Multibody 3D Simulation Tool for Vehicle Development

Tiago Miguel Serralha Pinheiro de Carvalho

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisor(s): Prof. Luís Alberto Gonçalves de Sousa
Prof. João Manuel Pereira Dias

Examination Committee

Chairperson: Prof. Afzal Suleman
Supervisor: Prof. João Manuel Pereira Dias
Member of the Committee: Prof. Ricardo José Fontes Portal

October 2022

Declaration

I declare that this document is an original work of my own authorship and that it fulfils all the requirements of the Code of Conduct and Good Practices of Universidade de Lisboa.

Declaração

Declaro que o presente documento é um trabalho original da minha autoria e que cumpre todos os requisitos do Código de Conduta e Boas Práticas da Universidade de Lisboa.

Resumo

O presente trabalho visa explorar os conceitos de sistemas multicorpo e o trabalho desenvolvido por Nikravesh, De Jalon e Bayo, para desenvolver uma ferramenta de aplicação geral para permitir à equipa de formula student do IST, FST Lisboa, incorporar a dinâmica de multicorpo no seu projeto de geometria da suspensão. Para atingir este objetivo, foi desenvolvido um programa cinemático 3D, com uma interface em *Excel*, no qual as restrições cinemáticas básicas e os princípios algébricos foram formulados e modificados para utilizar os parâmetros de Euler e para assegurar que o código funciona sem singularidades. Após o desenvolvimento, o programa cinemático foi comparado com o software *MSC Adams*, que foi então utilizado como base para desenvolver um programa mais complexo de dinâmica direta 3D. Este programa foi desenvolvido utilizando um integrador preditor-corretor, Adams-Bashforth-Moulton, em combinação com a Formulação Lagrangiana Aumentada de indice 1 que depois foi verificado com recurso ao sistema de benchmarking desenvolvido por González et al. e o software MSC Adams. O programa desenvolvido incorpora elementos-chave para a simulação de modelos de suspensão de veículos, tais como elementos de força linear e não linear. Além disso, o processo de comparação com o *MSC Adams* provou que o programa em si é fiável e capaz de executar os cinco problemas propostos para testar características específicas de sistemas de corpos múltiplos. Consequentemente, estes resultados permitiram a identificação das limitações do programa e a definição de orientações futuras para o seu desenvolvimento.

Palavras-chave: Análise Cinemática 3D, Análise Dinâmica 3D, Sistemas de Corpos Múltiplos, Análise comparativa.

Abstract

The present work aims to explore the concepts of multibody systems and the work completed by Nikravesh, De Jalon e Bayo, to develop a general-purposed tool to allow the formula student team of IST, FST Lisboa, to incorporate multibody dynamics into its suspension geometry design. To reach this objective a 3D kinematic program, with an *Excel* interface, was developed in which the basic kinematic constraints and algebraic principles were formulated and modified to use Euler parameters and to ensure that the code runs free of singularities. After the development, the kinematic program was compared with the *MSC Adams* software which was then used as a base to develop a more complex 3D forward dynamics program. This program was developed using a predictor-corrector integrator, Adams-Bashforth-Moulton, in combination with an index-1 Augmented Lagrangian Formulation which then was verified with recourse to the benchmarking system developed by González et al. and the *MSC Adams* software. The program developed incorporates key elements for the simulation of vehicle suspension models, such as linear and non-linear force elements. Additionally, the benchmarking work proved that the program itself is reliable and capable of running all five benchmarking problems developed to test specific multibody systems characteristics. Consequently, these results allowed for the identification of the program's limitations and the definition of future guidelines for its development.

Keywords: Multibody Systems, 3D Kinematics, 3D Forward Dynamics, Benchmarking.

Contents

List of Tables

List of Figures

List of Algorithms

List of Acronyms

2D Two Dimensional

3D Three Dimensional

ALF Augmented Lagrangian Formula

CAD Computer-aided Design

CAE Computer-Aided Engineering

CoM Centre of Mass

DAE Differential-Algebraic Equation

DOF Degrees of Freedom

FFT Fast-Fourier Transformation

GSTIFF Gear Stiff Integrator

I3 Index-3 Formulation

LU Lower-Upper Factorisation

MBD Multibody Dynamics

MBS Multibody System

ODE Ordinary Differential Equation

SI2 Stabilized Index-2 Formulation

TSVD Truncated Single Value Decomposition

VSVO Variable-Step, Variable-Order

WSTIFF Wielenga Stiff Integrator

Nomenclature

Greek symbols

α	Convergence rate penalty parameter.
β	ALF penalty system.
μ	Damping frequency penalty parameter.
Ω	Natural frequency penalty parameter.
ω	Cartesian angular velocity.
Φ	Position constraint vector.

Roman symbols

A	Euler parameter transformation matrix.
C	Euler parameter component of the Jacobian matrix .
c	Damper damping coefficient.
d	Global variable length vector.
E	Mechanical energy.
e	Orientational rotation axis direction vector.
f	Force element force vector.
g	ALF unconstrained force vector.
J	Inertia tensor.
k	Spring stiffness coefficient.
l	Variable lenght vector for force elements.
M	Mass matrix.
n	Moment vector.
p	Euler parameter vectors.

q	Generalised coordinate vector.
r	Global position vector.
s	Vector defined in relation to the body centre of mass.

Subscripts

0	Initial value.
d	Dependent variable.
i	Defined in relation to body i.
id	Independent variable.
j	Defined in relation to body j.
q	Jacobian matrix.
t	First time derivative.
tt	Second time derivative.

Superscripts

$'$	Defined in relation to the local frame coordinate.
(d)	Driving component.
0	Undeformed length.
	Acceleration vector.
	Velocity vector.
c	Corrected values.
da	Damper force.
e	Estimated values.
P	Vector of point P.
s	Spring force.
u	Uncorrected values.

Chapter 1

Introduction

Computer-Aided Engineering (CAE) is the use of computer software to improve product design, including simulation, optimisation and validation of products and processes. With the increase of the available computer power since the late 80s, this broad field of study has become standard in most industries. One of the most frequently used CAE simulations analysis is the study of the multibody dynamics of rigid bodies, which is the focus of this thesis.

In its essence, Multibody Dynamics (MBD) is the study of the dynamic behaviour of a system of rigid or flexible bodies and their interconnections, a Multibody System (MBS), caused by external forces or motion excitations acting on said system [1].

1.1 Motivation

The formula student project is unique as it creates the opportunity for students to work like real engineers, and, within the safety rules, allows the application of several new ideas implying a constant search for innovation.

FST Lisboa is a well-established team in the formula student world. However, its prototypes usually do not have the best dynamic performance and weight above 230 kg, while top teams achieve cars below the 170 kg mark. Contributing to such a lack of performance is the team's approach to the design of its prototypes, prioritising reliability above performance parameters and opting to use conservative designs and safety factors. Additionally, the team's vehicle dynamics department does not possess a tool that enables its process design of the suspension geometry to be time effective and performance-focused since it is divided among various tools that are not compatible and entirely reliable or properly verified. These tools are the *perfectKinematics*, a multibody kinematic simulator capable of performing position analysis only, *carDynamics*, capable of calculating the mass transfer of the car in several scenarios and *SusForce* which intends to provide the force distribution throughout the suspension geometry.

In that context, the need for a general-purpose multibody kinematics and dynamic simulation program that allows studying the behaviour of the system throughout its different operational scenarios such as braking and cornering, cornering and acceleration, pure acceleration, pure braking and pure cornering becomes essential.

Currently, several commercial programs, such as *MSC Adams* [2], *OptimumG* [3] and *Simulia* [4] can meet the needs of the team as exposed by Schiehlen [5]. However, these programs present a list of disadvantages that prevent an easy introduction of them in FST Lisboa. Firstly, none of these programs are taught in the courses of mechanical or aerospace engineering at the University of Lisbon, which generates difficulties in improving the complexity of the team's models since every year new and inexperienced, students must learn the basics of each program and then improve their models. Secondly, the price tag in most of these programs is higher than what the team can afford, or its sponsored version is limited and does not allow the creation of complex models such as a formula student prototype. Additionally, these are closed source programs which make impossible the modification of the code for unique systems that may be present in the prototype or post-processing routines that can be performed using the results acquired from its usage. Conversely, *MATLAB*, *Excel* and *Solidworks* are programs that the team members are accustomed to due to their use during their academic path and, consequently, are more viable for the intended application.

1.2 Objectives and Deliverables

The main objective of this thesis is to use *MATLAB*, *Excel* and *Solidworks* to develop a general-purposed kinematic and dynamic multibody simulator. The operation of the program must be intuitive to the user and must combine the strengths of each of these programs to simplify the design process of the suspension geometry. Furthermore the development of the program must allow for future implementations of subroutines related to the vehicle dynamic context and numerical improvements without requiring a program overhaul.

The second inherent objective is to benchmark the developed code against a set of proposed benchmarking problems of MBS Simulation tools, to identify program limitations and define future guidelines for its development.

Finally, a model of a total or quarter suspension, should be developed and simulated with recourse to the program, to ensure that it can satisfy its main goal of serving the team's needs during its design process.

1.3 Thesis Outline

Following this introductory note, Chapter 2 addresses the concepts and theory needed to develop the code that is implemented on the simulator, including joint equations, non-linear solvers, integration algorithms and stabilisation of constraint violations in the forward dynamics simulation.

Chapter 3 exposes the code structure, its development methodology and the algorithm choices performed to ensure its reliability and its general-purpose application. Subsequently, Chapter 4 presents the benchmarking multibody models, the obtained results and discusses the limitations of the simulator. Chapter 5 states the conclusions drawn from this work and discusses the future implementations that the program may suffer and what fields of study should be integrated into its subroutines.

Chapter 2

Theoretical Background

This chapter addresses the multibody theory used to develop the forward kinematic and forward dynamic analysis tool which comprises topics on the theory of vectors, space body orientation, kinematic constraint equations and multibody systems kinematic and dynamic problems overview.

2.1 Theory of Vectors and Matrices

As stated by Nikravesh [6] and Jazar [7], the mathematical foundation of the analysis of kinematic and dynamic motions is composed of vector algebra which is used to represent most of its physical quantities. Therefore, operations such as vector addition, multiplication and differentiation are essential to the development of the tool proposed in Chapter 1.

In its geometric form vector algebra is not suited for computer implementation henceforth this document will adopt a systematic matrix formulation of vector algebra, Eqs. (2.1) and (2.2), known as algebraic vector representation.

$$A \equiv [a_{ij}] \equiv \begin{bmatrix} a_{11} & ... & a_{1n} \\ a_{21} & ... & a_{2n} \\ ... & ... & ... \\ a_{m1} & ... & a_{mn} \end{bmatrix} \tag{2.1}$$

$$a = \begin{bmatrix} a_{(x)} \\ a_{(y)} \\ a_{(z)} \end{bmatrix} = \begin{bmatrix} a_{(x)} & a_{(y)} & a_{(z)} \end{bmatrix}^T \tag{2.2}$$

2.1.1 Algebraic Vector Operations

Due to the triviality of the algebraic operations and its wide availability in the different books that address multibody dynamics theory, the only important element that accrues from the algebraic vector representation that must be referenced in this thesis is the skew-symmetric matrix. The skew-symmetric matrix allows converting a vector into a corresponding matrix. For the case of spacial multibody dynamics, the 3×3, Eq.(2.3), and 4×4, Eq.(2.4), skew-symmetric matrices are used.

$$\tilde{a} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \tag{2.3}$$

$$\bar{a} = \begin{bmatrix} 0 & -a_1 & -a_2 & -a_3 \\ a_1 & 0 & a_3 & -a_2 \\ a_2 & -a_3 & 0 & a_1 \\ a_3 & a_2 & -a_1 & 0 \end{bmatrix} \tag{2.4}$$

The introduction of the skew-symmetric matrix concept allows the simplification of the cross-product operation which in turn will allow determining with more ease the derivatives of the joint position vectors used to elaborate the right-hand side equations of the acceleration analysis problem. Tab. 2.1 compares the geometric forms with their algebraic counterparts and illustrates how the skew-symmetric concept influences them.

Geometric	Algebraic
$\vec{a}$	a
$\vec{a} + \vec{b}$	$a + b$
$\alpha\vec{a}$	αa
$\vec{a} \cdot \vec{b}$	$a^T b$
$\vec{a} \times \vec{b}$	$\tilde{a}b$
$\vec{a} \cdot (\vec{b} \times \vec{b})$	$a^T \tilde{b}c$
$(\vec{b} \times \vec{c}) \cdot \vec{a}$	$(\tilde{b}bc)^T a$
$\vec{a} \times (\vec{b} \times \vec{c})$	$\tilde{a}\tilde{b}c$
$(\vec{a} \times \vec{b}) \times \vec{c}$	$\widetilde{\tilde{a}b}c$

Table 2.1: Vector terms in geometric and algebraic forms [6].

2.2 Spacial Multibody Theory

A multibody system in its essence is composed of a set of rigid or flexible bodies that are interconnected by kinematic joints that constrain their relative motion and force elements, which can be springs, dampers, actuators, or active elements such as an electric motor. A body is assumed to be rigid if its deformation can be neglected and thus have no impact on the system motion and in the dynamic analysis [8, 9].

Fig. 2.1 illustrates a generic multibody system and its composition. In this thesis only rigid bodies and ideal joints are considered, and non-standard joints and interactions are specifically developed for specific simulation scenarios.

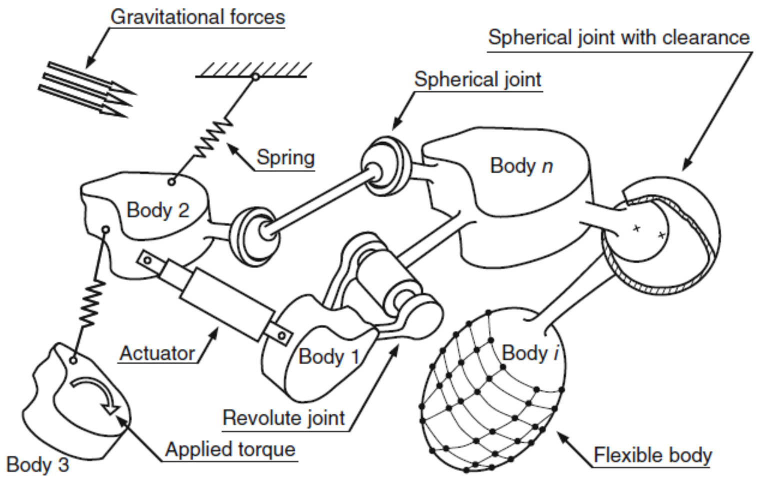


Figure 2.1: Representation of a generic multibody system with its most significant components [10].

2.2.1 Generalised Coordinates and Degrees of Freedom

Before proceeding with the fundamentals to characterise the body and joint locations and orientations, it is necessary to introduce the concepts that allow the user to characterise the configuration of the system. To describe the objects independently of the choice of coordinates, Cartesians or cylindrical, generalised coordinates are used. Generalised coordinates make no difference between positions and angles and if the generalised velocities can be defined, then it is possible to characterise the behaviour of the body in the future.

In space, to define the position and orientation of a rigid body, it is necessary to define three components of translation and three components of rotation, corresponding to six generalised coordinates, which implies that to define the configuration of N unconstrained bodies it is necessary to define 6*N components, that are known as Degrees of Freedoms (DOFs). Each unconstrained body will have then a total of 6 DOFs. However, if constraints are applied it becomes possible to fully define the system motion by defining a minimum number of independent variables. Strictly speaking, DOFs define the number of directions that a body can move, thus fully characterising its configuration and allowing the user to easily perform the debug of a complex system model [6, 10].

To find the number of independent variables, DOFs, that a complex system has, it is possible to implement the $Chebychev - Grübler - Kutzbach$ criterion and its mobility formula, presented in Eq.(2.5). Where n is the number of bodies or moving links of the system, j is the number of joints and f_i is the number of DOFs constrained by each joint [11].

$$M = 6n - \sum_{i=1}^{j}(6 - f_i) \tag{2.5}$$

2.2.2 Local and Global Coordinate Systems

In accordance with Shabana [9], it is possible to state that a multibody system configuration is described by using measurable vector quantities, that demand the usage of an appropriate system of coordinates. In this case two types of coordinate systems are required, the global or inertial and the local or body coordinate systems.

In multibody systems, each body has an associated local coordinate system, which translates and rotates with its respective body during the system motion. Thus, the proper definition of the local coordinate system is important, since its own position and orientation can be used to define the position and orientation of its associated body. In this thesis, each local coordinate system is defined so that each body's Centre of Mass (CoM) corresponds to its origin, and its axes are coincident with the principal axes of inertia of the corresponding body.

Since it is also assumed that all the bodies are rigid, it is possible to state that the distance between the local frame origin and any point located in the body remains constant during the correspondent body motion. Thus, the position of a body and its attached joints or force elements can be fully characterised by defining a set of two vectors, as can be seen in Fig. 2.2. The vector $\vec{r}_i$, which is defined about the inertial frame and locates the position of the CoM of the body, and the vector $\vec{s_i^P}$ which is defined about the body CoM and represents the position of the joint or force element in that body.

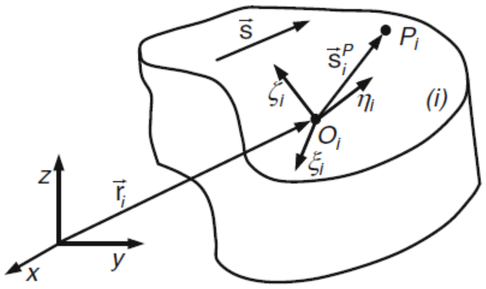


Figure 2.2: Spacial location of an unconstrained body i [10].

In addition to defining the local frame position, it is necessary to also define its orientation. Considering that for any simulation the initial predisposition of the bodies is given, or it is possible to be retrieved with recourse to a Computer-aided Design (CAD) model, it is usual to use the initial orientation of the bodies to define the initial local frame orientation.

Therefore, introducing the methods that use the initial orientation of the bodies to define the initial local frame orientation is fundamental. The first method requires the initial definition of three coplanar points located in two of the body principal's axes of inertia, that are used to define the origin and two of the axes of the local frame. The third axis is then determined by applying the cross product to the already known axes.

The unit vectors of these three axes, whose components are known as direction cosines, can be arranged in a matrix in accordance with Eq.(2.6) to form the rotation matrix of the local frame with respect to the inertial frame. In turn, this rotation matrix can be used to transform the local vector $\vec{s_i'^P}$ into the global vector $\vec{s}_i^P$, as seen in Eqs (2.7) and (2.8). In other words, the rotation matrix introduces the general transformation from local coordinates to global coordinates [6].

$$A_i = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, u_{\xi i} = \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \end{bmatrix}, u_{\eta i} = \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \end{bmatrix}, u_{\zeta i} = \begin{bmatrix} a_{13} \\ a_{23} \\ a_{33} \end{bmatrix} \tag{2.6}$$

$$r_i^P = r_i + s_i^P \tag{2.7}$$

$$s_i^P = A_i s_i'^P \tag{2.8}$$

Euler Angles and Euler Parameters

However, the first method is not always the most practical and it is possible to use other methods to define the transformation matrix in terms of more suitable sets of coordinates. The Euler angles provide a set of three independent angles that represent a sequence of three successive rotations that can be employed in twelve different sequences and fully define the orientation of the body. Since the Euler angles represent these three rotations, it is possible to state that the transformation matrix A is equal to the product of three rotation matrices [6, 10, 11].

The rotational matrices of the XYZ sequence, also known as Bryant angles, and their correspondent product is represented in Eqs. (2.9), (2.10), (2.11) and (2.12).

$$D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\phi_1 & -s\phi_1 \\ 0 & s\phi_1 & c\phi_1 \end{bmatrix} \tag{2.9}$$

$$C = \begin{bmatrix} c\phi_2 & 0 & s\phi_2 \\ 0 & 1 & 0 \\ -s\phi_2 & 0 & c\phi_2 \end{bmatrix} \tag{2.10}$$

$$B = \begin{bmatrix} c\phi_3 & -s\phi_3 & 0 \\ s\phi_3 & c\phi_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.11}$$

$$A = DCB = \begin{bmatrix} c\phi_2 c\phi_1 & -c\phi_2 s\phi_3 & s\phi_2 \\ c\phi_1 s\phi_3 + s\phi_1 s\phi_2 c\phi_3 & c\phi_1 c\phi_3 - s\phi_1 s\phi_2 s\phi_3 & -s\phi_1 c\phi_2 \\ s\phi_1 s\phi_3 - c\phi_1 s\phi_2 c\phi_3 & s\phi_1 c\phi_3 + c\phi_1 s\phi_2 s\phi_3 & c\phi_1 c\phi_2 \end{bmatrix} \quad (2.12)$$

The advantage of any sequence of Euler angles in having three independent rotational coordinates is nullified by the occurrence of a phenomenon known as gimbal lock or singularities that appear when two rotational axes coincide leading to the loss of one DOF [12]. It can, however, be concluded that this phenomenon is an inherent problem associated with the use of three rotational coordinates [6].

This issue can be addressed by employing the use of Euler parameters, whose application to multibody systems is extensively discussed in a series of articles written by Nikravesh ([13–15]), and arise from the Euler's Theorem "*The general displacement of a body with one point fixed is a rotation about some axis*" [6], that introduces the concept of the orientational axis of rotation.

