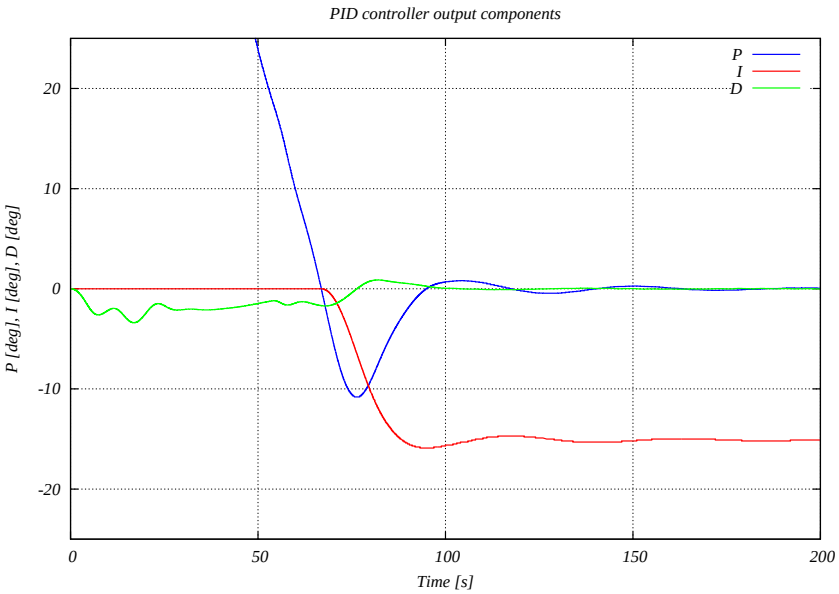



Figure 5.9: PID controller action in terms of components

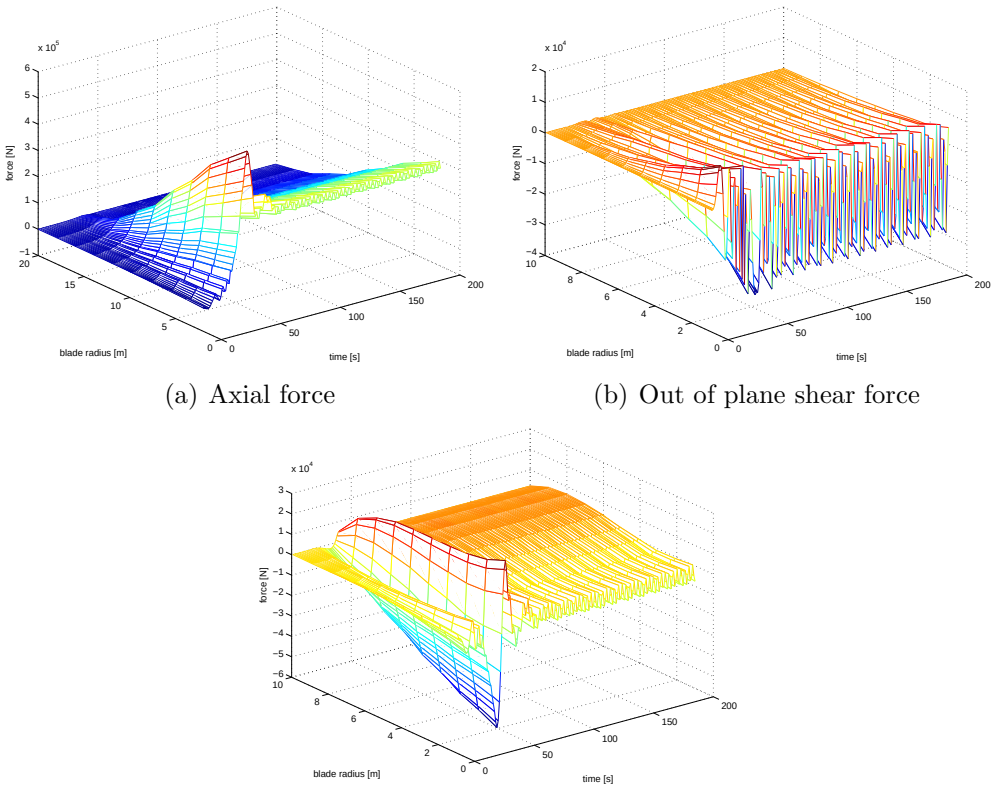
performances are given in table 5.2

| CPU MHz | 2002.5 |
|---|---|
| Cache Size | 512 KB |

Table 5.2: Processor performances

It is important to monitor the computational costs for the different simulation layouts. The working environment is made up of two different softwares: MBDyn and Scicos. The two components have big differences in terms of computational efficiency, with a great superiority for MBDyn. However, also the model described in the softwares are definitely different: MBDyn analyzes the nonlinear dynamics of a complex mechanical system, while Scicos have to model a quite simple controller, with few nonlinearity.
In conclusion, for the case presented here, the CPU time needed by MBDyn is much higher (20 times higher than the time needed by Scicos).

The overall CPU time required for the simulation in this case is $224330ms$ ($3min$, $44s$, $330ms$). For MBdyn alone to simulate the same model, without external interaction to add the control loop, $185990ms$ ($3min$, $05s$, $990ms$) are needed to complete the simulation.

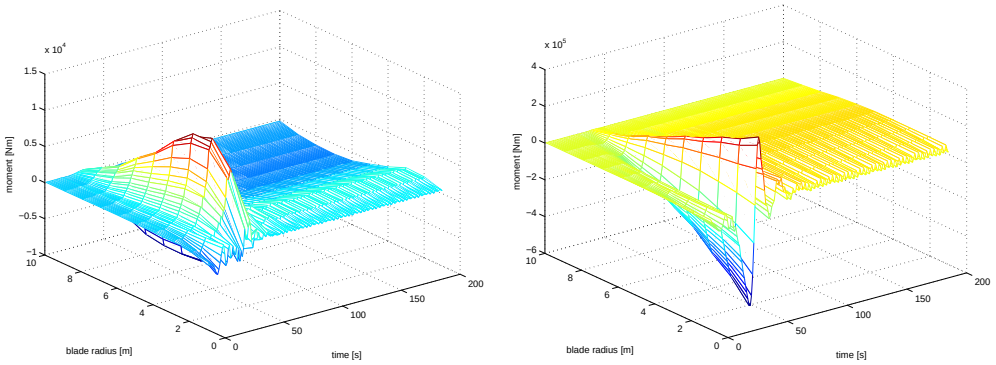



(a) Axial force

(b) Out of plane shear force