The Euler parameters are defined as four non-independent rotational coordinates $p^T = [e_0, e_1, e_2, e_3]$ that represent the rotation of the local frame around the orientational rotation axis, with the direction vector of this axis being $u^T = [u_x, u_y, u_z]$. These identities are defined by the Eqs. (2.13) and (2.14) and are related to each other through Eq.(2.15). Its transformation matrix is also enunciated in Eq.(2.16) [16].

$$e_0 = cos(\frac{\phi}{2}) \quad (2.13)$$

$$e = sin(\frac{\phi}{2})u \quad (2.14)$$

$$e_0^2 + e_1^2 + e_2^2 + e_3^2 = 1 \Leftrightarrow p^T p - 1 = 0 \quad (2.15)$$

$$A = 2 \begin{bmatrix} e_0^2 + e_1^2 - \frac{1}{2} & e_1 e_2 - e_0 e_3 & e_1 e_3 + e_0 e_2 \\ e_1 e_2 + e_0 e_3 & e_0^2 + e_2^2 - \frac{1}{2} & e_2 e_3 - e_0 e_1 \\ e_1 e_3 - e_0 e_2 & e_2 e_3 + e_0 e_1 & e_0^2 + e_3^2 - \frac{1}{2} \end{bmatrix} \quad (2.16)$$

Euler Parameters Derivative

With the introduction of the Euler parameters, it is also necessary to present a set of identities and equations that establish a relation between the derivatives of the three Cartesian rotational coordinates and the Euler parameters derivatives.

Nikravesh [6] starts by defining the identities G and L, represented in Eqs. (2.17) and (2.18), that relate the global and the local coordinates of rotation with the Euler parameters, respectively.

$$G = \begin{bmatrix} -e_1 & e_0 & -e_3 & e_2 \\ -e_2 & e_3 & e_0 & -e_1 \\ -e_3 & -e_2 & e_1 & e_0 \end{bmatrix} \tag{2.17}$$

$$L = \begin{bmatrix} -e_1 & e_0 & e_3 & -e_2 \\ -e_2 & -e_3 & e_0 & e_1 \\ -e_3 & e_2 & -e_1 & e_0 \end{bmatrix} \tag{2.18}$$

By introducing the vectors ω and ω', corresponding to the global and local angular velocity respectively, alongside some mathematical manipulation that is not of interest for this work, it is possible to obtain Eqs. (2.19) and (2.20), which relate the first Euler parameter derivative with the angular velocity vector.

$$\omega = 2G\dot{p} \tag{2.19}$$

$$\omega' = 2L\dot{p} \tag{2.20}$$

Applying the differentiation with respect to time to Eqs. (2.19) and (2.20), the relation between the angular acceleration and the second Euler parameters derivative is obtained, as presented in Eqs. (2.21) and (2.22).

$$\dot{\omega} = 2G\ddot{p} \Leftrightarrow \ddot{p} = \frac{1}{2}G^T\dot{\omega} - \frac{1}{4}(\omega^T\omega)p \tag{2.21}$$

$$\dot{\omega'} = 2L\ddot{p} \Leftrightarrow \ddot{p} = \frac{1}{2}L^T\dot{\omega'} - \frac{1}{4}(\omega'^T\omega')p \tag{2.22}$$

2.3 Constraint Equations

The constraint equations constitute the algebraic relation between the independent and dependent generalised coordinates that define the relative translation and rotation between two bodies. The three types of constraints that form the basis of the kinematic and dynamic analysis, and their contributions to the Jacobian matrix, the right-hand side of the velocity and acceleration constraint equations will be presented next.

2.3.1 Joint Constraint Equations

The standard joint equations, in their basis, are comprised of a combination of constraints between two vectors, with the exception of the spherical joint which fixes the distance of two bodies to a point P during its motion. These vectors can be of fixed length, a type one constraint, or variable length, a type two constraint, and are defined about the same coordinate frame, the global coordinate system.

As exposed by Nikravesh [6], the constraint between two vectors varies between maintaining the two vectors parallel or perpendicular during the system motion. For the present work, however, the usage of the parallel constraint equations is avoided due to the possible occurrence of a critical case when both vectors are parallel to one of the global coordinate axes.

The occurrence of this critical case sets the entries of that constraint in the Jacobian matrix to zero, causing the matrix to become singular, which in turn leads to numerical difficulties [6]. Thus, to avoid the usage of the parallel constraint, an orthogonal triad is established using the vector $\vec{s}_i$, which produces two arbitrary vectors that are perpendicular between themselves and $\vec{s}_i$, as represented in Fig. 2.3. Subsequently, if these vectors are made to be also perpendicular to the vector $\vec{s}_j$, using a perpendicular type one constraint, it is possible to guarantee that $\vec{s}_i$ and $\vec{s}_j$ will always remain parallel during the body motion [17].

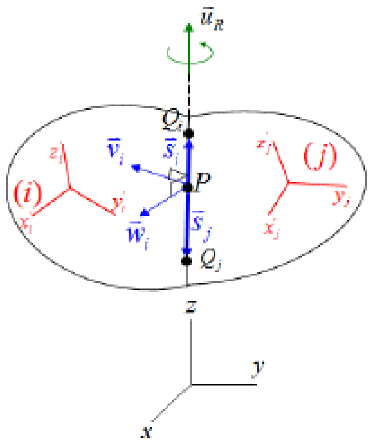


Figure 2.3: Revolute joint parallel constraint replacement [17].

Eqs. (2.23) to (2.26) represent the basic vector constraint equations previously addressed. Vector $\vec{d}$ present in Eqs. (2.24) and (2.26) is a vector of variable length as previously mentioned and it is possible to infer from Eq.(2.27) that the global components of the vector will depend on the absolute position of the point.

$$\Phi^{(n1,1)} \equiv s_i^T s_j \tag{2.23}$$

$$\Phi^{(n2,1)} \equiv s_i^T d \tag{2.24}$$

$$\Phi^{(p1,2)} \equiv \tilde{s}_i s_j \tag{2.25}$$

$$\Phi^{(p2,2)} \equiv \tilde{s}_i d \tag{2.26}$$

$$d = (r_j + s_j^B) - (r_i + s_i^B) \Leftrightarrow d = r_j + A_j s_j'^B - r_i - A_i s_i'^B \tag{2.27}$$

Having presented the building blocks, the introduction of the more complex kinematic joints equations becomes feasible. However, in the interest of brevity, the more trivial joint equations can be found by directly consulting Nikravesh [6] or Dye [17]. Only the revolute, cylindrical and prismatic joints that suffer changes due to the replacement of their parallel building blocks are enunciated in Eqs. (2.28), (2.29) and (2.30) respectively.

$$\begin{bmatrix} \Phi^{(s,3)} \equiv d \\ \Phi^{(p1,2)} \equiv \tilde{s}_i s_j \end{bmatrix} = 0 \Leftrightarrow \begin{bmatrix} r_i + A_i s_i^P - r_j - A_j s_j^P \\ W_i^T s_j \\ V_i^T s_j \end{bmatrix} = 0 \tag{2.28}$$

$$\begin{bmatrix} \Phi^{(p1,2)} \equiv \tilde{s}_i s_j \\ \Phi^{(p2,2)} = \tilde{s}_i d \end{bmatrix} = 0 \Leftrightarrow \begin{bmatrix} W_i^T d \\ V_i^T d \\ W_i^T s_j \\ V_i^T s_j \end{bmatrix} = 0 \tag{2.29}$$

$$\begin{bmatrix} \Phi^{(p1,2)} \equiv \tilde{s}_i s_j \\ \Phi^{(p2,2)} = \tilde{s}_i d \\ \Phi^{(n1,1)} \equiv h_i^T h_j \end{bmatrix} = 0 \Leftrightarrow \begin{bmatrix} W_i^T d \\ V_i^T d \\ W_i^T s_j \\ V_i^T s_j \\ V_i^T W_j \end{bmatrix} = 0 \tag{2.30}$$

Additionally, it is also possible to establish constraints that result directly from the combination of two standard joints, such as the more known spherical-spherical constraint and spherical-revolute constraint, whose equations can be consulted in Nikravesh [6], or the prismatic-revolute constraint, which was specifically deduced for this work, and is represented in Eq.(2.31).

$$\begin{bmatrix} \Phi^{(n2,1)} \equiv q_i^T d \\ \Phi^{(n2,1)} \equiv s_i^T d \\ \Phi^{(n1,1)} \equiv q_i^T q_j \\ \Phi^{(n1,1)} \equiv s_i^T s_j \end{bmatrix} = 0 \tag{2.31}$$

The deduction of these joint constraint equations starts with the already deduced equations from Nikravesh [6] for the composite revolute-cylindrical joint, represented in Fig. 2.4, and the replacement of its parallel vector constraint by the first two equations of the previously presented system in Eq.(2.31). Knowing that the difference between a revolute-cylindrical and a revolute-prismatic joint is that body j can only translate in the vector $\vec{s}_j$ direction and not rotate about the axis represented by the same vector, there is a need to introduce a fourth equation that guarantees that the vector $\vec{q}_i$, Eq.(2.32), and vector $\vec{q}_j$, Eq.(2.33), remain perpendicular during the bodies motion and consequently restrict the rotation of body j along the $\vec{s}_j$ axis.

$$\vec{q}_i = \vec{s}_i \times \frac{\vec{d}}{|\vec{d}|} \tag{2.32}$$

$$\vec{q}_j = \vec{s}_j \times \frac{\vec{q}_i}{|\vec{q}_i|} \tag{2.33}$$

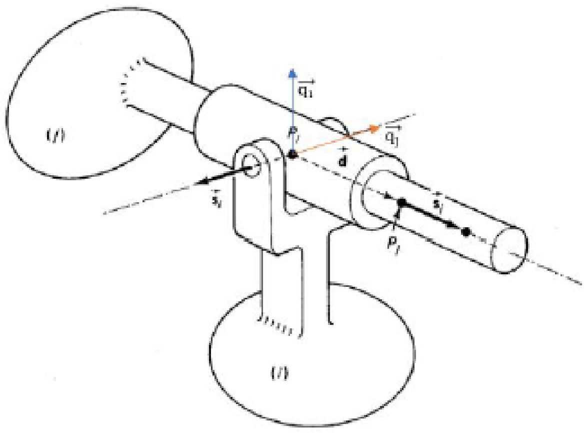


Figure 2.4: Representation of the perpendicular vectors q_i and q_j in Nikravesh revolute-cylindrical joint diagram [6].

With the introduction of all needed joints equations, it is important to resume all the information gathered into Tab. 2.2, that details the number of DOF that each of the aforementioned constraints restrict and whether they are translational or rotational. This information is vital to the selection of the constraints that are applied to each body together with the system typology and characterisation of the driving constraints that cause the motion of the system.

Joint Type	Translational DOF	Rotational DOF	Total DOFs restricted
Spherical	3	0	3
Universal	3	1	4
Revolute	3	2	5
Cylindrical	2	2	4
Prismatic	2	3	5

Table 2.2: Restricted DOFs for the standard joint equations [18].

Velocity and Acceleration Contribution

The joint constraint equations previously presented to the reader are the ones used to perform the position analysis, Eq.(2.34). However, to perform the velocity and acceleration analysis, it is also necessary to evaluate its first and second derivatives, Eqs. (2.35) and (2.36).

$$\Phi = \Phi(q,t) = 0 \tag{2.34}$$

$$\Phi_q \dot{q} = -\Phi_t \tag{2.35}$$

$$\Phi_q \ddot{q} = \gamma \Leftrightarrow \Phi_q \ddot{q} = -(\Phi_q \dot{q})_q \dot{q} - 2\Phi_{qt}\dot{q} - \Phi_{tt} \tag{2.36}$$

In the specific case of the joint constraint equations, a time dependency is not exhibited and the terms Φ_t,Φ_{qt} and Φ_{tt} are zero. Therefore, to perform both velocity and acceleration analysis, it is only necessary to define the joint constraint equations entries for the acceleration term $-(\Phi_q \dot{q})_q \dot{q}$ and to the Jacobian matrix, Φ_q. For the latter, the entries are equal to the constraint equations partial derivatives with respect to the generalised absolute coordinates q, Eq.(2.37).

$$\Phi_q = \begin{bmatrix} \frac{\partial \phi_1}{\partial q_1} & \frac{\partial \phi_1}{\partial q_2} & \cdots & \frac{\partial \phi_1}{\partial q_n} \\ \frac{\partial \phi_2}{\partial q_1} & \frac{\partial \phi_2}{\partial q_2} & \cdots & \frac{\partial \phi_2}{\partial q_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial phi_m}{\partial q_1} & \frac{\partial \phi_m}{\partial q_2} & \cdots & \frac{\partial \phi_m}{\partial q_n} \end{bmatrix} \tag{2.37}$$

As a result, the entries to the Jacobian matrix are dependent on the type of rotational coordinates that are used. Tab. 2.3 resumes the entries for the most common constraints, for both the Jacobian and the right-hand side acceleration vector, assuming that the Euler parameters are used as rotational coordinates.

Φ	Φ_{r_i}	Φ_{p_i}	Φ_{r_j}	Φ_{p_j}	$\gamma^{(m)}$
$\Phi^{(n1,1)}$	0^T	$s_j^T C_i$	0^T	$s_i^T C_j$	$s_i^T h_j + s_j^T h_i - 2\dot{s}_i^T \dot{s}_j$
$\Phi^{(n2,1)}$	$-s_i^T$	$-s_i^T B_i + d^T C_i$	s_i^T	$s_i^T B_j$	$-s_i^T (h_i^B - h_j^B) + d^T h_i - 2\dot{s}_i^T \dot{s}_j$
$\Phi^{(p1,2)}$	0	$-\tilde{s}_j C_i$	0	$\tilde{s}_i C_j$	$\tilde{s}_j h_j - \tilde{s}_j h_i - 2\dot{\tilde{s}}_i \tilde{s}_j$
$\Phi^{(p2,2)}$	$-\tilde{s}_i$	$-\tilde{s}_i B_i - \tilde{d} C_i$	$\tilde{s}_i$	$\tilde{s}_i B_j$	$-\tilde{s}_i (h_j^B - h_i^B) - \tilde{d} h_i - 2\dot{\tilde{s}}_i \dot{d}$
$\Phi^{(s-s,1)}$	$-2d^T$	$-2d^T B_i$	$2d^T$	$2d^T B_j$	$2d^T (h_i^P - h_j^P) - 2\dot{d}^T \dot{d}$
$\Phi^{(s,3)}$	I	C_i	$-I$	$-C_j$	$h_i^P - h_j^P$

Table 2.3: Standard constraints components for the Jacobian matrix and the right hand side acceleration analysis vector [6].

$$B_k = 2(G_k \bar{s}_k^{'B} + s_k^{'B} p_k^T), k = i, j \tag{2.38}$$

$$C_k = 2(G_k \bar{s}_k^{'} + s^{'} p_k^T), k = i, j \tag{2.39}$$

$$h_k = -2\dot{G}_k \dot{L}_k^T s_k^{'} \tag{2.40}$$

$$h_k^B = -2\dot{G}_k \dot{L}_k^T s_k^{'B} \tag{2.41}$$

$$h_k^P = -2\dot{G}_k \dot{L}_k^T s_k^{'P} \tag{2.42}$$

However, when the Euler parameters are time-dependent, it is possible to solve the equations of motion more efficiently by not writing them in terms of Euler parameters. Thus, Nikravesh [6] introduced a set of modified kinematic equations that allow the Jacobian matrix to be written in terms of Euler parameters, while being modified to allow the equation of motion to be written in terms of Cartesian coordinates. As such, Tab. 2.4 shows the converted components of the Jacobian matrix and acceleration vector [17].

Φ	$\Phi_{r_i}^{(m)}$	$\frac{1}{2}\Phi_{p_i}^{(m)} L_i^T$	$\Phi_{r_j}^{(m)}$	$\frac{1}{2}\Phi_{p_j}^{(m)} L_j^T$	$\gamma^{\#}$
$\Phi^{(n1,1)}$	0^T	$-s_j^T \tilde{s}_i A_i$	0^T	$-s_i^T \tilde{s}_j A_j$	$-2\dot{s}_i^T \dot{s}_j + \dot{s}_i^T \tilde{\omega}_i s_j + \dot{s}_j^T \tilde{\omega}_j s_i$
$\Phi^{(n2,1)}$	$-s_i^T$	$-(d + s_i^B)^T \tilde{s}_i A_i$	s_i^T	$-s_i^T \tilde{s}_j^B A_j$	$-2\dot{d}^T \dot{s}_i - d^T \tilde{\omega}_i \dot{s}_i + s_i^T (\tilde{\omega}_i \dot{s}_i^B - \tilde{\omega}_j \dot{s}_j^B)$
$\Phi^{(p1,2)}$	0	$\tilde{s}_j \tilde{s}_i A_i$	0	$-\tilde{s}_i \tilde{s}_j A_j$	$-2\tilde{\dot{s}}_i \dot{s}_j + \tilde{s}_j \tilde{\omega}_i \dot{s}_i - \tilde{s}_i \tilde{\omega}_j \dot{s}_j$
$\Phi^{(p2,2)}$	$-\tilde{s}_i$	$(\tilde{s}_i \tilde{s}_i^B + \tilde{d} \tilde{s}_i) A_i$	$\tilde{s}_i$	$-\tilde{s}_i \tilde{s}_j^B A_j$	$-2\tilde{\dot{s}}_i \dot{d} + \tilde{s}_i (\tilde{\omega}_i \dot{s}_i^B - \tilde{\omega}_j \dot{s}_j^B) + \tilde{d} \tilde{\omega}_i \dot{s}_i$
$\Phi^{(s-s,1)}$	$-2d^T$	$2d^T \tilde{s}_i^P A_i$	$2d^T$	$-2d^T \tilde{s}_j^P A_j$	$2\dot{d}^T \dot{d} + 2d^T (\tilde{\omega}_i \dot{s}_i^P - \tilde{\omega}_j \dot{s}_j^P)$
$\Phi^{(s,3)}$	I	$-\tilde{s}_i^P A_i$	$-I$	$\tilde{s}_j^P A_j$	$-\tilde{\omega}_i \dot{s}_i^P + \tilde{\omega}_j \dot{s}_j^P$

Table 2.4: Standard constraint converted components for the Jacobian matrix and the right hand side acceleration analysis vector [17].

2.3.2 Driving Constraints

For this work, the method of appended driving constraints is used in opposition to the coordinate partitioning method, due to its lower requirement of computation power. Thus, additional driving constraints must be defined to equal the number of independent generalised coordinates [6].

In other words, if there is a total of n dependent and independent Degrees of Freedom and m Degrees of Freedom restricted by the kinematic constraints, it is necessary to define $k = n - m$ driving constraints, for a kinematic analysis.

These driving constraints are equations that define each independent coordinate as a function of time [6]. This dependency on time may cause the first and second derivatives to be different from zero and must be taken into consideration when formulating velocity and acceleration problems, Eqs. (2.43) and (2.44) respectively, where the superscript *(d)* is denoting the driving equations.

$$\begin{bmatrix} \Phi_q \\ \Phi_q^{(d)} \end{bmatrix} \dot{q} = \begin{bmatrix} 0 \\ -\Phi_t^{(d)} \end{bmatrix} \tag{2.43}$$

$$\begin{bmatrix} \Phi_q \\ \Phi_q^{(d)} \end{bmatrix} \ddot{q} = \begin{bmatrix} -(\Phi_q \dot{q})_q \dot{q} \\ -(\Phi_q^{(d)} \dot{q})_q \dot{q} - 2\Phi_{qt}^{(d)} \dot{q} - \Phi_{tt}^{(d)} \end{bmatrix} \tag{2.44}$$

2.3.3 Euler Parameters Constraints

As viewed in Sec. 2.2.2 during the introduction of the Euler parameters, it was concluded that the four rotational coordinates that are introduced with their usage are not independent. Consequently, it is necessary to introduce a constraint that relates the dependent DOF with the remaining three. For this purpose Eq.(2.15) is used.

If the Jacobian matrix, which is used to calculate the velocity and acceleration of each body, is written as a function of the Euler parameters, the constraint equation becomes the derivative of Eq.(2.15), resulting in Eq.(2.45) [17].

$$C_q \dot{q} = \begin{bmatrix} 0 & 2p_i^T \end{bmatrix} \begin{bmatrix} \dot{r}_i \\ \dot{p}_i \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix} \tag{2.45}$$

Following this thought process, the right-hand side of the acceleration problem ends up corresponding to the second derivative of Eq.(2.15), Eq.(2.46) [17].

$$C_q \ddot{q} = -\dot{p}_i^T \dot{p}_i \tag{2.46}$$

2.4 Kinematic Analysis

Several authors in the literature have defined and elaborated upon the kinematic analysis problem such as Nikravesh [6], Jazar [7], De Jalon and Bayo [19], and Flores and Lankarani [20]. According to them, the performance of a kinematic simulation is the resolution of three sub-problems, that allow the user to have an overview of the motion of the multibody system, without needing the data to fully characterise each body in the system, such as mass, moments of inertia and position of the CoM.

This is best evidenced by the definition of kinematic analysis by Lankarani and Paulo Flores [20] "*The kinematic analysis is the study of the motion of a multibody system, independently of the causes that produce it*". As a result, during a kinematic analysis forces are not considered and the driving constraints fully define the motion of the system, implying that these must equal the number of independent DOFs of the system, as previously stated in Sec. 2.3.2.

The finite displacement problem represented by Eq.(2.47), also known as position analysis, consists of calculating the final position of the system, based on the finite displacement input for the driving constraints and a given fixed initial position. The constraint equations that compose this problem, defined in Sec. 2.3, are non-linear in terms of the generalised coordinates, q, requiring the use of an iterative solution algorithm. Usually, the Newton-Raphson method is used for this task, however, *MATLAB* possesses its own set of non-linear solvers that employ other algorithms such as trust-region, trust-region Dogleg and Levenberg-Marquardt algorithms.

$$\begin{bmatrix} \Phi(q) \\ \Phi^{(d)}(q,t) \end{bmatrix} = 0 \tag{2.47}$$

The remaining velocity and acceleration sub-problems, represented by Eqs. (2.43) and (2.44), are linear in the generalised coordinates and its derivates, $\dot{q}$ and $\ddot{q}$. The velocities of the dependent elements of the system are determined from the previously obtained positions and driving velocities. In the case of the accelerations, its determination not only takes into account the positions and driving accelerations but also the velocities that are necessary to equate the right hand side vector, γ, resulting from the joint constraint equations contributions, as represented by the flowchart of Fig. 2.5.

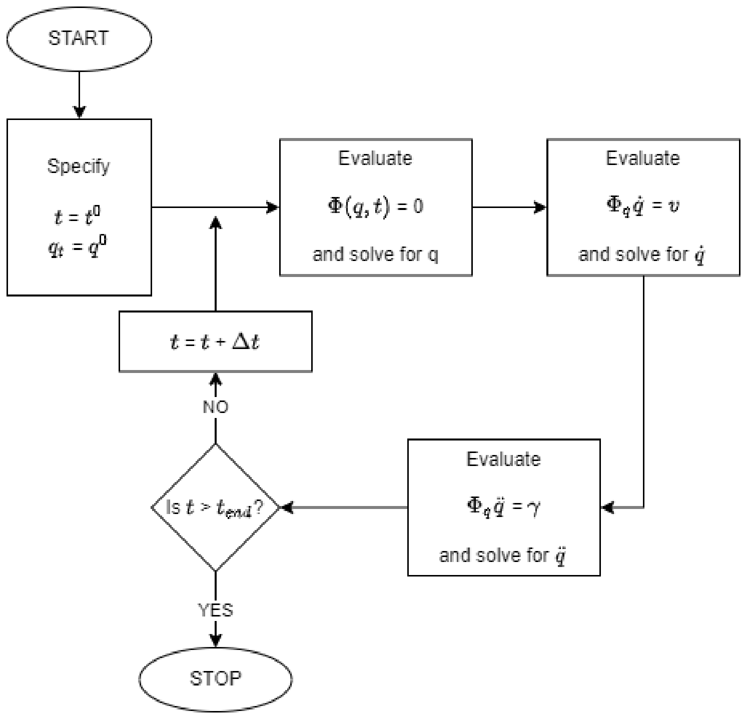


Figure 2.5: Flowchart that describes the process to perform a kinematic simulation [10].

Nikravesh [6] mentions several methods for solving the set of linear equations that compose these two last problems, namely the Gaussian elimination, Gauss-Jordan reduction, and the L-U factorisation methods. However, in similarity with the non-linear solvers, *MATLAB* provides its own solver that selects the best resolution method between the Gauss elimination, Cholesky decomposition, L-U factorisation, and pivoting, based on the matrix that defines the system [17, 21].

2.5 Dynamic Analysis

Similarly to the kinematic analysis, the dynamic analysis problem has been widely studied and developed upon in the thematic literature [6, 7, 9, 10, 20, 22–24]. In this work, however, the dynamic analysis only contemplates the problem defined as forward dynamics.

In comparison to the kinematic analysis, the dynamic simulation or kinetic analysis is a far more complex problem to solve since it provides a relationship between the multibody system motion and the forces that cause it [25]. Therefore, the inertial characteristics of the system must be defined, such as the inertia tensor, mass, and position of the CoM.

2.5.1 Forces

During a simulation, a multibody system can be subjected to several forces, which are divided into external forces, that are outside the boundaries of the systems, and internal forces, such as the constraint forces, introduced by the kinematic joints.

According to Flores [10], there are multiple sources for these forces, and they can be gravitational forces, spring-damper-actuator forces, externally applied forces, contact forces and frictional forces. However, the last two are out of this work scope and only the principles of implementation of the translational spring-damper-actuator and rotational spring are worth introducing to the reader.

Akin to the kinematic constraints, these force elements make use of vectors to define pairs of forces acting between two bodies. These pairs of forces can be defined as acting on the CoM or at an arbitrary point of the body, in which case, besides the pair of forces, there is also an application of a moment.

For the translational force elements, a vector $\vec{l}$ is defined between the two body points, as represented by Eq.(2.48), which is used to define the unit vector of the applied forces, the spring displacement, and the rate of change of the damper length [6].

$$l = r_j + A_j s_j^{'P} - r_i - A_i s_i^{'P} \tag{2.48}$$

Eqs. (2.49) and (2.50) represent the translational force elements equation for each body, where $f^{(s)}$ is the magnitude of the force to be applied. For both the actuator and the spring, the bodies are being pulled towards each other if $f > 0$, and being pushed if $f < 0$. In the case of the spring, the magnitude of the force is calculated by the Eq.(2.51) which represents the linear translational spring equation, where k is the spring stiffness constant and l^0 is its undeformed length.

$$f_i^{(s)} = f^{(s)} u \tag{2.49}$$

$$f_j^{(s)} = -f^{(s)} u \tag{2.50}$$

$$f^{(s)} = k(l - l^0) \tag{2.51}$$

The linear translational damper equation, used to calculate the magnitude of the force exerted by the damper, is represented by Eq.(2.52). The magnitude is dependent on the rate of change of the damper length, which is calculated using Eqs. (2.53) and (2.54). Since the damper opposes the relative motion of the two bodies, if the rate of change is positive, $\dot{l} > 0$, then the forces of the damper is exhibiting a pull on the two bodies, if the rate of change is negative, $\dot{l} < 0$, the forces of the damper exhibit a push.

$$f^{(d)} = c\dot{l} \tag{2.52}$$

$$\dot{l} = \frac{l^T \dot{l}}{l} \tag{2.53}$$

$$\dot{l} = \dot{r}_j + A_j \tilde{\omega}_j s^{'}_{P_j} - \dot{r}_i - A_i \tilde{\omega}_i s^{'}_{P_i} \tag{2.54}$$

In contrast to the force elements previously introduced, the rotational spring only introduces a pure moment and requires that both bodies be connected by a revolute joint whose revolution axis, s, is coincident with the rotational spring axis. Eq.(2.55) represents the equation for the moment magnitude, where θ is the angle calculated using Eqs. (2.56) and (2.57) between the two ends of the spring which are attached to each one of the bodies i and j, Fig. 2.6, while θ^0 is the undeformed length. If $\theta > \theta^0$, the moment is positive in the rotational direction, and it is applied to each body using Eqs. (2.58) and (2.59) [6].

$$n^{(r-s)} = k(\theta - \theta^0) \tag{2.55}$$

$$\theta = cos^{-1}\left(\frac{s_i^T s_j}{|s_i| \cdot |s_j|}\right) \;,\; 0 \leq \theta \leq \pi \tag{2.56}$$

$$s^T \tilde{s}_i s_j = \begin{cases} \geq 0 & \text{for } 0 \leq \theta \leq \pi \\ \leq 0 & \text{for } \pi \leq \theta \leq 2\pi \end{cases} \tag{2.57}$$

$$n_i^{(r-s)} = n^{(r-s)} \tag{2.58}$$

$$n_j^{(r-s)} = -n^{(r-s)} \tag{2.59}$$

2.5.2 Equations of Motion

In the case of multibody systems, the formulation of the equations of motion contemplates a translational and a rotational component. The translational component is constituted by Newton's equation, Eq.(2.60), while the rotational equations of motion are composed by the Euler equations, Eq.(2.61) [10].

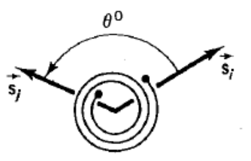


Figure 2.6: Representation of the rotational spring [6].

$$m\ddot{r} = f \tag{2.60}$$

$$J^{'}\dot{\omega} + \tilde{\omega}^{'}J^{'}\omega^{'} = n^{'} \tag{2.61}$$

These two equations can be combined to form the Newton-Euler equations of motion represented by the system in Eq.(2.62). Where m is the scalar mass of the body, I is a 3×3 identity matrix, and J' is the inertia tensor in the body frame.

$$\begin{bmatrix} mI & 0 \\ 0 & J^{'} \end{bmatrix} \begin{bmatrix} \ddot{r} \\ \dot{\omega}^{'} \end{bmatrix} = \begin{bmatrix} f \\ n - \tilde{\omega}^{'}J^{'}\omega^{'} \end{bmatrix} \tag{2.62}$$

The system of Eq.(2.62) is for unconstrained bodies, however, for a constrained system, it is necessary to consider the reaction forces of each joint. According to Nikravesh [6] and Shabana [9], these can be expressed with the resource to the augmented formulation and the Lagrange multipliers (λ) method, to form the equations of motion for a constrained multibody system, Eq.(2.63).

$$M\ddot{q} - \Phi_q^T\lambda = g \tag{2.63}$$

Due to the introduction of the reaction forces, the system in Eq.(2.63) becomes undetermined with more m unknowns, the Langrange multipliers, than differential equations, and thus assumes infinite solutions. However, according to Uchida [23], it is possible to use the second time derivative of the constraint equations introduced in Sec. 2.3, to produce a determined $n+m$ differential-algebraic system of equations of index one, represented by Eq.(2.64).

$$\begin{bmatrix} M & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ -\lambda \end{bmatrix} = \begin{bmatrix} g \\ \gamma \end{bmatrix} \tag{2.64}$$

Furthermore, for the implementation of some algorithms needed to solve the forward dynamic problem, it may be easier to work with the equations of motion written in terms of Euler parameters. In this case, Nikravesh and Chung [15], presents the transformed equations of the mass matrix, Eq.(2.65), of the moments present in the force vector, Eq.(2.66), as well as the transformed equation for force elements, Eq.(2.67), where $n_i^{'} = [n_\xi, n_\eta, n_\zeta]$ and $n_i = [n_{e0}, n_{e1}, n_{e2}, n_{e3}]$.

$$M_i = \begin{bmatrix} mI & 0 \\ 0 & [4L^T J' L]_i \end{bmatrix} \tag{2.65}$$

$$g_i = \begin{bmatrix} f^T & (n - 8\dot{L}^T J' L\dot{p})^T \end{bmatrix}_i^T \tag{2.66}$$

$$n_i = 2L_i^T n_i' \tag{2.67}$$

2.5.3 Procedure and Constraint Stabilisation

The dynamic analysis starts with the consistency check of the initial conditions to ensure that there are no constraint violations, followed by the resolution of an initial value problem composed by the Differential-Algebraic Equation (DAE) system of Eq.(2.64). As suggested by Fig. 2.7, given set of initial conditions (positions and velocities), the initial system can be solved for $\ddot{q}$ and λ and then integrated by using an initial solver algorithm such as a Runge-Kutta or predictor-corrector algorithm [20], providing the system velocities $\dot{q}$ and positions q for the next time step.

However, the numerical error inherent to the integration of these $\ddot{q}$ and $\dot{q}$ values can lead to the quadratic accumulation of significant position-level constraint violations over time [23, 26]. This occurs because the system of Eq.(2.64) does not use the position and velocity constraint equations explicitly, allowing for its violation during the integration process [20].

Several methods to mitigate the errors of these violations in the position and velocity equations have been developed and must be integrated into any multibody simulator that uses the second derivative of the constraint equations for the index reduction of the DAE system [23]. Some of these methods are the Baumgarte stabilisation, the penalty method, the Augmented Lagrangian Formula, the coordinate partitioning method and the direct correction method described by Yoon, Sugjoon [22].

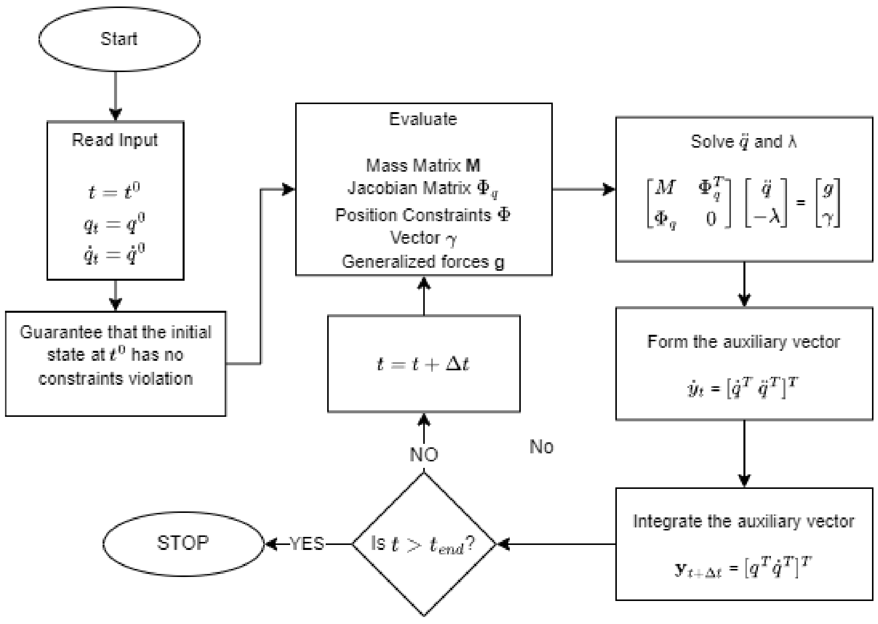

Start
Read Input
$t = t^0$
$q_t = q^0$
$\dot{q}_t = \dot{q}^0$
Guarantee that the initial state at t^0 has no constraints violation
Evaluate
Mass Matrix **M**
Jacobian Matrix Φ_q
Position Constraints Φ
Vector γ
Generalized forces g
Solve $\ddot{q}$ and λ
$\begin{bmatrix} M & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ -\lambda \end{bmatrix} = \begin{bmatrix} g \\ \gamma \end{bmatrix}$
Form the auxiliary vector
$\dot{y}_t = [\dot{q}^T \ \ddot{q}^T]^T$
Integrate the auxiliary vector
$\mathbf{y}_{t+\Delta t} = [q^T \dot{q}^T]^T$
$t = t + \Delta t$
NO
No
Is $t > t_{end}$?
YES
STOP

Figure 2.7: Flowchart of the computational procedure for the dynamic analysis.

Chapter 3

Code and Development Methodology

This chapter exposes the methodology adopted in the development of the simulator, its pseudo-code structure and mode of operation. It is also focused on the inputs and outputs of the program, as well as the different available algorithms to solve the proposed problems of Chapter 1 and their selection.

3.1 Development Methodology

3.1.1 Requirements

The tool developed for this work is a general proposed multibody system software intended to be used by FST Lisboa, as a vehicle dynamics simulator. As a result, it must not only meet the basic requirements enumerated by Kortüm et al [1], such as availability of basic MBS modelling elements, simulation reliability and robust and efficient solvers, but it must also meet or allow the future implementation of functionalities specific to vehicle dynamics. Of these specific requirements, Kortüm et al [1] highlights the need for the consideration of tire contact models, non-linear effects such as friction, non-linear and discontinuous force laws, as well as the ability to model complex track geometries and their stochastic irregularities.

3.1.2 Milestones

With the requirements specified, the objectives introduced in Sec. 1.2 are deconstructed and divided into smaller milestones. The first milestone is the implementation of the program pre-processing structure, modelling the *Excel* to receive all the bodies input, namely their position vector and orientation, and the joints inputs, as well as the pre-processing function that allocates the information to the body and joint data structures.

The second milestone is the implementation of the constraint equations presented in Sec. 2.3, followed by the implementation of the kinematic algorithm detailed in Sec. 2.4 and consequent verification with linear driver inputs and non-linear driver inputs.

Since the pre-processing structure is already implemented, the third milestone focuses on its adaptation to be flexible in running the user picked simulation, extracting, and allocating the body inertial inputs, initial forces and torques, and force elements. Furthermore, the implementations are verified for linear and non-linear discontinuous forces and force elements.

The fourth and final milestone is the implementation of graphic outputs, such as a Three Dimensional (3D) animated plot, which facilitate the overall debug and benchmark of the program.

3.1.3 Debug

As exposed in Fig. 3.1, during each milestone implementation a verification of the results obtained by the code is done using a slider-crank model, whose simplicity allows for a faster adaptation and simulation of the specific features implemented for each milestone. If the results of the verification present a difference greater than 5 % in comparison to the MSC Adams, a debug phase follows, where theoretical formulas are checked, breakpoints are issued, and the workspace variables are verified for possible discrepancies.

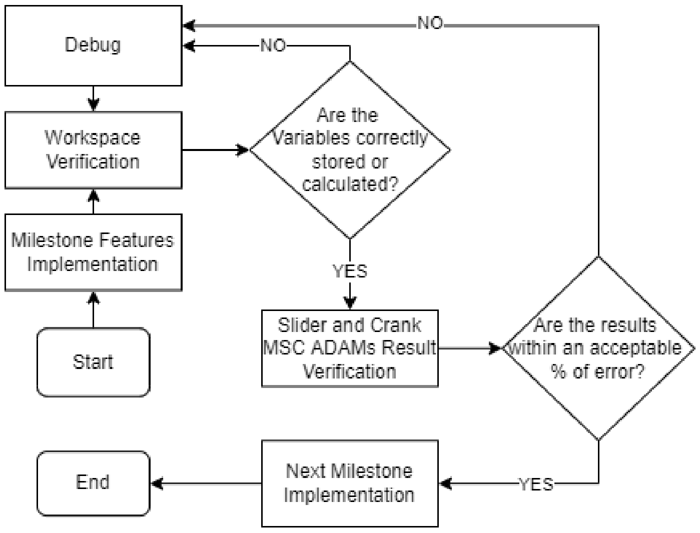

Debug
NO
NO
Workspace Verification
Are the Variables correctly stored or calculated?
Milestone Features Implementation
YES
Start
Slider and Crank MSC ADAMs Result Verification
Are the results within an acceptable % of error?
End
Next Milestone Implementation
YES

Figure 3.1: Flowchart of the methodology used during the development of this work.

However, the requirements of the program imply the capability of simulating models that have more complexity than a planar slider-crank model, and thus may cause other untested errors to arise. Therefore, besides the debug in between milestones an overall debug phase must occur, after the milestone's implementation.

This phase is done by using different models taken from the literature, namely from the *IFToMM* library [27]. Of these models, it is important to highlight the flyball governor model, represented in Fig. 3.2 and Fig. 3.3, whose dynamic motion is not trivial, and its system composition allows to perfectly illustrate the debug methodology adopted for this work, as represented in Fig. 3.4.

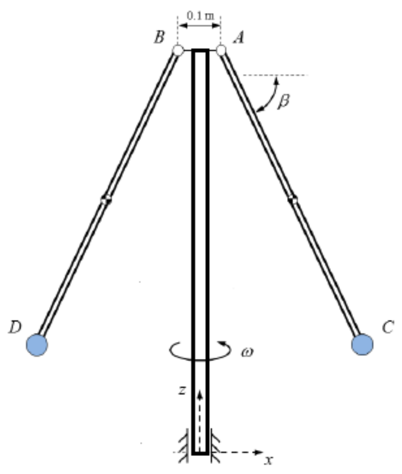


Figure 3.2: Simplified flyball governor model [27].

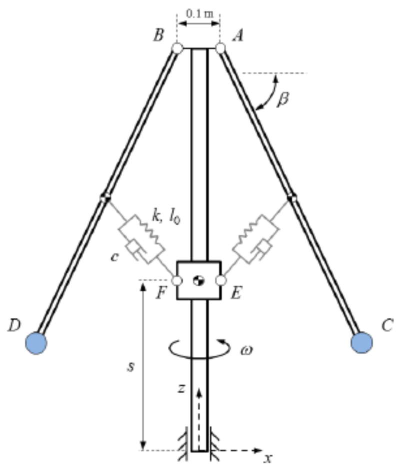


Figure 3.3: Full flyball governor model [27].