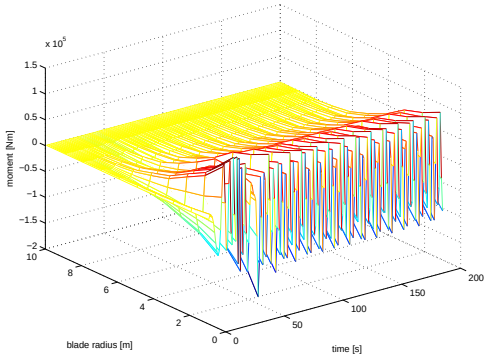
(c) In plane shear force

Figure 5.10: Time history of CART blades internal forces over blade radius




(a) Torsional moment

(b) In plane bending moment




(c) Out of plane bending moment

Figure 5.11: Time history of CART blades internal moments over blade radius

## 5.6 Multirate CPU Time Saving

An interesting situation is represented by the multirate layout, in which MBDyn and Scicos simulate their own models integrating the equations at different frequencies.

It is common to have situations in which subsystems have different frequencies. In these cases, it happens that if the integration frequency is determined based on the fast subsystem characteristics, a high computational cost is reached, otherwise, if the slow subsystem is considered, the accuracy for the fast subsystem is lowered. To save CPU time and accuracy, the multirate layout may be employed.

Considering the CART example, for instance, a detailed behavior of the electrical generator, to analyze the oscillations in the output current, requires a much smaller timestep, say $0.002s$, while the previous simulation, to analyze the turbine dynamics coupled with the pitch and torque controllers, has been carried with a timestep of $0.01s$.