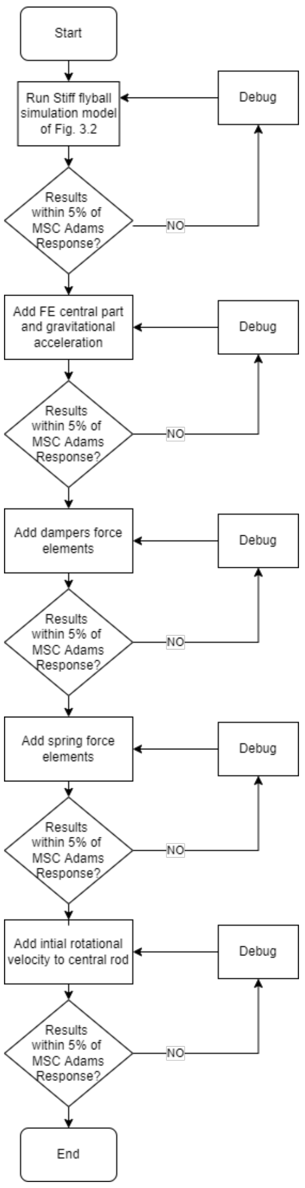
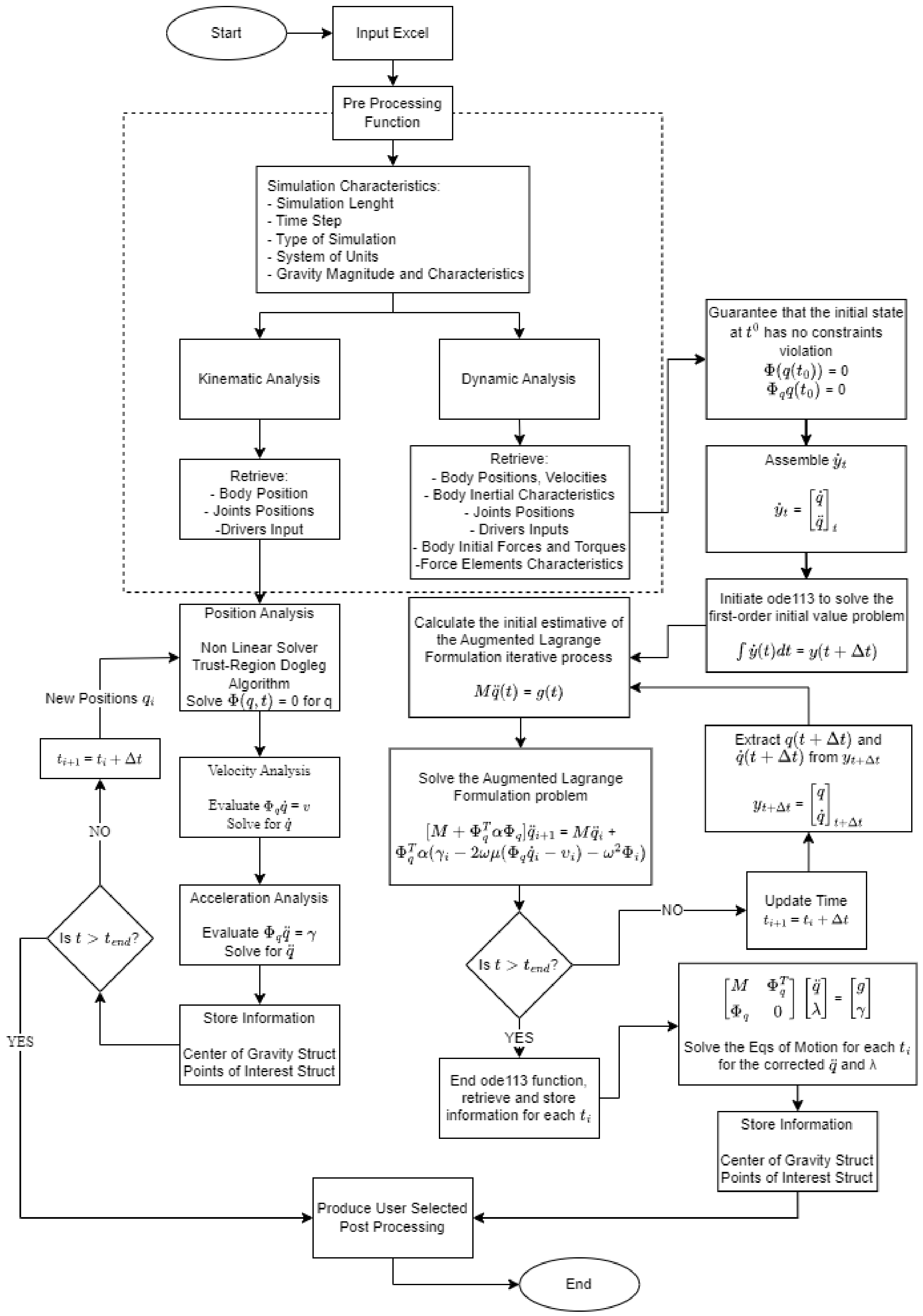

Start
Run Stiff flyball simulation model of Fig. 3.2
Debug
Results within 5% of MSC Adams Response?
NO
Add FE central part and gravitational acceleration
Debug
Results within 5% of MSC Adams Response?
NO
Add dampers force elements
Debug
Results within 5% of MSC Adams Response?
NO
Add spring force elements
Debug
Results within 5% of MSC Adams Response?
NO
Add intial rotational velocity to central rod
Debug
Results within 5% of MSC Adams Response?
NO
End

Figure 3.4: Flowchart of the methodology used during the debug process of complex multibody models.

3.2 Pseudo-Code

A pseudo-code of the program is presented in Fig. 3.5 allowing the reader to have an overview of its operation steps. The following sections bear a more detailed explanation of the pseudo-code and its steps.

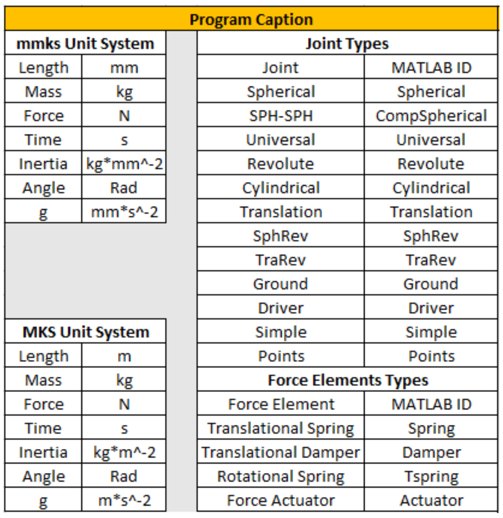


Start
Input Excel
Pre Processing Function
Simulation Characteristics: - Simulation Lenght - Time Step - Type of Simulation - System of Units - Gravity Magnitude and Characteristics
Kinematic Analysis
Dynamic Analysis
Retrieve: - Body Position - Joints Positions -Drivers Input
Retrieve: - Body Positions, Velocities - Body Inertial Characteristics - Joints Positions - Drivers Inputs - Body Initial Forces and Torques -Force Elements Characteristics
Guarantee that the initial state at t^0 has no constraints violation $\Phi(q(t_0))$ = 0 $\Phi_q q(t_0)$ = 0
Assemble $\dot{y}_t$ $\dot{y}_t = \begin{bmatrix}\dot{q}\\ \ddot{q}\end{bmatrix}_t$
Initiate ode113 to solve the first-order initial value problem $\int \dot{y}(t)dt = y(t+\Delta t)$
Position Analysis Non Linear Solver Trust-Region Dogleg Algorithm Solve $\Phi(q,t)$ = 0 for q
New Positions q_i
$t_{i+1} = t_i + \Delta t$
NO
Velocity Analysis Evaluate $\Phi_q \dot{q} = v$ Solve for $\dot{q}$
Acceleration Analysis Evaluate $\Phi_q \ddot{q} = \gamma$ Solve for $\ddot{q}$
Is $t > t_{end}$?
YES
Store Information Center of Gravity Struct Points of Interest Struct
Calculate the initial estimative of the Augmented Lagrange Formulation iterative process $M\ddot{q}(t) = g(t)$
Extract $q(t+\Delta t)$ and $\dot{q}(t+\Delta t)$ from $y_{t+\Delta t}$ $y_{t+\Delta t} = \begin{bmatrix}q\\ \dot{q}\end{bmatrix}_{t+\Delta t}$
Solve the Augmented Lagrange Formulation problem $[M + \Phi_q^T \alpha \Phi_q]\ddot{q}_{i+1} = M\ddot{q}_i + \Phi_q^T \alpha(\gamma_i - 2\omega\mu(\Phi_q \dot{q}_i - v_i) - \omega^2 \Phi_i)$
NO
Update Time $t_{i+1} = t_i + \Delta t$
Is $t > t_{end}$?
YES
$\begin{bmatrix}M & \Phi_q^T\\ \Phi_q & 0\end{bmatrix}\begin{bmatrix}\ddot{q}\\ \lambda\end{bmatrix} = \begin{bmatrix}g\\ \gamma\end{bmatrix}$ Solve the Eqs of Motion for each t_i for the corrected $\ddot{q}$ and λ
End ode113 function, retrieve and store information for each t_i
Store Information Center of Gravity Struct Points of Interest Struct
Produce User Selected Post Processing
End

Figure 3.5: Pseudo-code of the tool developed during this work.

Inputs

The input phase of the program corresponds to the first five steps of the pseudo-code for any simulation, its structure is composed of an *Excel* file and a *MATLAB* function that is responsible for extracting and allocating the data needed for each simulation to its correspondent data structures. The *Excel* file works through an identification code system, represented in Fig. 3.6, that is fundamental in allowing *MATLAB* to identify the modelling elements, such as joints and force elements, and the system of units in which the simulation data is filled in.



Program Caption

mmks Unit System	
Length	mm
Mass	kg
Force	N
Time	s
Inertia	kg*mm^-2
Angle	Rad
g	mm*s^-2

MKS Unit System	
Length	m
Mass	kg
Force	N
Time	s
Inertia	kg*m^-2
Angle	Rad
g	m*s^-2

Joint Types	
Joint	MATLAB ID
Spherical	Spherical
SPH-SPH	CompSpherical
Universal	Universal
Revolute	Revolute
Cylindrical	Cylindrical
Translation	Translation
SphRev	SphRev
TraRev	TraRev
Ground	Ground
Driver	Driver
Simple	Simple
Points	Points

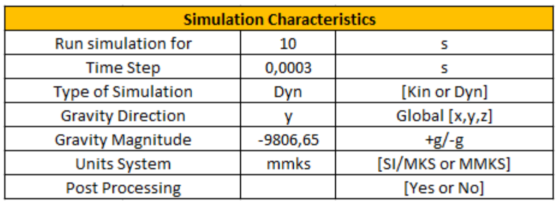
Force Elements Types	
Force Element	MATLAB ID
Translational Spring	Spring
Translational Damper	Damper
Rotational Spring	Tspring
Force Actuator	Actuator

Figure 3.6: Program identification code system.

The *Excel* is divided into six sheets that require the understanding of the user since it can actively influence the simulation results. These will be presented next in detail and are organised by simulation characteristics (*SimParam*), body positions and dynamic inputs (*Bodies*), joints data (*Joints*), motion driver's input data (*Joints Drivers*), force elements characteristics (*Force Elements*), and suspension points identification for post-processing (*PosProcessing*).

SimParam sheet

In the simulation characteristics sheet, the user is asked to fill in two tables as presented in Fig. 3.7 and Fig. 3.8. The first table, in Fig. 3.7, defines the simulation length, time step, the existence of gravity and its direction and magnitude, the data units system and if vehicle dynamics specific post-processing information is desired.

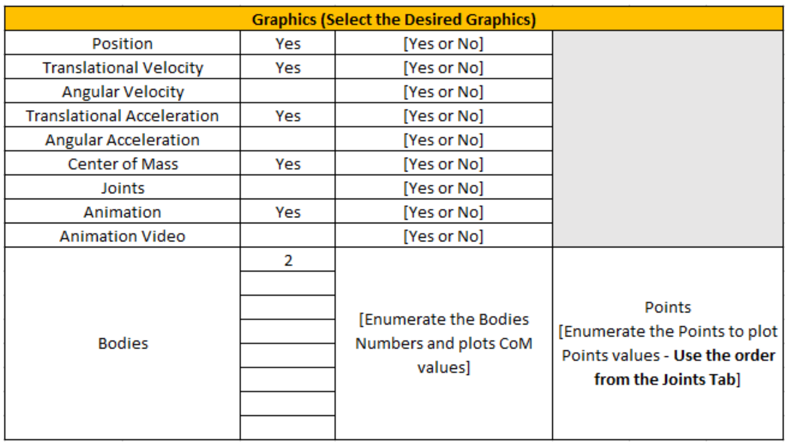


Simulation Characteristics		
Run simulation for	10	s
Time Step	0,0003	s
Type of Simulation	Dyn	[Kin or Dyn]
Gravity Direction	y	Global [x,y,z]
Gravity Magnitude	-9806,65	+g/-g
Units System	mmks	[SI/MKS or MMKS]
Post Processing		[Yes or No]

Figure 3.7: *Excel* table for the inputs of the overall simulation characteristics.

The second table, in Fig. 3.8, gives the user the option of requesting a 3D animation and Two

Dimensional (2D) plots of position, velocity and acceleration of the bodies CoM or points of interest, while being flexible to allow the selection of the specific variables needed for the study.

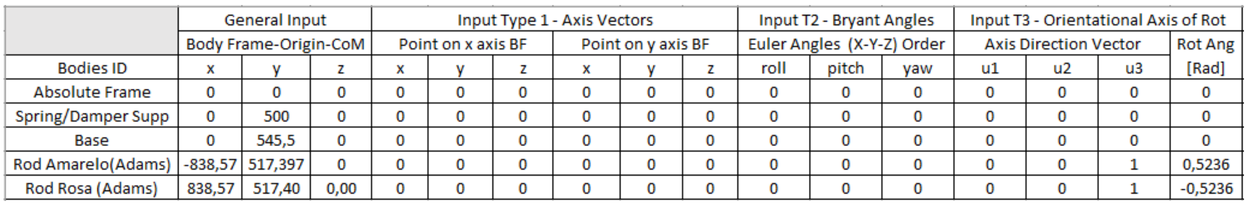


Graphics (Select the Desired Graphics)			
Position	Yes	[Yes or No]	
Translational Velocity	Yes	[Yes or No]	
Angular Velocity		[Yes or No]	
Translational Acceleration	Yes	[Yes or No]	
Angular Acceleration		[Yes or No]	
Center of Mass	Yes	[Yes or No]	
Joints		[Yes or No]	
Animation	Yes	[Yes or No]	
Animation Video		[Yes or No]	
Bodies	2	[Enumerate the Bodies Numbers and plots CoM values]	Points [Enumerate the Points to plot Points values - **Use the order from the Joints Tab**]

Figure 3.8: *Excel* table where the user inputs the variables to be plotted and the correspondent bodies or points.

Body Sheet

The inputs given to the bodies sheet are dependent on the type of simulation requested by the user. For both kinematic and dynamic simulations, the position of the body must be specified by a vector $\vec{r}$, as illustrated in Fig. 2.2, and its orientation can be specified using three different types of inputs, as illustrated by the table in Fig. 3.9.



	General Input			Input Type 1 - Axis Vectors						Input T2 - Bryant Angles			Input T3 - Orientational Axis of Rot			
	Body Frame-Origin-CoM			Point on x axis BF			Point on y axis BF			Euler Angles (X-Y-Z) Order			Axis Direction Vector			Rot Ang
Bodies ID	x	y	z	x	y	z	x	y	z	roll	pitch	yaw	u1	u2	u3	[Rad]
Absolute Frame	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Spring/Damper Supp	0	500	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Base	0	545,5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Rod Amarelo(Adams)	-838,57	517,397	0	0	0	0	0	0	0	0	0	0	0	0	1	0,5236
Rod Rosa (Adams)	838,57	517,40	0,00	0	0	0	0	0	0	0	0	0	0	0	1	-0,5236

Figure 3.9: *Excel* table for the inputs needed to define the position of the bodies.

The first input type uses the three coplanar points method detailed in Sec. 2.2.2. The user defines two vectors in two of the body's principal axes which are then used to obtain the third vector of the local frame by applying the cross product. The unit vectors of these three vectors can afterwards be used to form the rotation matrix.

The second input type uses the sequence of three successive rotations known as Bryant angles, described in Sec. 2.2.2. In this input the user defines the magnitude of each rotation which then is used to calculate the rotation matrix through Eq.(2.12).

However, as stated before these input methods can be practical in some specific cases as inputs but cannot be used during the calculus of the kinematic or dynamic simulations due to the presence of singularities. Thus, using the entries of the calculated rotation matrices, these two inputs are transformed into their respective Euler parameters.

Firstly, Eq.(3.1) is used to calculate the value of e_0. If this Euler parameter is found to be non-zero, then the set of Eqs. (3.2), (3.3) and (3.4) can be used to calculate the remaining three Euler parameters.

$$e_0^2 = \frac{trA + 1}{4} \tag{3.1}$$

$$e_1 = \frac{a_{32} - a_{23}}{4e_0} \tag{3.2}$$

$$e_2 = \frac{a_{13} - a_{31}}{4e_0} \tag{3.3}$$

$$e_1 = \frac{a_{21} - a_{12}}{4e_0} \tag{3.4}$$

If the e_0 is found to be zero, then the set of Eqs. (3.5), (3.6) and (3.7) must be used to calculate the remaining Euler parameters, where trA is the trace of the rotation matrix and the a_{ij} are the matrices entries of Eq.(2.12).

$$e_1^2 = \frac{1 + 2a_{11} - trA}{4} \tag{3.5}$$

$$e_2^2 = \frac{1 + 2a_{22} - trA}{4} \tag{3.6}$$

$$e_3^2 = \frac{1 + 2a_{33} - trA}{4} \tag{3.7}$$

The third input type yields the literal application of the Euler theorem discussed in Sec. 2.2.2, as the user defines the unit vector of the orientational rotation axis and the magnitude of the rotation. The program then makes use of Eqs. (2.13) and (2.14), to acquire the body's Euler parameters. For both input types 2 and 3, the program assumes that the input of a positive angle corresponds to a counter-clockwise rotation.

If the selected simulation is dynamic, the body inputs also comprise the inertial characteristics, initial velocities, initial forces and torques. The inertial characteristics are composed of the bodies mass and moments of inertia. As stated in Sec. 2.2.2 for this work, it is assumed that the local body frame axes are coincident with the principal axes of inertia of the body. Thus the inputted moments of inertia are assumed to be the principal moments of inertia of the respective body defined about the local frame, which can then be transformed to the global frame using Eq.(3.8).

$$\mathbf{J} = AJ^{'}A^T \tag{3.8}$$

Finally, it is important to reference that the forces and torques that actuate on the body's can be inputted, in Excel, as constant forces or as functions, using function handles that define anonym functions

"*'@(t) function*".

Joints Sheet

For the joints sheet, it is requested that the user identifies the bodies restricted by the joints, defines the global position of the joints, $\vec{r}_P$, and the joints specific auxiliary vectors, such as $\vec{s}_i$ of Fig. 2.3, needed to fully define the joints constraint equation presented in Sec. 2.3. With these vectors and the body position vector, $\vec{r}$ of Fig. 3.10, the body global vectors are determined through the sum of vectors as illustrated by Eq.(3.9).

$$\vec{r}^P = \vec{r} + \vec{s}^P \tag{3.9}$$

Lastly, it is also requested of the user to identify and define the global position of the points of interest of the simulation, $\vec{r}_{PoI}$.

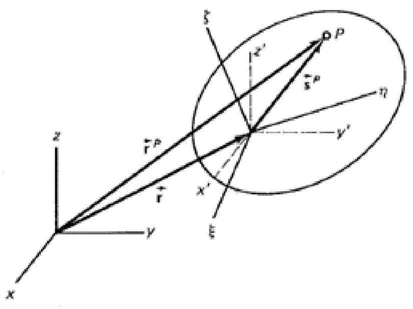


Figure 3.10: Definition of the global position of point P using vectors [6].

Driving Inputs Sheet

Similarly to the joints sheet, in the driving inputs sheet the data needed to define the driving constraint equations, detailed in Sec. 2.3.2, is requested. The user must indicate the body to which the driving input is applied, and its direction to identify if it is a translational input or a rotational input. If the value is lower than 3 it is a translational input, if it is higher then it is a rotational input.

Depending on the direction of the input, the program uses the defined function and the input direction vector $\vec{u}$ to calculate the translational displacement, as illustrated for the x component in Eq.(3.10), or the angular rotation, as illustrated for the rotation around the x component in Eq.(3.11).

$$\Phi(n,1) = x - u_x \cdot f(t_i) \tag{3.10}$$

$$\Phi(n,1) = e_1 - u_x \cdot sin(\frac{f(t_i)}{2}) \tag{3.11}$$

As discussed in Sec. 2.3.2, when driving inputs are present it is necessary to also compute the right-hand side components of the velocity and acceleration problems. As a result, it is also requested of the user to define the first and second derivative of the driving equation as an anonymous function handle "*'@(t) function*", that is then used to calculate the magnitude of the velocity and acceleration input.

If the input is rotational then the calculated angular velocities and accelerations are converted to Euler Parameters using the Eqs. (2.19) and (2.20) for the angular velocities, and Eqs. (2.21) and (2.22) for the accelerations.

Force Elements Sheet

The force elements sheet is only required to be filled in if the simulation is dynamic. In this case, the user must identify the bodies that are connected by the force elements and the position vectors of the connection points, $\vec{r}_i^P$ and $\vec{r}_j^P$ illustrated in Fig. 3.11. In the special case of the torsional spring, the position vectors to define are $\vec{s}_i$ and $\vec{s}_j$ as depicted by Fig. 2.6.

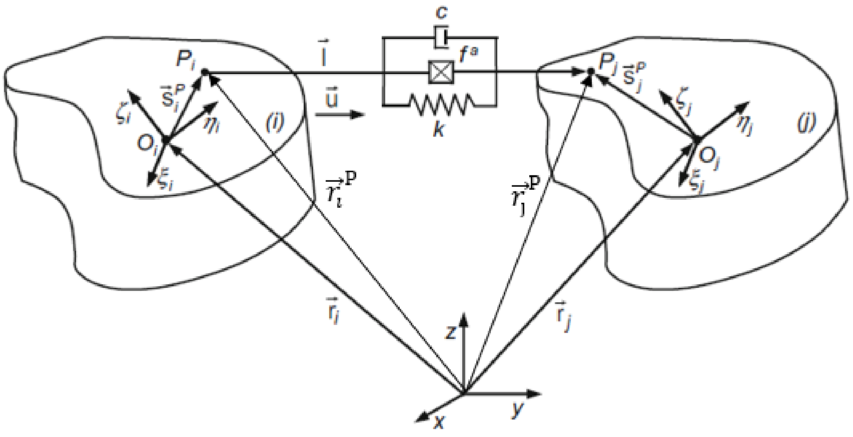


Figure 3.11: Definition of the global position of the force elements connection points using vectors [10].

If the force elements are linear, the user must also provide the spring stiffness, linear damping coefficient or the torsional constant for the translational spring, translational damper, and torsional spring respectively. However, if the simulation requires the modelling of non-linear and even discontinuous springs, dampers, or actuators the user needs to provide the force functions in an anonymous string handle, "*'@(t) function(t)*", as well as the number of functions and the function interval.

If the number of functions is equal to three the program assumes that the first function is valid between $]-\infty, x_1[$, the second function is valid between $[x_1, x_2]$ and the third function is valid between $]x_2, +\infty[$. If the user chooses to define only two functions, the program will use the value of x_1 to define the interval between both function, in other words, the first function is valid between $]-\infty, x_1[$ and the second function is valid between $[x_1, +\infty[$.

Finally, if only one function is defined, the program will not make use of the defined interval and will assume that the function is valid between $]-\infty, +\infty[$.

Post Processing Sheet

The last sheet is responsible for the vehicle dynamics post-processing, it is only called in the program if the user requests it, and this main function is to attribute a specific *MATLAB* ID, that corresponds to a prototype part, to the points of interest defined in the joints sheet allowing the program to identify the points needed for the post-processing calculations.

3.2.1 Kinematic Analysis Pseudo-Code

In Sec. 2.4 the problem of the Kinematic Analysis was introduced, stating that the analysis is composed of the resolution of three sub-problems as summarised in Fig. 2.5. The transformation of this flowchart into an algorithm is straightforward and a standard general purposed algorithm has been presented by several authors, such as Nikravesh [6] and Paulo Flores et al [10] and is presented in Alg. 3.1. This algorithm is the base of the kinematic analysis of this work's general purpose tool, and thus its equivalent pseudo-code is represented on the program's overall pseudo-code of Fig. 3.5, and its numerical methods are discussed in the following sections.

Algorithm 3.1: *Appending driving constraints kinematic algorithm.*

```
1: Set a time step counter $i$ to $i = 0$ and initialise $t^i$ = $t^0$ (initial time).
2: Append k driving equations to the constraint equations.
3: for $t_i \in [t_0, t_f]$ do
4:     Solve $\Phi(q,t) = 0$ for $q$.
5:     Update the bodies coordinates $q$.
6:     Solve $\Phi_q \dot{q} = v$ for $\dot{q}$.
7:     Update the bodies velocities $\dot{q}$.
8:     Solve $\Phi_q \ddot{q} = \gamma$ for $\ddot{q}$.
9:     Update the bodies accelerations $\ddot{q}$.
10:    Storage the values of $q$, $\dot{q}$ and $\ddot{q}$ for $t_i$.
11:    if $t_i = t_f$ then
12:        End simulation and post-process the attained results.
13:    else if $t_i \neq t_f$ then
14:        Increment $t^i$ to $t^{i+1}$, and go to (4).
15:    end if
16: end for
```

Newton-Raphson and Steepest Descent

As stated in Sec. 2.4, the sub-problem of the finite displacements consists in finding the new positions that the system takes when a finite displacement is applied to an input element [19], which mathematically is equivalent to finding the roots of the non-linear system of Eq.(2.47), $\Phi(q)$.

According to Conte and Boor [28], the problem of finding the roots of a non-linear vector-valued function, $f(x)$, can only be solved by resorting to computational techniques that obtain approximate solutions. Within these techniques, it is possible to use optimisation algorithms, that will lead to a system of equations represented in Eq.(3.12), that can be mathematically manipulated to provide the solution to the roots problem if it assumes the specific function of Eq.(3.13), which every minimum is a solution of equation $f(x) = 0$.

$$\nabla F(x) = 0 \tag{3.12}$$

$$\nabla F = 2(f^{'})^{T} f \ \ , \ with \ \ f^{'} = \frac{\partial f}{\partial x_h} \tag{3.13}$$

Standard optimisation techniques are usually based on the simpler steepest descent and Newton-Raphson methods that yield, iteratively, an approximation of the minimum x^* of a continuously differentiable function, f, given an initial guess, x_0 [29]. The steepest descent method, consists on searching for a minimum nearest to x along the direction of the gradient, $\nabla f(x)$, until a local minimiser, $t^* > 0$, is found that satisfies Eq.(3.14). When this condition is satisfied, the next iteration takes $x^{(m+1)} = x_0 - t * \nabla F(x)$ as the next approximation of x^*, guaranteeing that the steepest descented method always decreases the function value and makes progress towards the minimiser, x^* [28, 30–32].

$$F(x^{(m+1)}) < F(x^m) \tag{3.14}$$

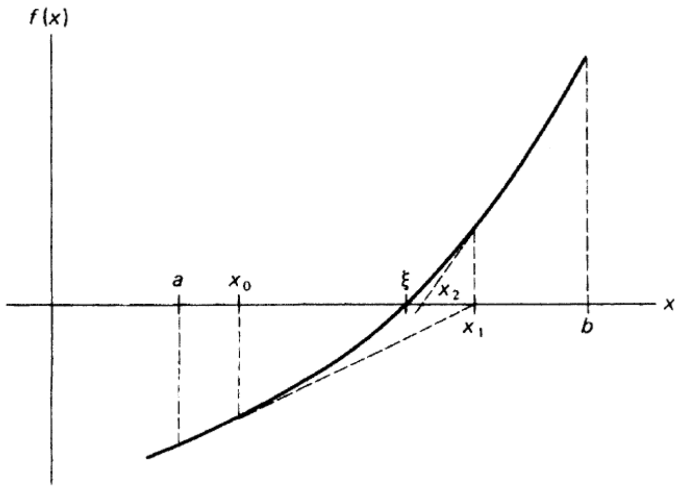


Figure 3.12: Newton Convergence [28].

The Newton-Raphson method takes the tangent to the initial guess, x_0, and solves it to its root, which then forms the new approximation to x^*, as observable in the Fig. 3.12 [28]. This is usually the first method to be attempted when trying to solve an optimisation problem since its iteration function, Eq.(3.15), provides a quadratic convergence in the neighbourhood of the solution, alongside a high computational efficiency. However, it is not always guaranteed that the Newton-Raphson method converges to the desired root, and Conte [28] remarks that Newton's method will frequently diverge unless an initial approximation is carefully selected.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{3.15}$$

As stated before, most of the modern algorithms are able to combine the Newton-Raphson, and the steepest descent method, that is guaranteed to converge but slowly. *MATLAB* itself has three optimisation algorithms already implemented in a built-in function, *fsolver*, of these algorithms only the Trust-Region-Dogleg algorithm and Levenberg-Marquardt algorithm try to combine the methods described above or a variation of them.

Levenberg-Marquardt and Trust Region Dogleg Algorithm

Whilst the Levenberg-Marquardt algorithm can be thought of as purely the combination of the steepest descent method and the Gauss-Newton method, which is the application of the Newton-Raphson method to the least-squares problem, the Dogleg algorithm is the application of its combination alongside the trust region method.

According to Manolis [33], Levenberg-Marquardt tries to solve a variation of the normal equations, also called the least square cost function, represented in Eq.(3.16) to find the step, δ_P, that minimises $||x - f(p) - J\delta_P||$.

$$N\delta_P = J^T\epsilon, \; with \; N \equiv J^TJ + \mu I \; and \; \mu > 0 \tag{3.16}$$

In Eq.(3.16), μ represents the damping of the Levenberg-Marquardt algorithm, responsible for alternating between the steepest descent method and the Gauss-Newton method based on the reduction of the squared distance, $\epsilon^T\epsilon$, where $\epsilon = x - x^{'}$ and $x^{'} = f(p)$. If the squared distance is not reduced when solving Eq.(3.16), the damping term is increased, and the process is repeated during the same iteration using the steepest descent method until a reduction is achieved. However, if the damping term is decreased and assumes a small value, the Levenberg-Marquardt algorithm uses the Gauss-Newton method, ensuring that the algorithm is always capable of reducing the squared distance efficiently [33, 34].

The combination of the Gauss-Newton and the steepest descent method in the Dogleg algorithm is controlled not by a damping term, but through the use of a trust-region. The algorithm seeks to minimise a quadratic model function, L, which behaves like the objective function, f, over the trust region whose radius is chosen based on the quality of the approximation of the model function to f in previous iterations. In other words, if the model accurately predicts the behaviour of f, the trust region radius is increased and longer steps are tested. If it fails to predict the function behaviour the radius of the trust region is reduced [33].

Assuming the trust regions has a radius of Δ, the Dogleg method selects the update step, δ_k, based on the length of the different steps. If the Gauss-Newton step, δ_{gn}, is within the trust region radius, the step is assumed to be the Gauss-Newton step, $\delta_{gn} = \delta_{gn}$. In case the Cauchy point, which is the point lying on the gradient direction that minimises the quadratic model, is outside the trust region, then the new step is assumed to be the intersection between the steepest descent direction and the trust region boundary. Additionally, if the Cauchy point is inside the trust region, then the new step is assumed to be the intersection of the line between the Cauchy point and the Gauss-Newton step, forming the Dogleg step shown in Fig. 3.13 [35].

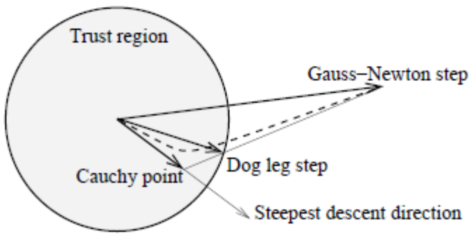


Figure 3.13: Dogleg approximation of the curved optimal trajectory [33].

Algorithm Choice

According to Manolis [33] and Goharian [36], the trust-region Dogleg algorithm has a higher computational efficiency than the Levenberg-Marquardt method, although both of them are Newton step based methods with a quadratic speed of convergence near x^* [37].

This higher computational efficiency is obtained because the Levenberg-Marquardt step always requires the augmented normal equations to be solved, which means that every update to the damping term calls for a new solution of the normal equations and thus the realisation of an inefficient step.

In contrast, the Dogleg algorithm always reaches a new step through the intersection of the trust region boundary, meaning that it does not need to find the Gauss-Newton step in the case of failure and thus the number of times that the algorithm has to solve the Gauss-Newton step is reduced [36].

Knowing that the formulation of the finite displacement problem posed in this work can result in large-scale and sparse systems, due to the lack of interaction between the different joints, it is possible to draw parallels between this problem and the Bundle Adjustment problem addressed by Manolis [33].

As a result, it is feasible to infer that the trust-region Dogleg algorithm produces solutions that are of the same quality when compared to the Levenberg-Marquardt algorithm, but at a fraction of the time as outlined in Tab. 3.1 [33, 36].

Model	Levenberg-Marquardt [s]	Trust Region Dogleg [s]
Slider-Crank	0.2409	0.1897

Table 3.1: Comparison of the efficiency, through the simulation time of the developed general purposed tool, between the Levenberg-Marquardt and Trust Region Dogleg algorithms.

3.2.2 Dynamic Analysis Pseudo-Code

In Sec. 2.5.3, an overall view of the methods to solve the generic forward dynamic problem considered in this work were presented. Of those the methods of stabilisation can heavily influence the transcription of the flowchart, Fig. 2.7, to an algorithm that can be implemented in *MATLAB*. Therefore, it is important to mention that the following Alg. 3.2 only corresponds to the implementation of the ALF.

Algorithm 3.2: *Augmented Lagrangian Formula dynamic algorithm [10, 38].*

```
1: Specify initial conditions for positions $q_0$ and velocities $\dot{q}_0$ and perform a consistency check using the Inertia-Weighted correction method to ensure that the initial conditions fulfil the position and velocity constraints, as indicated in Eqs. (2.47) and (2.43).
2: Start the initial value problem solver, ode113, that uses the Variable-Step, Variable-Order (VSVO) Adams-Bashforth-Moulton predictor-corrector method.
3: Assemble the global mass matrix $M$, evaluate the Jacobian matrix $\Phi_q$, construct the constraint equations $\Phi$, determine the right-hand side of the velocities $\nu$ and accelerations $\gamma$, and calculate the force vector $g$.
4: Calculate the initial estimative for the iterative scheme of the ALF, $M\ddot{q} = g$.
5: Use the iterative ALF method to solve the linear set of equations of motion to determine the generalised accelerations $\ddot{q}$, using the generalised coordinates $q$ and the generalised velocities $\dot{q}$ of the current step $t_i$.
6: Assemble the vector $\dot{y}_t$ with the generalised velocities $\dot{q}$ and accelerations $\ddot{q}$ of the current time step $t_i$, to perform the numerical integration routine that solves the first-order initial value problem for the next time step $t_i + \Delta t$.
7: Extract and update, from the newly calculated vector $y_{t+\Delta t}$, the values of the generalised positions $q$ and velocities for the time step $t_{i+1} = t + \Delta t$.
8: Update the time variable and proceed to the new time step $t_{i+1}$, by repeating points 2, 3, 4, 5 and 6 until the final time is reached and then exit the ode113 solver.
9: With the updated values of the generalised coordinates $q$ and velocities $\dot{q}$, solve the equations of motion for each time step $t_i$, to obtain the corrected generalised accelerations $\ddot{q}$.
```

The ALF Alg. 3.2 introduced the methods used in this work to solve the forward dynamic problem. These methods and their choice will be further discussed in the rest of Chapter 3.

Initial Condition Consistency Check

In forward dynamics, the generalised positions q^i and velocities $\dot{q}^i$ are determined through the numerical integration of the correspondent time accelerations $\ddot{q}^i$ without resorting to Eqs. (2.43) and (2.47). As a result, it is usual that these variables contain numerical errors that cannot be eliminated which then lead to the violation of the kinematic constraints.

One of the crucial measures to avoid the propagation of these errors is to start the integration process with a set of initial conditions that do not violate the kinematic constraints, Eqs. (2.43) and (2.47). However, in complex systems, it is usual that only estimated values for the coordinates and velocities can be given as input, q^e and $\dot{q}^e$, and therefore it is usual that a pre-processing step is taken to adjust these estimates.

In this work, two methods are considered for the correction of the initial conditions and were studied in detail by Nikravesh [26]. The first method treats the initial conditions correction problem as the analysis of a kinematic system. The second method is based on the minimisation of the sum-of-squares of the adjustments made to the coordinates and velocities.

In the kinematic analysis method, the coordinates and velocities are divided into dependent, q_d, and independent variables, q_{id}. The independent variables are picked by the user and assumed constant for the calculation of the remaining unknown variables, done through Eqs. (3.17) and (3.18), to produce a system configuration that is constraint compliant. In Eq.(3.18), I_{id} represents an $n_{dof} \times n_{dof}$ identity matrix.

$$\Phi(q_d, q^e_{id}) = 0 \tag{3.17}$$

$$\begin{bmatrix} \Phi_{q(d)} & \Phi_{q(id)} \\ 0 & I_{id} \end{bmatrix} \begin{bmatrix} \dot{q}_d \\ \dot{q}_{id} \end{bmatrix} = \begin{bmatrix} 0 \\ \dot{q}^e_{id} \end{bmatrix} \tag{3.18}$$

In contrast, the objective of the second method is to find a set of $\Delta's$, Eqs. (3.19) and (3.20), that adjusts the estimated variables in a way that the corresponding sum-of-squares of the corrections is minimised, whilst the position and velocity constraints are respected. For this adjustment, all the variables can be adjusted equally if no weighting factors are used or it can be influenced if the forward dynamic mass matrix is used as a weighting factor, also known as inertia weighted correction.

$$q^0 = q^e + \Delta q \tag{3.19}$$

$$\dot{q}^0 = \dot{q}^e + \Delta v \tag{3.20}$$

The implementation of this method is straightforward: the determination of the corrective Δ for the position variables is done through the use of a Newton-Raphson process, which is detailed in Alg. 3.3, due to its equations non-linearity. The velocity analysis $\Delta's$ are found through Eqs. (3.21) and (3.22) after the position adjustment.

$$\varepsilon = \Phi_q \dot{q}^e \tag{3.21}$$

$$\Delta\dot{q} = -M^{-1}\Phi_q^T(\Phi_q M^{-1}\Phi_q^T)^{-1}\varepsilon \tag{3.22}$$

Since the mass matrix and Jacobian used for the inertia weighted correction are the same as the ones used in forward dynamics, the implementation of this method requires less programming effort than readapting the Jacobian to solve the system of Eq.(3.18). This is one of the shortcomings inherent of the kinematic analysis correction method, alongside the selection of the independent coordinates [26]. This selection, if improperly made may lead to the program not yielding a solution or yielding a solution far from the desired initial configuration of the system.

For these reasons, and because the inertia weighted correction produces better adjustments for rotational coordinates in space, as illustrated in Fig. (3.14), it is the method chosen and implemented in this work [26].

Algorithm 3.3: *Newton-Raphson inertia weighted correction algorithm [26].*

```
1: while $q^{i+1} - q^i \leq$ tolerance do
2:     Evaluate $\sigma^i = \Phi(q^i)$.
3:     Compute $\Delta q^i = -\Phi_q^{-1}\Phi_q^T(\Phi_q M^{-1}\Phi_q^T)^{-1}\sigma$.
4:     Correct $q^{i+1} = q^i + \Delta q^i$.
5:     Evaluate $\Delta = q^{i+1} - q^i$.
6:     Set $i = i + 1$.
7:     Update the body positions with the new values $q = q^{i+1}$.
8: end while
```

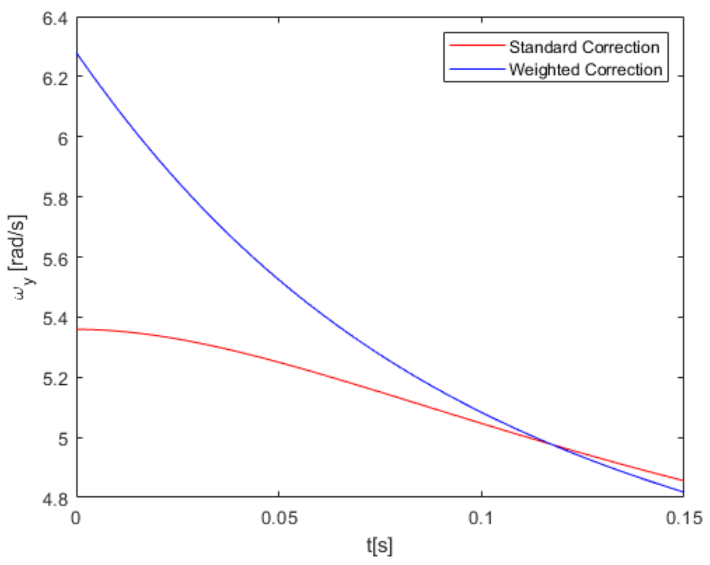


Figure 3.14: Comparison of the adjusted values of the flyball governor model with initial velocity $\omega_y = 2\pi$ rad/s for the standard versus the Inertia weighted correction.

Initial Value Problem

The system of Eq.(2.64), that describes the forward dynamic analysis problem, is a system of DAEs of index one, which can be transformed into a system of second-order Ordinary Differential Equation (ODE), once the vector of Lagrange multipliers is eliminated from the system of equations [38].

This is advantageous since the ODEs are more amenable to integration than a system of DAEs and the characteristics of its numerical methods such as stability, convergence, and accuracy have been studied in more detail than the DAEs numerical methods [19]. For the case of first-order ODEs, several efficient integration algorithms can be used to compute their solution for specified initial conditions [39], and Nikravesh [6] showed that a second-order ODE, Eq.(3.23), can be transformed into a first-order ODE system if a set of new variables is written as exemplified through Eqs. (3.24) and (3.25).

$$\ddot{y}_1 = f(y_1, \dot{y}_1, t) \tag{3.23}$$

$$\dot{y}_1 = y_2 \tag{3.24}$$

$$\dot{y}_2 = f(y_1, y_2, t) \tag{3.25}$$

This process is applied to the equations of motion by defining the vector $\dot{y}$ to contain the generalised velocities and accelerations for the actual time step and the vector y to contain the generalised coordinates and velocities that result from the integration of $\dot{y}$ for the next time step, $t + \Delta t$, as presented in Eqs. (3.26) and (3.27).

$$\dot{y}_t = \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix}_t \tag{3.26}$$

$$y_{t+\Delta t} = \begin{bmatrix} q \\ \dot{q} \end{bmatrix}_{t+\Delta t} \tag{3.27}$$

With the first order ODE defined, the starting generalised coordinates and velocities can be used to transform the dynamic analysis problem from an ODE with infinite solution to a well-posed initial value problem with only one solution, which can be solved through a numerical integration algorithm [6].

These numerical methods do not attain analytical solutions but approximate solutions at discrete times, $(t_1, ...t_n)$, whose time step, $h = t^{(i+1)} - t^i$, can be constant or variable. The approximate solution at each point t^i will present a total error, $e = |y(t^i) - y(i)|$, where $y(t^i)$ is the exact solution and $y(i)$ is the approximate solution, composed by the truncation error, which depends on the nature of the numerical algorithm, and the round-off errors [6].