The aim here is not to give a detailed analysis of the generator nor the turbine behavior, but rather to give an idea of what is the time saving due to using the multirate layout to simulate the machine behavior.
Therefore, the CPU time is determined relative to the simulation of the already presented system, both in multirate layout and in singlerate (now with a refined timestep).
The processor performances do not change, and the reference is again table 5.2.

The overall CPU time required for the singlerate simulation with a timestep of $0.002s$ is $934960ms$ ($15min$, $34s$, $960ms$).

It is obvious, by looking at the computational costs, that carrying such a simulation in a singlerate layout is not convenient at all, but one would prefer to extract the results from a simulation with a simpler model of the electrical generator, and then build another model, in which the detailed generator behavior is described, giving the previous results as inputs.

An integrated way is, instead, to describe the detailed generator directly in the general model, but then exploit the multirate layout. The simulation carried with a timestep of $0.01s$ in MBDyn and $0.002s$ in Scicos, in this multirate cosimulation environment, costs $227810ms$ ($3min$, $47s$, $810ms$) of CPU.

## 5.7 Wind Turbine Modeling Objective

Wind turbines are usually driven by automatic controllers for two reasons: ensure a smooth time evolution of loads and electrical power and reduce the power output (as well as rotor velocity) at high wind speed.

The second aim is directed in both reducing the costs of the plant and increasing the safety of operations. This corresponds to the reason why a blade pitch controller has been applied to the CART.

The electrical generator torque control is achieved by simply defining a functional relation between the rotational speed and the generator internal variable resistance. When this control is active, the blade pitch is kept constant at zero, and the rotor is operated at maximum efficiency. Due to its simplicity, this control scheme will not be analyzed further.

The blade pitch control, instead, is interesting in different situations, and the overall plant performances might heavily depend on this component.

It is of interest to investigate whether the layout proposed in this work is feasible or not. To this aim, the speed rotor is monitored during wind changes. A good blade pitch controller is capable to keep the rotor velocity as close as possible to the rated speed, possibly with minimal oscillations.

To check the performances of the controller, a variable wind speed situation is simulated. Specifically, steps are applied in wind magnitude, during the simulation. A wind magnitude of $12\frac{m}{s}$ is initially input in the model. After $120s$ a step of $1.5\frac{m}{s}$ is added to the wind speed, and after other $80s$, another step is added, equal to the first one. Results are shown in figure 5.12

The rotor rated speed is followed with little oscillations, and the transition time to damp out the speed error is less than 40 seconds, in a wind magnitude interval form $12\frac{m}{s}$ to $15\frac{m}{s}$, which is representative of the working conditions of the CART. It is thus demonstrated that the layout proposed in this chapter is a feasible application for a controlled wind turbine.

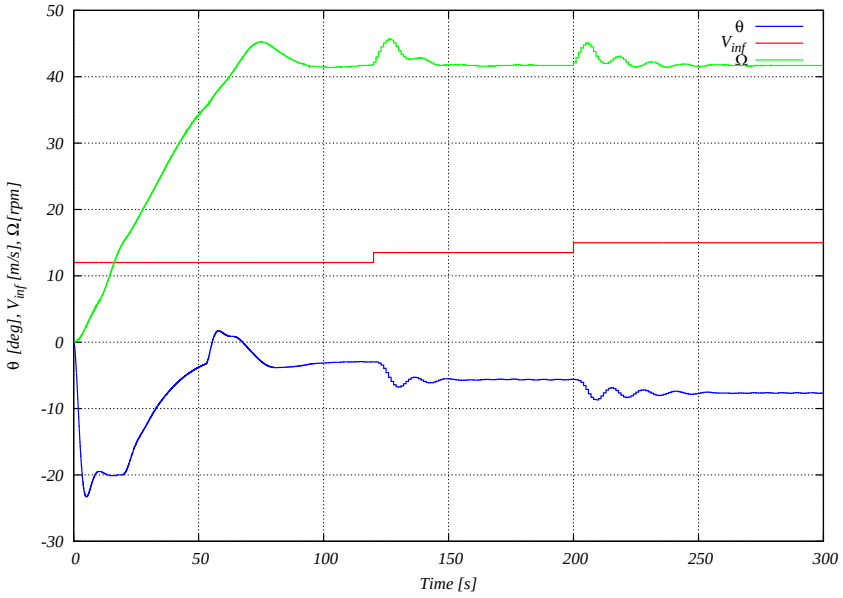



Figure 5.12: Results for steps in wind magnitude

# Chapter 6

# Helicopter Dynamics Application

It is common that complex mechanical systems show different time scales, as described in chapter 2. Helicopters represent one of the best examples. In these machines, different time scales are related to different dynamical subsystems, as summarized in table 6.1.

| Subsystem | Characteristic Frequency |
|---|---|
| Flight Dynamics | 1 Hz |
| Main Rotor Dynamics | 7 Hz |
| Tail rotor dynamics | 37 Hz |

Table 6.1: Typical helicopter subsystems frequencies

In this chapter, the model of a light twin-engine utility helicopter, namely the MBB Bo105 by *Bölkow* (figure 6.1), is presented.

A whole model of the helicopter has been built, in which the main and tail rotors are modeled together with part of the flight dynamics in two distinct multibody models, and a block scheme connects them together and implements a tail collective pitch controller.

The aim is to show the computational efficiency of the multirate layout in the simulation of the whole model, describing both slow and fast components together.


Figure 6.1: The MBB Bo 105

## 6.1 Bo 105 Description

The Bo105, shown in figure 6.1, is a small twin-engine multipurpose helicopter built by MBB. It is a highly maneuverable and relatively small helicopter, with an empty weight of about 1200kg and a maximum gross weight of 2300kg.

This machine has a four-bladed hingeless main rotor, with a large equivalent hinge offset, thus showing high bandwidth and maneuverability. For such a machine, dynamic coupling between different components is a very important issue, since the preliminary studies related to the design. Also, the coupling with automatic controls (stability augmentation and autopilot) needs to be evaluated by means of accurate models.

The tail is controlled by a two-bladed teetering rotor, which works as a pusher, on the left side of the helicopter.

The main rotor has a nominal speed of 44.4 rad/s (7 Hz), and the tail rotor is rated at 233 rad/s (37 Hz). The three-views shown in figure 6.2 gives the impression of the helicopter main characteristics.

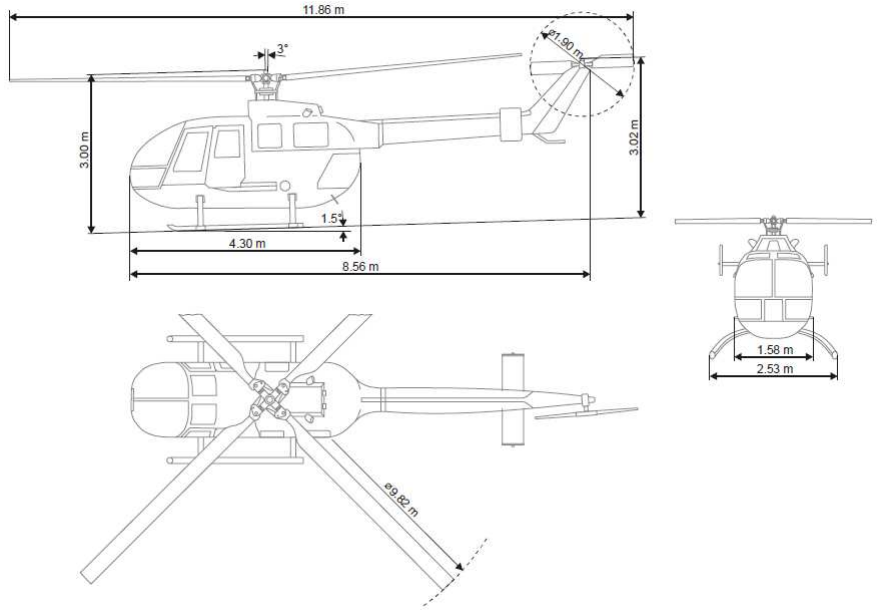



Figure 6.2: The MBB Bo 105 three-view sketch

## 6.2 The Multibody Model

The whole mechanical system is divided into two different models. The first one describes the elements related to the main rotor and the fuselage, while the other contains the definition of the tail rotor.