There are two basic approaches on which these algorithms are based: the Taylor series expansion and the polynomial approximation. The approximation of the solution through the Taylor series expansion, represented by Eq.(3.28), implies that the derivative terms of $f(y^i, t^i)$ with respect to t^i, that need to be evaluated, increase with the order of the algorithm. This in turn is extremely costly and implies that only lower-order Taylor series algorithms can be used [6].

$$y(t^{(i+1)}) = y(t^i) + hf(y^i, t^i) + \frac{h^2}{2!}\dot{f}(y^i, t^i) + higher - order\ terms \tag{3.28}$$

The Runge-Kutta algorithms introduced in Eq.(3.29) avoid the Taylor series' need to evaluate higher order derivatives of f. Instead, they approximate the next step value, $y_{(n+1)}$, at the same order of accuracy as the Taylor series, by using the present value y_n and the sum of the weighted average of i increments, $(k_1, ..., k_i)$, where its coefficients are usually represented in the extended Butcher tableau shown in Tab. 3.2 [6, 40].

$$
\begin{aligned}
y_{(n+1)} &= y_n + h\sum_{i=1} b_i k_i \\
k_1 &= f(t_n, y_n), \\
k_2 &= f(t_n + c_2 h, y_n + h(a_{21}k_1)), \\
k_3 &= f(t_n + c_3 h, y_n + h(a_{31}k_1 + a_{32}k_2)), \\
&\;\;... \\
k_i &= f(t_n + c_i h, y_n + h\sum_{j=1}^{i-1} a_{ij}k_j)
\end{aligned}
\tag{3.29}
$$

0					
c_2	a_{21}				
c_3	a_{31}	a_{32}			
...	...	...	...		
c_s	a_{s1}	a_{s2}	...	$a_{s,s-1}$	
	b_1	b_2	...	b_{s-1}	b_s
	b_1^*	b_2^*	...	b_{s-1}^*	b_s^*

Table 3.2: Extended generic Butcher tableau.

This simplification allows for the use of higher order algorithms such as *MATLAB ode45*, which is an explicit Runge-Kutta $(4, 5)$ formula based on the Dormand-Prince pair. The Dormand-Prince pair is an adaptative method that uses two methods of a different order, one of the fourth and the other of the fifth order, to produce an approximate solution, $y_{(n+1)}$, and an estimate of the local truncation error using Eq.(3.30), where $y^*_{(n+1)}$ is the solution of the lower order method. The error estimation, $e_{(n+1)}$, is then used to adapt the step size of the integration. If the error is higher than a user-defined threshold, the current step is repeated with a lower step size. If the error is smaller than the threshold, the step size is increased for the next step to save time [41, 42].

$$
e_{(n+1)} = y_{(n+1)} - y^*_{(n+1)} = h\sum_{s}^{i=1}(b_i - b_i^*)k_i \tag{3.30}
$$

The methods mentioned above, based on the Taylor series approximation, are single step methods that only require information on the current time step, (t_i, y_n), to approximate the solution of the next step, $(t_{(i+1)}, y_{(n+1)})$, and therefore require a minimum amount of storage. On the other hand, the polynomial approximation methods are multistep methods that make use of previous steps information, $(t_{(n-p)}, y_{(n-p)})$, to produce an approximation of the next time step, $(t_{(i+1)}, y_{(n+1)})$, requiring a higher amount of storage [19].

The general equation of the multistep methods is represented by Eq.(3.31), which uses the Newton backward difference formula, Eq.(3.32), to approximate the integration of the function $f(t, y)$ with its values at previous time steps, resulting in a series of methods that are defined by its α and β coefficients.

If $\beta_0 = 0$, the value of $f(t_{(n+1)}, y_{(n+1)})$ is not included and the method is explicit, also known as Adams-Basforth algorithms. If $\beta_0 \neq 0$, the value of $f(t_{(n+1)}, y_{(n+1)})$ is included and the method is implicit, also known as Adams-Moulton algorithms, and, as a result, more accurate and stable [6, 19, 28].

$$y_{(n+1)} = y_{(n-p)} + \int_{t_{(n-p)}}^{t_{(n+1)}} f(t, y)\, dt \tag{3.31}$$

$$\sum_{i=0}^{p+1} \alpha_i y_{(n+1-i)} + \sum_{i=0}^{k} \Delta t \beta_i f(t_{(n+1-i)}, y_{(n+1-i)}) \tag{3.32}$$

As an example of the application of the multistep generic Eqs. (3.31) and (3.32), the Adams-Bashforth and Adams-Moulton fourth order equations are represented by Eqs. (3.33) and (3.34), alongside with its correspondent error estimates, Eqs. (3.35) and (3.36). From the error estimate constants, it is possible to infer that the latter method is more accurate, although it is not self-starting since it requires an estimate of the value of $f_{(n+1)}$. This implies the use of an explicit method of the same order, like the Adams-Bashforth, to obtain a prediction of $f_{(n+1)}$, creating a predictor-corrector algorithm.

$$y_{(n+1)} = y_n + \frac{\Delta t}{24}(55f_n - 59f_{(n-1)} + 37f_{(n-2)} - 9f_{(n-3)}) \tag{3.33}$$

$$y_{(n+1)} = y_n + \frac{\Delta t}{24}(9f_{(n+1)} + 19f_n - 5f_{(n-1)} + f_{(n-2)}) \tag{3.34}$$

$$E = \frac{251}{720} \Delta t^5 f^{IV}(\xi) \tag{3.35}$$

$$E = -\frac{19}{720} \Delta t^5 f^{IV}(\xi) \tag{3.36}$$

MATLAB presents the option of Variable-Step, Variable-Order (VSVO) predictor-corrector *ode113* iterative algorithm. This method, depicted in Alg. 3.4, is composed of two methods of the same order, the explicit Adams-Bashforth and the more accurate implicit Adams-Moulton method. The first method also known as predictor-step is used to calculate the initial approximation, $y^0_{(n+1)}$, which is then used to initiate the Adams-Moulton method, the corrector-step.

The *ode45* and *ode113* algorithms can both be used to solve the equations of motion of MBSs, which are usually composed of large matrices and therefore are expensive to evaluate. As mentioned before, *ode45* is a single-step, explicit method and thus self-starting, that requires a minimum amount of storage since all of its several function evaluations of f are done at the current time step, t_n. In addition, the error estimation to adjust its time step, Eq.(3.30), requires the use of a higher order method, which implies an extra evaluation of function f, and its fixed order nature will limit the range of accuracies to which the program will be efficient [6, 19].

In contrast, with the appropriate time step, the function evaluations of the iterative Adams-Bashforth-Moulton method, *ode113*, can be kept below three evaluations per time step, but the method requires a higher amount of storage than *ode45*, since it has to store the value of previous step evaluations,

Algorithm 3.4: *Predictor-corrector algorithm [19].*

```
1: Use the explicit method (predictor) to obtain $y^0_{(n+1)}$.
2: while $\frac{y^k_{(n+1)} - y^{k-1}_{(n+1)}}{y^k_{(n+1)}} \leq$ tolerance do
3:     Use the implicit method (corrector) to determine the successive $y^k_{(n+1)}$, $k = 1, 2, ...$
4:     Evaluate $\frac{y^k_{(n+1)} - y^{k-1}_{(n+1)}}{y^k_{(n+1)}}$.
5:     if $\frac{y^k_{(n+1)} - y^{k-1}_{(n+1)}}{y^k_{(n+1)}} \leq$ tolerance then
6:         Go to next step $t_{i+1} = t_i + \Delta t$.
7:     else if $\frac{y^k_{(n+1)} - y^{k-1}_{(n+1)}}{y^k_{(n+1)}} \geq$ tolerance then
8:         Set a new $k = k + 1$ and calculate a new $y^k_{n+1}$.
9:     end if
10: end while
```

$(f_{(n-1)}, f_{(n-2)}, ...)$. By using the previous points information, *ode113* can automatically select the proper order and the proper time step size, which allows easier control of the propagation of both truncation and round-off errors and will maximize the range of accuracy requests in which the algorithm is efficient [6, 19, 20].

With the information previously presented and the knowledge that the total computational cost of the numerical integration process is mostly due to the number of function evaluations at each time step, (t_n), it is possible to infer that although no method will perform uniformly better than another method on all problems, *ode113* presents the most efficient option for the integration of the equations of motion and it will be the algorithm used in this work [19, 43]. These statements are supported by Lankarani [20] and Conte and Boor [28], who infer that, on some problems, the Runge-Kutta methods require almost twice as much computing time as the predictor-corrector algorithms, as can be inferred by Tab. 3.3.

Model	Adams-Basforth-Moulton [s]	Runge-Kutta Dormand-Prince Pair [s]
Flyball governor	124.34	355.37
Flyball governor simplified	59.79	109,49
Slider-Crank	26.26	68
Simple Pendulum	20.53	52.67

Table 3.3: Comparison of the efficiency, through the simulation time, between the Adams-Bashforth-Moulton (*ode113*) and Runge-Kutta Dormand-Prince Pair (*ode45*) for the different models that will be presented in the next sections.

Finally, it is necessary to refer that both the explicit Runge-Kutta, *ode45*, and the predictor-corrector, *ode113*, methods are conditionally stable, and the time step must be chosen depending on the characteristics of the problem, at the risk of the algorithm becoming unstable and a solution not being obtained [19, 38]. One of the cases where this may happen is when the system is determined to be stiff. A stiff system is referred to as an initial value problem in which the complete solution consists of fast and slow components, meaning that the ODE varies slowly in time but also has rapidly varying solutions, asymptotes [20].

In MBSs, the stiffness can be produced by components with large differences in their masses, stiffness, and damping, or it can be numerically induced due to a large number of components and equations of motion, or sudden or accumulated violations in the constraint conditions. For these cases, it is recommended to use a stiffly stable algorithm such as *ode15s*, which is a VSVO multistep solver based on the numerical differentiation formulas.

Stabilisation of Constraint Equations

As stated in Sec. 2.5.3, the determination of the generalised positions and velocities in the forward dynamics problem is done through the integration of the correspondent time step acceleration. During this integration, the position and velocity constraint, Eqs. (2.34) and (2.35), are not directly used and therefore violations of the constraints at the position and velocity levels will occur due to the round-off and truncation errors inherent to the integrations [44].

As such, it is necessary to implement, in any general-purpose multibody tool, methods that are capable of keeping such errors under control. These methods fall into three categories: constraint stabilisation algorithms, coordinate partitioning methods and direct correct formulations. Of these, the coordinate partitioning methods were excluded from consideration, despite being able to maintain good error control. This is due to the fact that early in this work, it was determined that the general-purpose tool would be developed following the appended driving constraint methodology, due to the poor numerical efficiency of the coordinate partitioning method for large systems [6, 20, 44].

The constraint stabilisation approaches can be divided into two methods, the Baumgarte stabilisation and the penalty formulations presented by De Jalon and Bayo [19]. The Baumgarte stabilisation is one of the simplest methods to stabilise the constraint violations. It applies the feedback control theory to the Lagrange's multiplier method, Eq.(2.64), by modifying the kinematic acceleration equations. These equations, represented in Eq.(2.36), are directly present in the second line of the system of Eq.(2.64), which implies that when the integration is performed the differential equation, Eq.(2.36), is also being integrated with respect to time [19, 20].

In this case, the general solution of the differential equation Eq.(3.37), is unstable for any constant vector, a_1, different than zero, which can be induced by the integration errors. This, will lead to an unbounded increase of the solution with time, and as a consequence to the violation of the position constraints of Eq.(2.34) [6, 19].

$$\Phi(q,t) = a_1 t + a_2 \tag{3.37}$$

Nevertheless, this unstable open-loop system, represented in Fig. 3.15, can be transformed into a closed-loop stable system, Fig. 3.16, if the position and velocity constraint violations are feedback to the acceleration problem [19, 20, 44]. This will transform the kinematic acceleration equations into

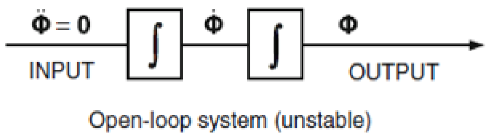


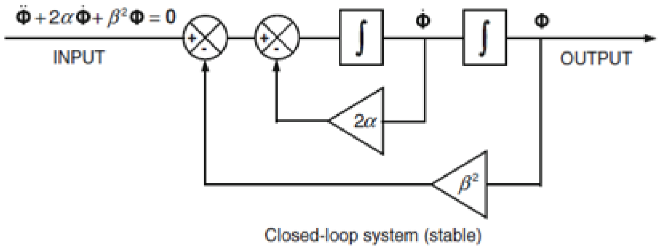
Figure 3.15: Unstable open-loop system that results from the double integration of the accelerations [20].

Eq.(3.38), whose generic solution is presented in Eq.(3.39), and consequently the system of equations of motion is changed into Eq.(3.40).

$$\Phi_q \ddot{q} = \ddot{\Phi} + 2\alpha \dot{Phi} + \beta^2 \Phi = 0 \tag{3.38}$$

$$\Phi = a_1 \exp^{s_1 t} + a_2 \exp^{s_2 t} \ with \ s_1, s_2 = -\alpha \pm \sqrt{\alpha^2 - \beta^2} \tag{3.39}$$

$$\begin{bmatrix} M & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ \lambda \end{bmatrix} = \begin{bmatrix} g \\ \gamma - 2\alpha\dot{\Phi} - \beta^2\Phi \end{bmatrix} \tag{3.40}$$



$\ddot{\Phi} + 2\alpha\dot{\Phi} + \beta^2\Phi = 0$
INPUT
$\dot{\Phi}$
Φ
OUTPUT
2α
β^2
Closed-loop system (stable)

Figure 3.16: Stable closed-loop system that results from the double integration of the accelerations obtained through the modified system of equations of motion from Eq.(3.40) [20].

In both Eqs. (3.37) and (3.39), the variables a_1 and a_2 are constant vectors that only depend on the initial conditions of the problem, and s_1 and s_2 are the roots of the characteristic equations. Through Eq.(3.39), it is possible to infer that if α and β are positive constants, the two roots will have a real negative part and the general solution will be stable and will tend to zero with time, thus controlling the propagation of the constraints violations, as depicted in Fig. 3.17.

Although the Baumgarte stabilisation is simple and numerically efficient it has its disadvantages, mainly its inability of solving the equations of motion when the system is close to singular configurations or redundant constraints are present. In addition, its control parameters are chosen ambiguously through numerical experiments, even if the parameters are chosen such as α is equal to β which causes the system to become critically damped [6, 19, 20, 44].

The penalty formulations are a set of constraint stabilisation methods that are derived from the penalty method presented by De Jalon and Bayo [19]. This method replaces the constrained optimisation problems of system of Eq.(2.64) by an unconstrained one, Eq.(2.62), that converges to the original solution, with an additional term, the penalty function. In MBSs, this corresponds to the elimina-

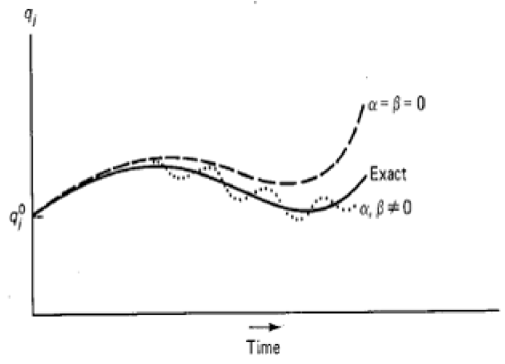


Figure 3.17: Comparison of control of the constraints violation by the Baumgarte stabilisation method with different values of α, β and the exact solution [6].

tion of the Lagrange multipliers from the equations of motion, which consequently reduces the number of equations from $n + m$ to a set of n linear differential equations represented by Eq.(3.41), where $\Phi_q^T \alpha(\ddot{\Phi} + 2\Omega\mu\dot{\Phi} + \Omega^2\Phi)$ represents the forces generated by the penalty system when the constraints are violated, α, Ω and μ are the penalty factors and Q is the unconstrained force vector [19].

$$M\ddot{q} + \Phi_q^T \alpha(\ddot{\Phi} + 2\Omega\mu\dot{\Phi} + \Omega^2\Phi) = Q \Leftrightarrow (M + \Phi_q^T \alpha \Phi_q)\ddot{q} = Q - \Phi_q^T \alpha(\dot{\Phi}_q \dot{q} + \dot{\Phi}_t + 2\Omega\dot{\Phi} + \Omega^2\Phi) \tag{3.41}$$

Similarly to the other constraint stabilisation methods, the penalty method, despite being capable of solving the forward dynamic problems when in the presence of redundant constraints and singular configurations, also suffers the issue of choosing the right penalty number. Numerically, this can be achieved by using large penalty factors. In fact, if $\alpha \to \infty$ the solution of the modified problem coincides with the exact solution of the original problem. However, the use of high values can lead to the ill-conditioning of the problem, which in turn may cause the program to yield an incorrect solution to the dynamic problem [19, 20].

To avoid the numerical ill-conditioning problem, De Jalon and Bayo [19] presented the iterative ALF. This method is a modified penalty formulation that keeps the robustness of the penalty method but seeks to improve the numerical conditioning of the problem and avoid its consequent instability for large penalty factors. Through the addition of an additional corrective term, $\Phi_q \lambda^*$, to the generic penalty formula, Eq.(3.41), resulting in Eq.(3.42) [19, 38].

$$M\ddot{q} + \Phi_q^T \alpha(\ddot{\Phi} + 2\Omega\mu\dot{\Phi} + \Omega^2\Phi) + \Phi_q^T \lambda^* = Q \tag{3.42}$$

By comparing the equation of the constrained system of Eq. (2.64) and (3.42), it is possible to infer that $\beta = \alpha(\ddot{\Phi} + 2\Omega\mu\dot{\Phi} + \Omega^2\Phi) + \lambda^*$, is equal to the Lagrange multipliers in the classic Lagrange method, as illustrated in Eq.(3.43). Therefore, in the limit, when the constraint conditions are satisfied, λ^* must be equal to λ. With these modifications to Eq.(2.64), it becomes possible to solve the equations of motion without the need to use the algebraic equation if the values of λ^* are known.

$$\lambda \cong \lambda^* + \alpha(\ddot{\Phi} + 2\Omega\mu\dot{\Phi} + \Omega^2\Phi) \tag{3.43}$$

It is possible to assume that $\lambda^* = 0$ to start the iterative ALF scheme if the initial conditions in the forward dynamic problem are checked and adjusted to satisfy the position and velocity constraints in Eqs. (2.34) and (2.35). As shown by da Silva, M. [38], this ultimately leads to Eq.(3.44), where $\ddot{q}_0$ is the initial acceleration value determined from Eq.(2.62). Indeed, this approach can be proven to preserve the initial conditions [44].

$$[M + \Phi_q^T \alpha \Phi_q]\ddot{q}_{(i+1)} = M\ddot{q}_i + \Phi_q^T \alpha(\gamma_i - 2\omega\mu(\Phi_q\dot{q}_i - \upsilon_i) - \omega^2\Phi_i) \tag{3.44}$$

This iterative scheme is performed until the difference between the two accelerations complies with a user-specified tolerance, ε, as depicted by Eq.(3.45), and it is the main difference between the ALF and the penalty method, where Eq.(3.41) is only evaluated once [19, 44]. In fact, it is due to the iterative process that the penalty values in the new penalty system of Eq.(3.46) do not need to be large, and in most applications, these assume values in the ranges present in Eq.(3.46). Thus, avoiding the ill-conditioning of the problem induced by large penalty values [19, 38, 45].

$$||\ddot{q}_{(i+1)} - \ddot{q}_i|| = \varepsilon \tag{3.45}$$

$$\beta_i = \alpha[\Phi_q\ddot{q}_{(i+1)} - \gamma_i + 2\omega\mu(\Phi_q\dot{q}_i - \upsilon_i) + \omega^2\Phi_i] \;\; with \;\; \begin{cases} 10^4 \leq \alpha \leq 10^7 \\ 0 \leq \mu \leq 1 \\ 0 \leq \omega \leq 10 \end{cases} \tag{3.46}$$

The extra computational effort needed to implement the iterative scheme is almost insignificant since convergence is usually obtained after two to four iterations, and the leading matrix, $[M + \Phi_q^T\alpha\Phi_q]$, only needs to be evaluated once before the start of the iterative scheme [38].

The last method to be considered for this work is the geometric elimination of the constraint violations, also known as direct correction, developed by Yoon et al. [22, 46]. This method seeks to fully eliminate the position and velocity constraint violations after each integration step, differentiating itself from the constraint stabilisation approaches, which modify the original Eq.(2.62) to control the constraint violations [44].