The two models can be used independently. The main rotor model can be used to simulate the relatively slow components of the dynamic of the machine, approximating the tail rotor effect as a slow varying force concentrated at tail tip, to stabilize and control the trajectory of the helicopter. The tail rotor model, instead, is used to catch the relatively high frequencies effects (for example load oscillations on the tail tip). If accurate modeling needs to be done, it is desirable to couple the two models together.

If traditional methods are used to build the coupling in a monolithic setup, high computational costs are reached, due to the complexity of the main rotor model and high frequency of the tail rotor. With the architecture presented in this work, not only automatic controller behavior is easily implemented, but also computational efficiency is enhanced by the multirate methods. The coupling of this two submodels is an example to show these benefits.

### 6.2.1 Main Rotor and Fuselage

The first multibody model contains the description of the fuselage and the main rotor components.

The fuselage is described by a modal element, which is a single element containing the modes (in this case the first four) of the body. This element is connected to different nodes, in order to give the relative displacements between them during the simulation. In this model, the main modal node is at the center of gravity of the fuselage, and links are defined to the main rotor attachment node, to tail rotor attachment node, pilot node and copilot node. For each blade, four beam elements are used to model the structural deformability and four aerodynamic elements linked to the structural nodes generate the aerodynamic forces. Rigid elements are used to model the main rotor hub and other secondary components.

Aerodynamic forces are applied to the fuselage, the vertical stabilizer and the horizontal stabilizer.

The main airframe node (the node located at the center of gravity) has only two degrees of freedom, which allow only the vertical translation and the yaw rotation of the fuselage. The other four degrees of freedom are excluded by infinitely rigid constraints. Vertical translation is maintained constant by the action of an autopilot, which controls the main rotor blades pitch.

The model is described by a total of 689 equations.

### 6.2.2 Tail Rotor

The tail rotor submodel is composed by two rigid element representing the blades, each of those is linked to an aerodynamic element generating lift, drag, and defining the tail rotor induced velocity. Different joints define the fuselage tail tip connection, the rotor hub and the teetering hub. A distance constraint between a blade-connected point and a tail-connected point is used to define the collective pitch.

The model is defined by a total of 77 equations.

## 6.3 Submodels Connection and Control

The software architecture presented in this thesis is used for both building the communication between the main rotor and tail rotor submodels and defining

the automatic control of the tail collective pitch, used to set the yaw rate of the machine.

The figure 6.3 shows the Scicos block scheme used to build the main simulation model.

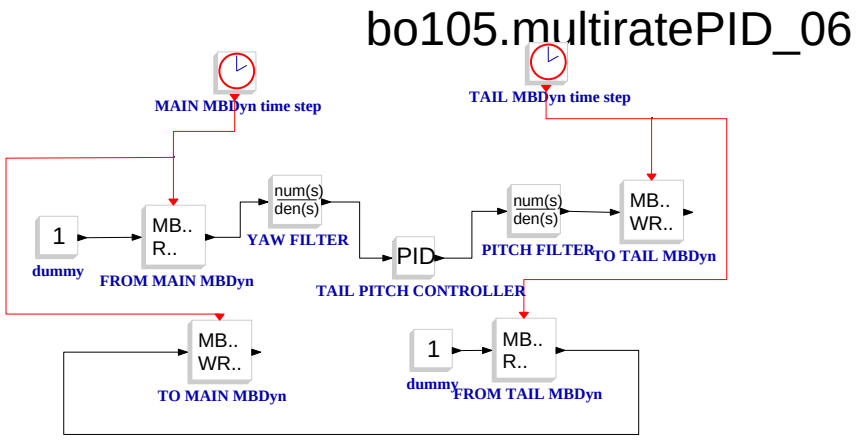



Figure 6.3: Scicos scheme for the submodels integration

The red clocks in the scheme of figure 6.3 control the time steps at which each submodel is integrated. Defining different values for the two clocks will tell the software to build the multirate setup.