Therefore, after obtaining the values of the uncorrected positions and velocities, q^u and $\dot{q}^u$ for an initial step, t_i, an iterative Newton Raphson scheme similar to Alg. 3.3, is initiated to determine the corrected positions, q^c, where $q^i = q^u$ and the third step equation is replaced by Eq.(3.47). The values of q^c are then used in the next step, $t_{(i+1)}$, and in the correction of the generalised velocities.

$$q^c = q^u - \Phi_q^T(\Phi_q\Phi_q^T)^{-1}\Phi(q^u) \tag{3.47}$$

For the velocity correction, because its equations are linear, the process only needs to be carried out once for each time step. These values are obtained through Eq.(3.48), where $\dot{\Phi}(q^c, \dot{q}^u)$ is evaluated using the velocity constraint, Eq.(3.49), with $\Phi_q(q^c)$ being the evaluated Jacobian for the new values of q^c [44, 46].

$$q^c = q^u - \Phi_q^T(\Phi_q \Phi_q^T)^{-1}\dot{\Phi}(q^c, \dot{q}^u) \tag{3.48}$$

$$\dot{\Phi}(q^c, \dot{q}^u) = \Phi_q(q^c)\dot{q}^u \tag{3.49}$$

Nevertheless, this method has its disadvantages. Firstly, it uses the classic Lagrange multiplier method to calculate the $\ddot{q}$ values, which will fail in singular system configurations or in the presence of redundant constraints. In contrast, the penalty formulation leading matrix, $[M + \Phi_q \alpha \Phi_q]$, will always be positive definite, even in singular positions, and therefore will always have a valid inverse matrix [45]. Finally, to compute the corrected values it is necessary to determine the Jacobian inverse, for this purpose the direct correction makes use of the Moore-Penrose inverse, $\Phi_q^+ = \Phi_q^T(\Phi_q \Phi_q^T)^{-1}$, which can be numerically instable for ill conditioned problems [44]. This, however, can be mitigated through the use of different regularisation techniques such as the Tikhonov regularisation and the Truncated Single Value Decomposition (TSVD), available in external toolboxes such as *Regtools* developed by C. P. Hansen [47] and *PBSID* regress developed by Houtzager and Gebraad [48].

The methods discussed above have been studied in great detail by Marques et al. [44], where their efficiency and accuracy have been tested in four different models of varying complexity. In the four sets of results presented in the article, the author compares the efficiency, through the computational time ratio, of the different methods with the unity, which is the classical Lagrange multiplier method, as well as the magnitude of the position and velocity constraint violations versus those obtained with the standard method. According to the results, the coordinate partitioning method and direct correction are the best-performing methods for the elimination of constraint violations. However, these occur in a controlled manner at orders of magnitude of 10^{-13} or lower, which can be assumed to be negligible for this work. Therefore, for the stabilisation algorithm choice only the computational efficiency, robustness, and ease of implementation will be considered.

In terms of computational efficiency, it is possible to infer from Tab. 3.4 and Tab. 3.5 that for the first example, a planar four-bar mechanism, the direct correction performs better than the ALF and the Baumgarte stabilisation [44]. However, these perform better for the remaining three models, which are the targeted problems for this general-purpose tool. It is also necessary to note that the efficiency of the Baumgarte and ALF methods is heavily influenced by the penalty values. This is evidenced by the results summarised in Tab. 3.6 which compare the time taken to perform a simulation in function of the penalty parameter α, with $\alpha \in [10^5, 10^7]$, while the remaining integration characteristics remain constant.

The implementation of these algorithms is straightforward. In this aspect, the Baumgarte stabilisation is the easiest to implement since it does not require the use of iterative processes as required by the ALF and direct correction. However, the programming effort of implementing this iterative process is not

	Standard Method	Baumgarte Stabilisation	ALF	Direct Correction
Planar Four Bar Mechanism	1	1.62	1.82	1.21
Bricards Mechanism	1	1.16	1.33	1.28
Slider-Crank	1	1.16	1.30	1.39
Front Suspension	1	1.10	1.17	1.25

Table 3.4: Comparative computational ratios between the different stabilisation algorithms and the standard Lagrange multiplier method [44].

Integration time step	1×10^{-3} s	Penalty - α	1×10^{7}
Integration algorithm	Runge-Kutta - 4th order	Penalty - ω	10
Baumgarte - α,β	5	Penalty - μ	1

Table 3.5: Table with the parameters utilised for the dynamic simulations [44].

Values of α	Simulation time (t_{sim}) [s]
10^5	9399.4
10^6	1195.9
10^7	148.8

Table 3.6: Evolution of the t_{sim} with increasing values of α while $\beta = 20$ and $\mu = 1$.

significant enough to impact the choice of method.

Finally, in terms of robustness and as previously mentioned, both the Baumgarte and the direct correction methods rely on the leading matrix of the classic Lagrange multipliers method, which results in both failing near singular positions. Additionally, the direct correction method also presents numeric instability due to the use of the Moore-Penrose inverse that the ALF does not exhibit. Inversely, the ALF and the Baumgarte stabilisation will suffer from the ambiguity of choosing the penalty values (α, β and μ), which is a significant hindrance but can be mitigated since it is possible to critically damp the Baumgarte control system, $\alpha = \beta$, and to restrict the penalty values, based on previous experiences, to the range presented in Eq.(3.46) [38, 49].

From the discussion of the three parameters, it is possible to infer that by choosing the ALF, the general-purposed tool will remain efficient and robust when performing forward dynamic simulations. In fact, the choice of the ALF is a trade-off between its robustness and the efficiency of the Baumgarte method, as well as the advantage of the direct correction in not needing to choose the penalty values. This trade-off is advantageous since the ALF implementation will allow the general-purpose tool to run simulations close to singular positions.

Chapter 4

Program Benchmarking and Results

In this chapter the methodology for the benchmarking of the developed code is addressed, its formalism is briefly compared with the *MSC Adams* solver options, whose solutions are used as reference. The benchmarking models are exposed, and the obtained results discussed.

4.1 Methodology

After the choice of algorithms and implementation of the general-purposed tool code, the next step is to benchmark it to allow the identification of its limitations and to determine its computational efficiency. However, before starting the benchmarking of the program, it is fundamental to understand that the efficiency of custom MBS codes, such as the one developed for this work, can be influenced by several factors, as illustrated in Fig. 4.1.

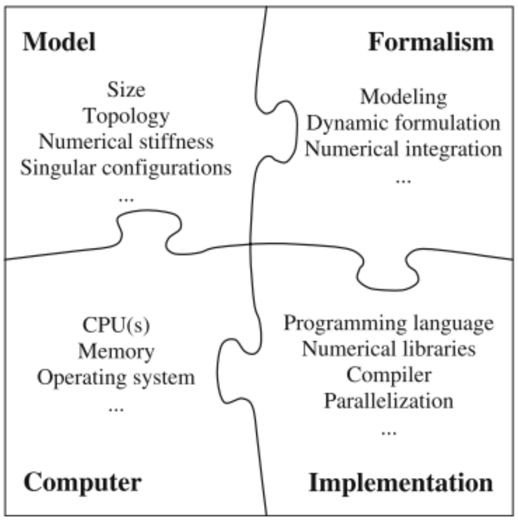


Figure 4.1: Puzzle of the factors that influence the benchmark of a MBS general code [50].

In Chapter 3, the formalism of the general-purposed tool was established, composed by how the general-purposed tool models its problems, the formulation it uses, and its numerical integration scheme. For the general-purposed tool developed, the modelling of the problem is made using *Excel*, previously analysed in detail in Sec. 3.2. In this *Excel*, all inputs are made in the global frame except for the inertial tensor, whose input is made in terms of the local reference frame which coincides with the principal axes of inertia, (I_{xx}, I_{yy}, I_{zz}). These inputs are then processed by the program to match the used MBD formulation.

The MBD formulation and numerical integration routine used by this program is composed of the ALF, without mass projection, and by the Adams-Bashforth-Moulton integration scheme which are capable of running simulations through singular configurations and redundant constraints, as stated in Sec. 3.2.2. These formalisms are optimum for the use of the general-purposed tool in the offline design process of the FST Lisboa prototypes. However, as stated by González et al., there is no optimal formalism for all kinds of problems, and as a result, this formalism is unable to run, for example, real-time hardware in loop simulations since this would require non-iterative and non-explicit methods, which is not this work formalism [50].

In terms of implementation, the main influence on the efficiency of the code developed in this work is the use of *MATLAB*. The *MSC Adams* simulation solver runs in *Fortran* and *C++*, both faster to execute and requiring less memory usage than *MATLAB* since the latter is an interpreted language. Furthermore, the routines of the general-purposed code may not be fully optimized and only take partial advantage of the *MATLAB* parallel computing toolbox, thus affecting the performance of the program for large-sized models versus a well-refined program such as *MSC Adams*.

The objective of the developed tool is to be applied in the development of future FST Lisboa prototypes. Therefore, it is necessary to evaluate the third factor that can influence the benchmarking, the hardware, taking into consideration the resources available to the team. Currently, the team has access to five desktops, with *Xeon* and *i7* processors, whose performance was evaluated through the *MATLAB bench* function. This function measures the execution time of six benchmarking tasks, which include a Lower-Upper Factorisation (LU), a Fast-Fourier Transformation (FFT), the execution of an ODE solver, the solving of a symmetric sparse matrix, and the execution of 2D and 3D plots. Of these tasks, only the fast Fourier transformation is not used by the general-purposed tool, thus the results obtained in Tab. 4.1 can give a perception of how this hardware can influence the efficiency of the program.

Machine	LU [s]	FFT [s]	ODE [s]	Sparse [s]	2D Plot [s]	3D Plot [s]
i7-4700MQ/12GB RAM Laptop	1.3680	0.9287	0.8255	0.6839	0.5142	0.6083
i7 4 Core/64 GB Desktop	0.6567	0.4006	0.4386	0.3434	1.2767	1.8160
Xeon 12 Core/64 GB Desktop	0.9612	0.5491	1.0902	0.8405	0.6512	0.6525

Table 4.1: Results of FST Lisboa desktops, for the execution times of the *MATLAB* bench function tasks.

The final factor that can influence the benchmarking is the model itself, namely, its size, if there are holonomic and non-holonomic constraints, and if redundant constraints or singular configurations are present. Additionally, if more complex situations such as flexible bodies or discontinuous effects are present, particularly impacts, clearances, and friction the program may require different formalisms to solve the model [50].

It is impossible to compare the performance and results of a code if the model used for the comparison and the integrator error tolerance required for the simulation is not the same for the code and the reference solution, in this case for the *MSC Adams*. To standardise this, González et al. developed a benchmarking system with a set of five basic 2D and 3D problems that have the advantage of being easy to model and are designed to test the ability of MBS codes to solve problems with specific extreme MBS characteristics. These problems and their specific characteristics are presented in Tab. 4.2.

ID	Problem	Evaluated characteristic
A01	Simple Pendulum	Example problem 2D
A02	N-four-bar mechanism	Singular Positions 2D
A03	Andrew's Mechanism	Very small time scale 2D
A04	Bricard's Mechanism	Redundant constraints 3D
A05	Flyball Governor	Stiff System 3D

Table 4.2: List of problems used for the benchmarking of the general-purposed tool [51].

For each of these test problems a corresponding model was built, with its specifications available online in a correspondent *PDF* file and *MSC Adams* model. Recently, another repository was developed by *IFtoMM* [52] that possesses more benchmarking problems with a complexity level similar to the original five of González et al. [51]. The *MSC Adams* files of González et al. were created in a 2008 version of *MSC Adams* and consequently are outdated and invalid to run in *MSC Adams* 2021 student version. Furthermore, both the *IFtoMM* repository and González et al. repository *PDF's* files do not possess the specific details needed to model some of these problems in the general-purposed tool.

Consequently, to uniformise the benchmarking process, it was established that the *PDF* description files would be used to remodel these problems into equivalent models in *MSC Adams*. This allows the confirmation of its solutions and to obtain all the data needed for the modelling of the problem in the general-purposed tool.

In essence, *MSC Adams* replaces the role that in future applications should be developed by *SolidWorks* or another CAD program.

4.2 Test Problems

Since the 3D kinematic program can be seen as an intermediate step to the 3D forward dynamic program, the main objective of the benchmarking phase is focused on simulating the five dynamic problems proposed by González et al., which isolate specific characteristics of the MBD formalism. Thus, testing the operational limits of the code allowing to infer the codes limitations, stability, and provides guidance for its future development and subsequent use in the simulation of vehicle prototypes [50, 51].

4.2.1 Simple Pendulum

The simple pendulum problem is used as a demonstrative example. Despite the fact that it does not evaluate a specific characteristic of the code developed, it constitutes an adequate starting point for the benchmarking of the program by being a very simplistic problem with an easily determined analytical solution. Allowing to draw conclusions on the proper working state of the core formalism of the code.

The mechanism is composed of a spherical mass of 11.21 kg, a rod weighting 1 kg with a length of 1 m and two joints, as presented in Tab. 4.3 and Tab. 4.4. The initial position of the mechanism is set at rest in the horizontal position under the effect of gravity with an acceleration of 9.86 m/s^2. The *MSC Adams* model is represented in Fig. 4.2, and the simulation is left to run in both programs for ten seconds, $t_{sim} = 10$ s, with a time step of five milliseconds, $\Delta t = 0.005$ s, in *MSC Adams*.

ID	Body	CoM $[x, y, z]$ [mm]	Mass [kg]	Inertia Tensor $[I_{xx}, I_{yy}, I_{zz}]$ [kgmm^2]
1	Absolute Frame	$[0, 0, 0]$	0	$[0, 0, 0]$
2	Massless Link	$[-500, 0, 0]$	1	$[1, 1, 1]$
3	Pendulum Mass	$[-1000, 0, 0]$	11.21	$[21967, 21967, 21967]$

Table 4.3: Global bodies modelling information relative to the simple pendulum mechanism.

Type	Body 1 ID	Body 2 ID	Global Position $[x, y, z]$ [mm]
Revolute	1	2	$[0, 0, 0]$
Spherical	2	3	$[-1000, 0, 0]$

Table 4.4: Global joints modelling information relative to the simple pendulum mechanism.

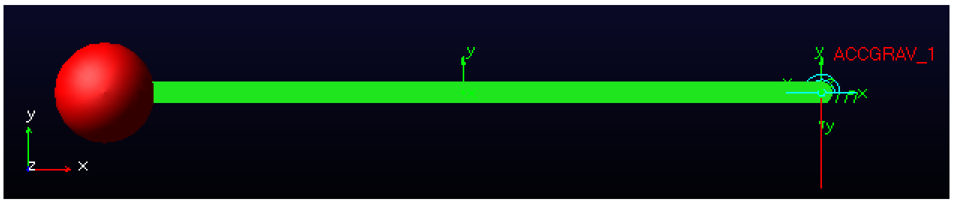


Figure 4.2: *MSC Adams* simple pendulum problem model.

For the general-purposed tool, because the *ode113*, Adams-Bashforth-Moulton, is a VSVO algorithm, it is only possible to set a maximum integration step, Δt_{max}, from which the integrator will vary, assuming values equal to the maximum step or lower to solve the forward dynamic problem. Nonetheless, it is possible to set specific times to which the *ode113*, subsequently, produces solutions of the same order of accuracy as the ones produced during the resolution of the integration process [53]. Therefore, it is possible to set this latter value to match the time step given to the *MSC Adams*, enabling the comparison between both solutions.

For this problem a maximum allowed integrator step of one millisecond is set, $\Delta t_{max} = 0,001$ s, and it is asked of the program to calculate a point at each five millisecond, $\Delta t_{calc} = 0.005$ s, to match the same calculated points as the *MSC Adams*.

For the *MSC Adams*, González et al. conducted a study of the performance of the different combinations between the available integration solvers and dynamic formalisms to solve the benchmarking problems with low and high integrator tolerances. Since *MATLAB* is expected to be less optimised than *MSC Adams*, which results in more numerical instability, only the latter values of high integrator tolerance, $Tol \geq 1 \times 10^{-5}$, will be considered in order to guarantee the validity of the computational efficiency comparison [51]. Of these combinations, for the *MSC Adams* 2021 student version, only four are available the Wielenga Stiff Integrator (WSTIFF) and the Gear Stiff Integrator (GSTIFF), which can be paired with an Index-3 Formulation (I3), or a Stabilized Index-2 Formulation (SI2). The results for these available combinations are present in Tab 4.5.

Method	A01	A02	A03	A04	A05
GSTIFF + I3	-	-	-	**24.1**	21.6
GSTIFF + SI2	-	-	-	66.9	25.1
WSTIFF + I3	49.3	**32.3**	-	28.2	23.9
WSTIFF + SI2	**42.5**	-	**102.4**	29.3	**16.8**

Table 4.5: CPU time in seconds for the benchmarking problems of the different available *MSC Adams* student version solvers [51].

With the results from Tab. 4.5 the chosen *MSC Adams* solver was the WSTIFF + SI2 configuration. Whilst for the general-purposed program the formalism used has already been introduced in Chapter 3, and the penalty parameters for this simulation were chosen based upon the convergence map of Tab. 4.6 and Tab. 4.7, and assume the values of $\alpha = 1 \times 10^8$, $\Omega = 300$ and $\mu = 1$. For both programs, the integrator tolerance was set to 1×10^{-6}.

α	Condition Number	Difference in y displacement [%]				
		$\Omega = 500$	$\Omega = 300$	$\Omega = 150$	$\Omega = 100$	$\Omega = 50$
10^6	8.724×10^{11}	DNC	DNC	DNC	DNC	4.571
10^7	8.724×10^{12}	0.5559	0.5878	0.9713	1.722	4.571
10^8	8.724×10^{13}	0.5606	0.5878	0.9713	1.722	4.571

Table 4.6: Difference (%) between the obtained solution for the simple pendulum and the *MSC Adams* solution by varying the system numerical stiffness, Ω, and its convergence rate, α. DNC - Did Not Converge.

α	Condition Number	Computational Time [s]				
		$\Omega = 500$	$\Omega = 300$	$\Omega = 150$	$\Omega = 100$	$\Omega = 50$
10^6	8.724×10^{11}	DNC	DNC	DNC	DNC	1252.8
10^7	8.724×10^{12}	229.5	181.9	157.6	135.8	90.6
10^8	8.724×10^{13}	61.6	42.7	29.5	20.2	16.4

Table 4.7: *MATLAB* computational time obtained for the simple pendulum by varying the system numerical stiffness, Ω, and its convergence rate, α. DNC - Did Not Converge.

The monitored variables are the $x(t)$ and $y(t)$ displacements at the CoM of the pendulum sphere, these variables are in line with the one chosen by González et al. [51]. The resulting curves are represented in the graphics of Fig. 4.3 and Fig. 4.4.

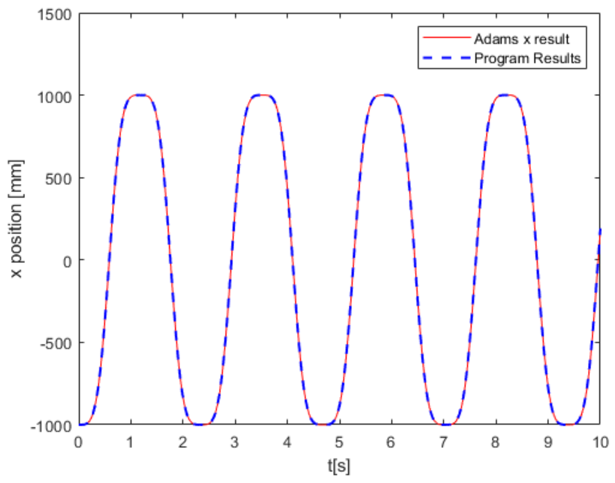


Figure 4.3: Evolution of the x position with time of the simple pendulum spheric mass.

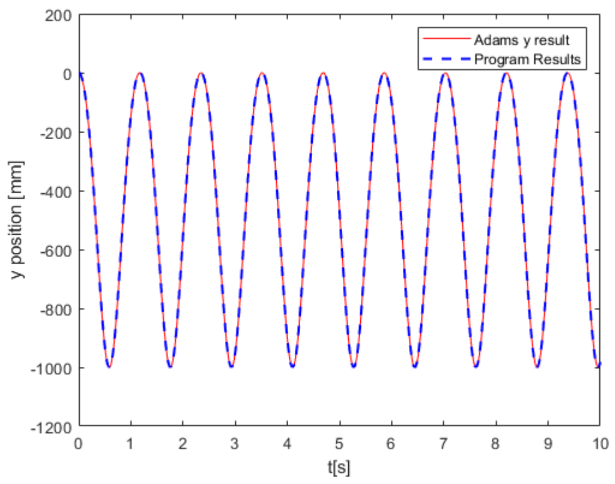


Figure 4.4: Evolution of the y position with time of the simple pendulum spheric mass.

From these curves, it is possible to infer that the simple pendulum problem can be effectively modelled into the general-purposed tool, producing results similar to the *MSC Adams*. Additionally, the differences (%) between the reference and the general-purposed tool obtained solution was also monitored during the length of the simulation, for both the x, y positions and their respective velocities and are presented in the graphics of Fig. 4.5, Fig. 4.6, Fig. 4.7 and Fig. 4.8.

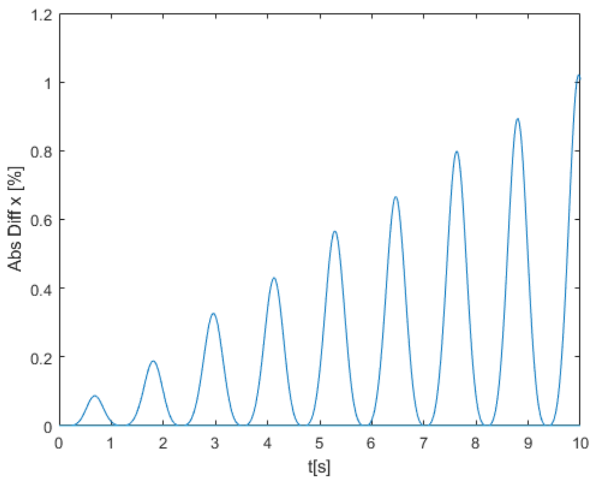


Figure 4.5: Evolution of the x difference (%) between the obtained solution and *MSC Adams* solution with time of the simple pendulum spheric mass.