The main rotor MBDyn model outputs the tail tip displacements and rotations and the yaw angle, and asks the tail reaction force as input. The tail rotor model needs the tail tip displacements as input as well as the pitch command, and outputs the reaction force to be applied on the fuselage.

## 6.4 Results and Costs

The tail collective pitch control is tested in a hovering condition. The aim here is not to design a sophisticated and robust control for all the possible operating conditions and configurations of the machine, but to show that the software architecture used is a feasible solution to simulate the dynamical behavior of the system considered, and it may be used to design or check the performances in similar contexts.

It is important to show the benefits of using multirate co-simulations in the case the two subsystems are simulated together to predict the motion of the

whole system by means of a single simulation.

Three different layouts are presented:

**Case 1** MBDyn is working alone without Scicos, and the two submodels interact in a single-rate manner. The time step to integrate both subsystems is equal to $2.2\,10^{-4}s$.

**Case 2** Scicos is responsible of submodels interaction, as explained in section 6.3. A single rate technique is used, with the same time step of case one.

**Case 3** in this case Scicos is used to define the interaction, but a multirate scheme is exploited: for the tail rotor model the same time step of case one and two is used, but the main rotor dynamic is computed with a time step of $1.1\,10^{-3}s$.

Case one is only used as a reference for computational costs, while results computed in cases two and three are compared.

Results in terms of altitude, yaw angle and tail rotor pitch command are shown in figure 6.4, 6.5 and 6.6, respectively. It is possible to note that the output considered do not differ: the relative error between the single-rate and the multirate output is always less than 0.3%.

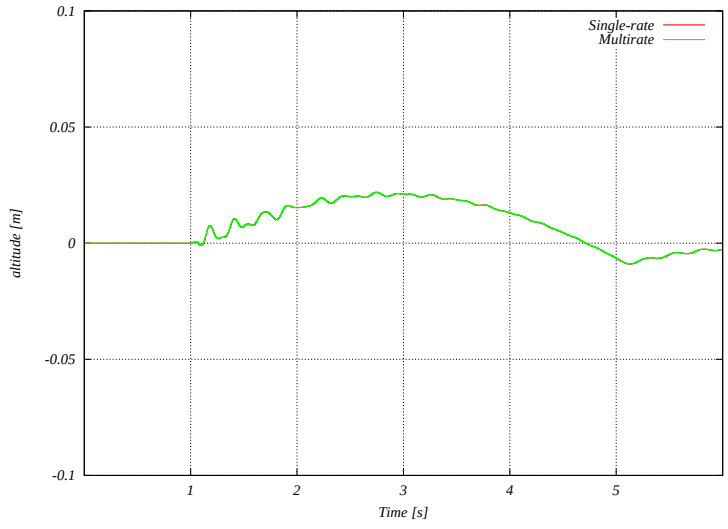



Figure 6.4: Helicopter relative altitude during the simulation

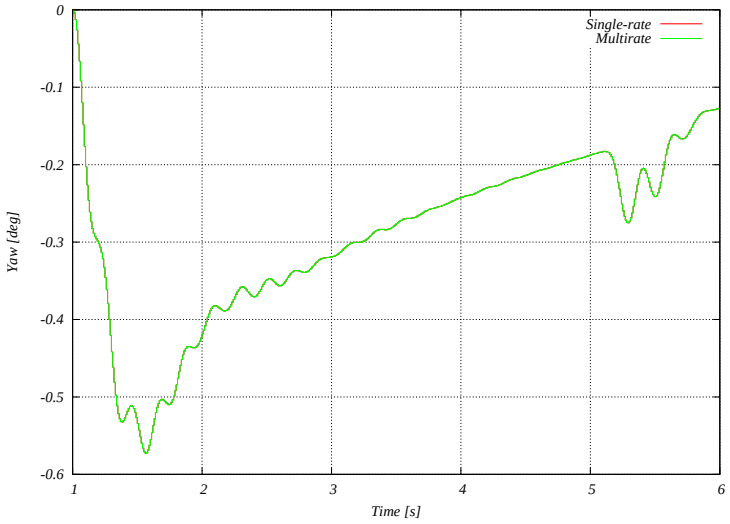



Figure 6.5: Helicopter yaw angle as function of time

The signal suffering the most for time step reduction is the tail rotor reaction force, shown in figure 6.7. It is easy to note that this value has a quite high frequency component, due to the rotational speed of the tail rotor. These oscillations have a frequency of about 75 Hz, two times larger than the tail rotor angular velocity, (being the tail rotor two-bladed). This issue does not lead to inaccurate results because the high frequencies applied on tail tip do not pass through the helicopter structure.

The computational costs, always based on simulations with the processor of table 5.2, are summarized in table 6.2, for a simulated time interval of 6 seconds.

| Case | Main rotor simulation cost [s] | Tail rotor simulation cost [s] |
|---|---|---|
| 1 | 299 | 57 |
| 2 | 340 | 55 |
| 3 | 62 | 42 |

Table 6.2: Computational costs for the different layouts

Table 6.2 demonstrate that the multirate layout leads to a huge computational saving. Not only the slow subsystem costs are reduced, but also the fast

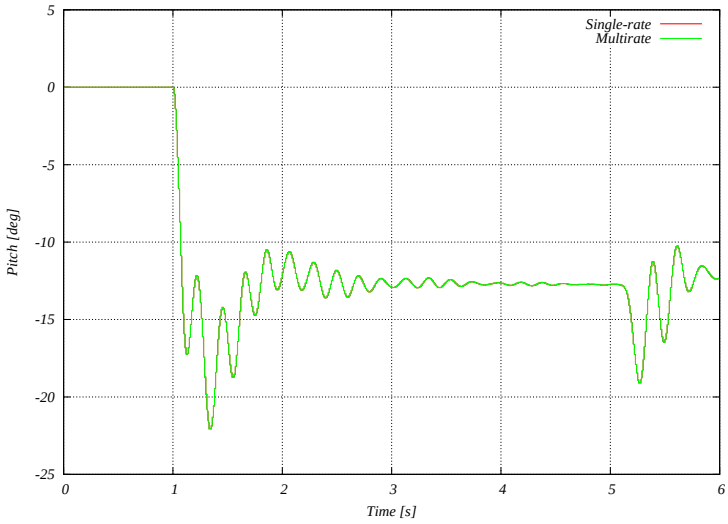



Figure 6.6: Controller action: tail rotor collective pitch as function of time

subsystem need a lower amount of CPU time, due to the fact that communication are less frequent in the case of the multirate technique.

The advantages shown in this section would further increase if many configurations and design conditions have to be analyzed, as it is the case, for example, of a helicopter or airplane design procedure.

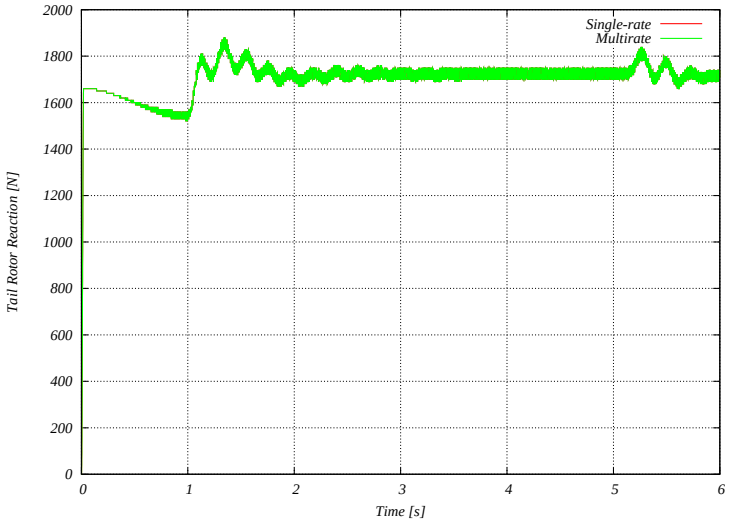



Figure 6.7: Reaction force at the interface of tail rotor and fuselage

# Chapter 7

# Conclusions

In this thesis multirate co-simulations have been studied. Performances, in terms of stability and accuracy, have been shown for different numerical methods which can be exploited in co-simulation setups.

A new numerical method has been defined, that should improve the computational efficiency, especially in parallel simulations. For this method also, stability and accuracy results have been shown, and it possible to conclude that it exhibits properties similar to the traditional methods.

Only based on free software tools, the co-simulation environment has been implemented and tested on a real application.

In conclusion, it has been shown that the multirate co-simulation setup presented in this work show excellent results. Benefits are obtained in both the model definition procedure and in computational efficiency, two important issues related to the simulation of mechanical systems dynamic.

# Bibliography


[1] W. Schiehlen: *Research Trend in Multibody System Dynamics*, Multibody System Dynamics, **18**, 3-13, (2007)

[2] F. Gonzales: *Efficient Implementations and Co-simulation Techniques in Multibody System Dynamics*, PhD thesis, (Ferrol, March 2010)

[3] M. Busch, M. Arnold, A. Heckmann, and S. Dronka: *Interfacing SIMPACK to Modelica/Dymola for multidomain vehicle system simulations*, SIMPACK News, 11, **2**, 1-3, (2007)

[4] R. Kubler and W. Schiehlen: *Modular simulation in multibody system dynamics*, Multibody System Dynamics, **4**, 107127, (2000)

[5] C. W. Gear and D. R. Wells: *Multirate Linear Multistep Method*, BIT, **24**, (1984), 484-502.

[6] S. Skelboe: *Stability Properties of Backward Differentiation Multirate Formulas*, Applied Numerical Math., **5**, 151-160, (1989)

[7] A. Verhoeven and E.J.W. Maten: *Stability Analysis of the BDF Slowest first Multirate Methods*, International Journal of Computer Mathematics, **84**(6), 895-923, (2007)

[8] A. Verhoeven, A. El Guennouni, E.J.W. Maten and R.M.M. Mattheij: *A General Compound Multirate Method for Circuit Simulation problems*, Scientific Computing in Electrical Engineering, ECMI, Vol. 9, 143-150, (2006)

[9] F. Gonzales, M. Gonzales and J. Cuadrado: *Weak Coupling of Multibody Dynamics and Block Diagram Simulation Tools*, Proceedings of the ASME IDETC/CIE 2009, (September 2009, San Diego, California, USA), DETC2009-86653

[10] G. Rodriguez: *Absolute Stability Analysis of Semi-Implicit Multirate Linear Multistep Methods*, (June 2002, Tonanzintla, Puebla)

[11] Quarteroni, Sacco, Saleri: *Matematica Numerica*, Springer Italia, (2008)

[12] Masarati P, Lanz M, Mantegazza P: *Multistep integration of ordinary, stiff and differential-algebraic problems for multibody dynamics applications*, In XVI Congresso Nazionale AIDAA, pages 71.110, Palermo, September 2428 (2001)

[13] K.A. Stol: *Geometry and Structural Properties for the Controls Advanced Research Turbine (CART) from Model Tuning*, National Renewable Energy Laboratory, NREL/SR-500-32087, August 25, 2003 November 30, 2003

[14] A. D. Wright and L. J. Fingersh: *Advanced Control Design for Wind Turbines*, NREL/TP-500-42437, (March 2008).

[15] L. Cavagna, A. Fumagalli, P. Masarati, M. Morandini and P. Mantegazza *Real-Time Aeroservoelastic Analysis of Wind-Turbines by Free Multibody Software*

[16] www.aero.polimi.it/mbdyn/documentation/examples/

[17] *MBDyn Input File Format.*

[18] www.gnu.org/licenses/gpl.html

[19] www.fsf.org/about/

[20] www.aero.polimi.it/ mbdyn

[21] www.mathworks.com/products/simulink/

[22] www.ni.com/matrixx/systembuild.htm

[23] www.scicos.org/

[24] www.scicoslab.org/

[25] www.scilab.org/products/xcos