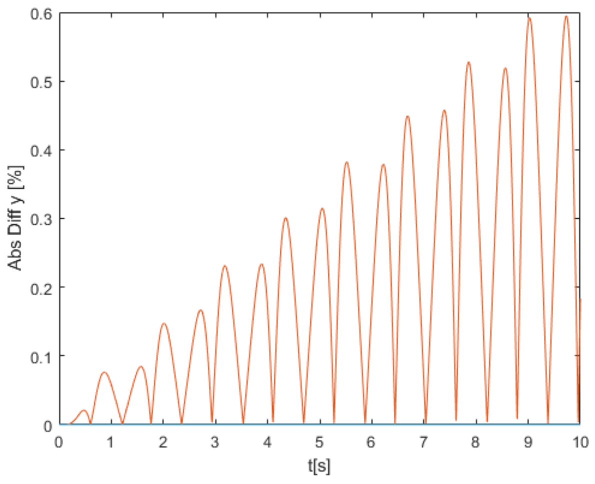


Figure 4.6: Evolution of the y difference (%) between the obtained solution and *MSC Adams* solution with time of the simple pendulum spheric mass.

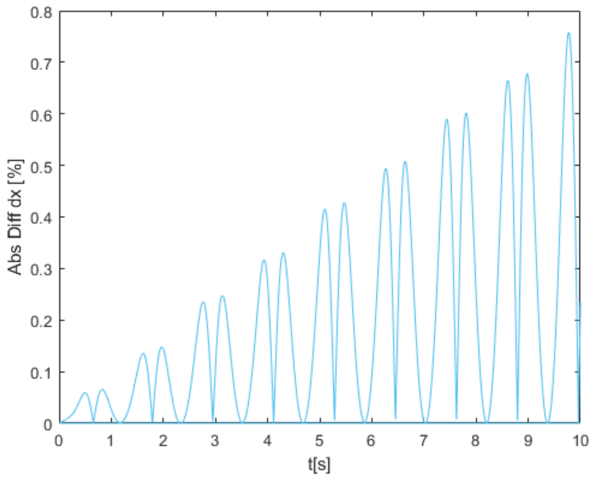


Figure 4.7: Evolution of the $\dot{x}$ difference (%) with time of the simple pendulum spheric mass.

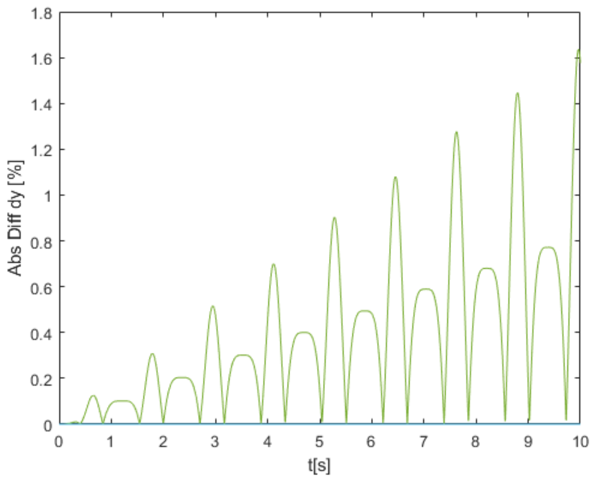


Figure 4.8: Evolution of the $\dot{y}$ difference (%) with time of the simple pendulum spheric mass.

It is noticeable, from these graphics that the results are within the 5 % acceptable margin established in Sec. 3.1.3. It is also possible to infer that these differences tend to increase linearly as the simulation develops, this occurs because the program solutions start to exhibit a small delay with the simulation elapsed time, as can be inferred from Fig. 4.9 and Fig. 4.10.

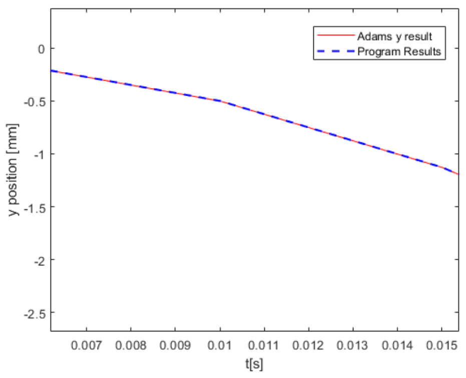


Figure 4.9: Phase shift in y at $t_{initial}$.

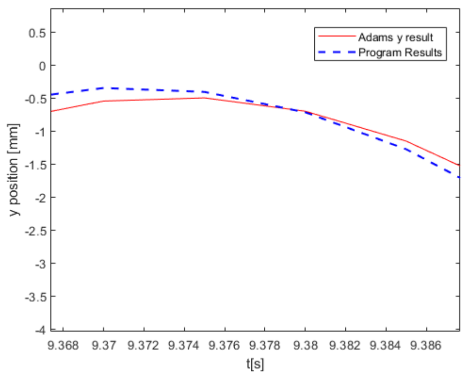


Figure 4.10: Phase shift in y at t_{final}.

The origin of this delay is not due to any error in the kinetic multibody formalism, but due to inaccuracies in the numerical integration process. Blajer, addressed the validity of the quantification of these inaccuracies through the variation of the total mechanical energy of the system, as being valid if the system in study is conservative, as is the case of the simple pendulum, where the input of energy does not occur, $\dot{E} = 0$ J/s [54].

Therefore, with the knowledge that in the initial position, at $t_{sim} = 0$ s, the system has all its energy stored as potential energy, and by applying Eq.(4.1), where E is the total energy, when $t_{sim} = t_{final}$ it is possible to infer that there is a residual energy dissipation of 8.7×10^{-3} J.

$$\Psi_E = E - E_0 = 0 \tag{4.1}$$

This numerical energy dissipation causes a phase shift in the numerical solution of the problem compared to its numerical exact solution, which translates into an advanced or backward calculated position of the system. It also results in the system velocity being increased or decreased, as can be observed for the close-up of the y position in Fig. 4.9 and Fig. 4.10, for the first and last peak in the sinusoidal response of Fig. 4.4 [54]. This numerical energy dissipation is more accentuated if mass-orthogonal projections are used to stabilise the integration scheme [51].

Furthermore, it is noticeable from Tab. 4.6, that the penalty values of Ω used during the simulation of this problem, are higher than the normal ranges suggested by De Jalon and Bayo and da Silva, M. [19, 38]. This occurs due to the instability created in the ALF by the *MATLAB* round-off errors at the end of each algebraic operation. Consequently, it is necessary to increase the value of Ω to enhance the stability properties of the ALF routine. This comes at a cost of computational effort since the Ω values induce numerical stiffness to the problem [49]. For future work, if a clean-up of residual of *MATLAB* after each calculation is implemented, it is possible to mitigate the use of these large values, unlocking faster computational times with lower differences (%), as depicted in Tab. 4.6.

4.2.2 N-four-bar mechanism

Problem A02 was first proposed by Bayo and Avello, to test the code's ability to perform simulations in critical scenarios where the mechanism runs near singular configurations [49]. The system in study is represented in the diagram of Fig. 4.11 and the *MSC Adams* model of Fig. 4.12, it is composed of five bars with a length of 1 m and a mass of 1 kg, as well as six revolute joints, as depicted by Tab. 4.8 and Tab. 4.9, respectively. Initially point B_0 has an initial velocity of 1.5 m/s, $V_x = 1.5$ m/s, and the mechanism is left to rotate around the z-axis under the effect of gravity for a simulation time of ten seconds, $t_{sim} = 10$ s.

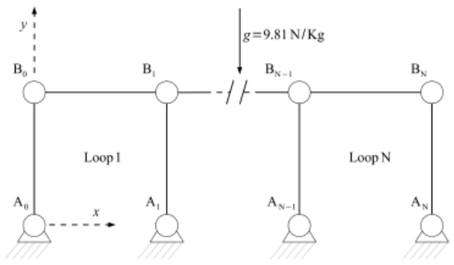


Figure 4.11: N-bar-four mechanism diagram [27].

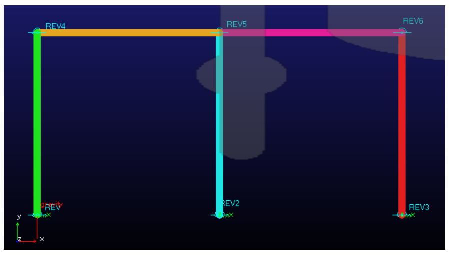


Figure 4.12: N-four-bar mechanism *MSC Adams* model.

ID	Body	CoM $[x, y, z]$ [m]	Mass [kg]	Inertia Tensor $[I_{xx}, I_{yy}, I_{zz}]$ [kgm^2]
1	Absolute Frame	$[0, 0, 0]$	0	$[0, 0, 0]$
2	Link 1	$[0, 0.5, 0]$	1.03	$[9.14 * 10{-}2, 1.7 * 10^{-4}, 9.15 * 10^{-2}]$
3	Link 2	$[0.5, 1, 0]$	1.03	$[1.71 * 10^{-4}, 9.14 * 10^{-2}, 9.15 * 10^{-2}]$
4	Link 3	$[1, 0.5, 0]$	1.03	$[9.14 * 10^{-2}, 1.71 * 10^{-4}, 9.15 * 10^{-2}]$
5	Link 4	$[1.5, 1, 0]$	1.03	$[1.71 * 10^{-4}, 9.14 * 10^{-2}, 9.15 * 10^{-2}]$
6	Link 5	$[2, 0.5, 0]$	1.03	$[9.14 * 10^{-2}, 1.71 * 10^{-4}, 9.15 * 10^{-2}]$

Table 4.8: Global bodies modelling information relative to the N-four-bar mechanism mechanism.

Type	Body 1 ID	Body 2 ID	Global Position $[x, y, z]$ [m]
Revolute	1	2	$[0, 0, 0]$
Revolute	2	3	$[0, 1, 0]$
Revolute	3	4	$[1, 1, 0]$
Revolute	4	1	$[1, 0, 0]$
Revolute	4	5	$[1, 1, 0]$
Revolute	5	6	$[2, 1, 0]$
Revolute	6	1	$[2, 0, 0]$

Table 4.9: Global joints modelling information relative to the N-four-bar mechanism mechanism

The simulation is run in *MSC Adams* with a solver combination WSTIFF + I3, chosen once again with recourse to the results obtained by González et al. represented in Tab. 4.5. For the general-purposed program, the formalism remains the same as in the previous problem, with the penalty parameters assuming values of $\alpha = 1 \times 10^{10}$, $\Omega = 1000$ and $\mu = 1$ after the convergence studies, displayed in Tab. 4.10 and Tab. 4.11. For both programs, the integrator tolerance was set to 1×10^{-6}.

α	Condition Number	Difference in y displacement [%]			
		$\Omega = 4000$	$\Omega = 2000$	$\Omega = 1000$	$\Omega = 500$
10^9	9.55×10^{10}	DNC	DNC	0.1635	0.4083
10^{10}	9.55×10^{11}	0.1143	0.1219	0.1680	0.4010
10^{11}	9.55×10^{12}	DNC	0.0919	0.1171	0.3351

Table 4.10: Difference (%) between the obtained solution for the N-four-bar mechanism and the *MSC Adams* solution by varying the system numerical stiffness, Ω, and its convergence rate, α. DNC - Did Not Converge.

α	Condition Number	Computational Time [s]			
		$\Omega = 4000$	$\Omega = 2000$	$\Omega = 1000$	$\Omega = 500$
10^9	9.55×10^{10}	DNC	DNC	139.9	103.3
10^{10}	9.55×10^{11}	479.1	245.8	150.3	113.9
10^{11}	9.55×10^{12}	DNC	592.8	446.8	409.3

Table 4.11: *MATLAB* computational time obtained for the N-four-bar mechanism by varying the system numerical stiffness, Ω, and its convergence rate, α. DNC - Did Not Converge.

In tandem with the last problem, the monitored variables are chosen to be in line with the ones monitored by González et al., which in this problem correspond to the x and y displacement of point B_0, represented in Fig. 4.13 and Fig. 4.14, respectively. The graphics corresponding to the differences between the two solutions are also presented for the x and y positions, Fig. 4.15 and Fig 4.16, alongside the graphics for the velocity differences in x and y, Fig. 4.17 and Fig. 4.18.

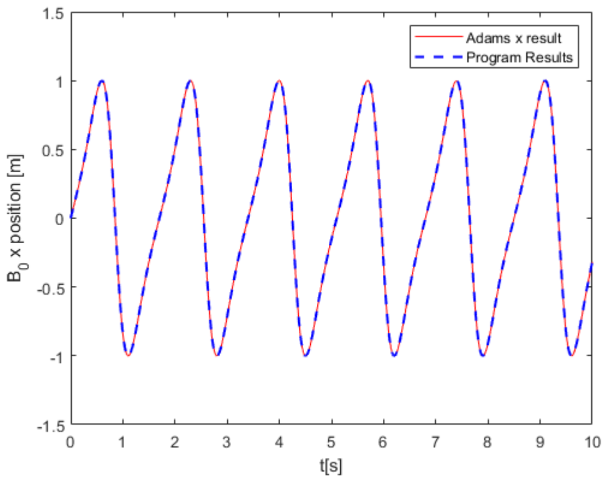


Figure 4.13: N-four-bar mechanism x position variation for point B_0.

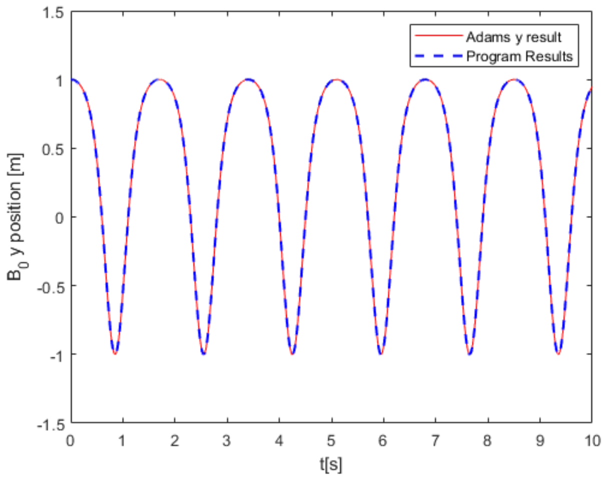


Figure 4.14: N-four-bar mechanism y position variation for point B_0.

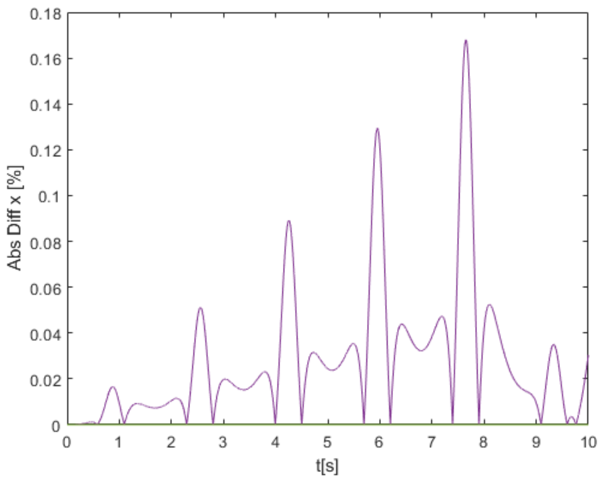


Figure 4.15: N-four-bar mechanism x differences (%) between the obtained solution and *MSC Adams* solution for point B_0.

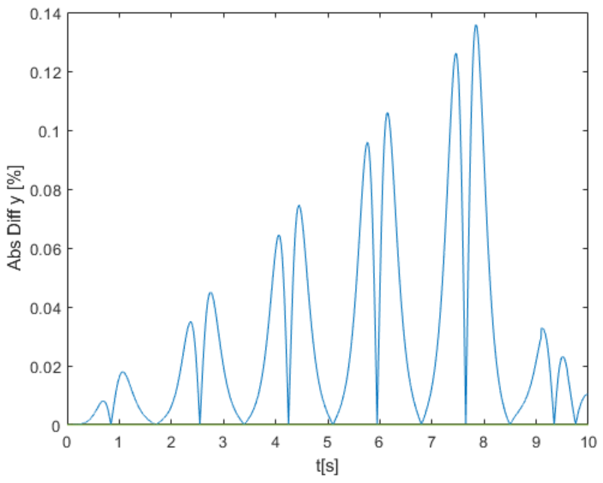


Figure 4.16: N-four-bar mechanism y differences (%) between the obtained solution and *MSC Adams* solution for point B_0.

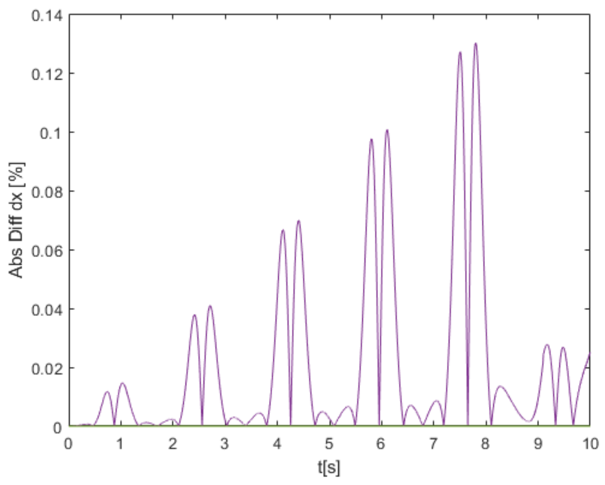


Figure 4.17: N-four-bar mechanism v_x differences (%) between the obtained solution and *MSC Adams* solution for point B_0.

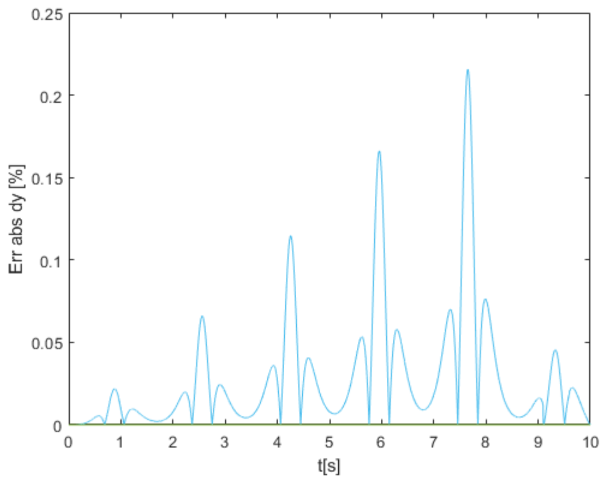


Figure 4.18: N-four-bar mechanism v_y differences (%) between the obtained solution and *MSC Adams* solution for point B_0.

From the graphics in Fig. 4.15 through Fig. 4.18 it is possible to infer that the differences between both solutions are within the acceptable margin of 5 %. Therefore, it is possible to state that the code is capable of solving simulations where the mechanism undergoes motion near singular configurations, which was expected since one of the advantages of the ALF is that the leading matrix, $[M + \Phi_q^T \alpha \Phi_q]$, is always positive definite, presenting a valid inverse matrix for the mechanism singular positions.

According to Bayo and Avello, simulations that undergo motion near singular positions are prone to a special high accumulation of errors produced by the numerical integration routine in combination with the round-off errors produced by the ALF near these positions [49]. This can be corroborated, even if the differences values are almost insignificant, by comparing the times in which the peaks of the difference in the y positions occur, Fig. 4.16, with the times in which point B_0 passes through the singular positions, $y_{B_0} = 0$, Fig. 4.14. This accumulation of errors was expected to result in the need to increase the numerical stiffness of the problem to enhance the stability of the ALF routine [49, 55]. To achieve this, the Ω values were increased, alongside the values of α to mitigate the increased computational effort required by the higher values of numerical stiffness.

Tab. 4.10 allows the conclusion, as expected, that the values of the difference between both solutions decrease in opposition to the Ω values. In contrast to the statement of da Silva, M., it is also shown that the penalizing coefficient, α, has some influence in the differences between both responses, which is not present in the case of the simple pendulum as represented in Tab. 4.6. However, this decrease in the differences (%) is outweighted by the increase of the computational time in *MATLAB*, which more than doubles for the values of Ω that did converge for $\alpha = 10^{11}$, as depicted in Tab. 4.11 [38]. This can be explained by the worsening of the conditioning of the problem with the increase of the α values, as visible in Tab. 4.11 by the increase in the conditioning number of the leading matrix. This increase can then lead to the development of round-off errors that may require the ALF to conduct more iterations to converge for a solution, resulting in an increased computational time [19].

Finally, it is possible to also infer that the program is capable of receiving translational velocities as an initial condition input and reproducing the motion caused by it, which is an important aspect for the performance of successful dynamic simulations in certain scenarios.

4.2.3 Andrew's Mechanism

Problem A03, also known as Andrew's mechanism, was first proposed by Schiehlen with the objective of studying the ability of different programs to study problems with very small-time scales [5]. The mechanism is represented in the diagram of Fig. 4.19 and by the *MSC Adams* model of Fig. 4.20, it is composed of seven small bodies, connected by revolute joints, whose motion is induced by an electric motor that actuates in point O, with a torque of 0.033 N/m. Since the mechanism itself is of small scale, the movement occurs in intervals of the order of one millisecond, 0.001 s, requiring an integrator capable of having integration step sizes lower than one millisecond, $\Delta t \leq 0.001$ s.

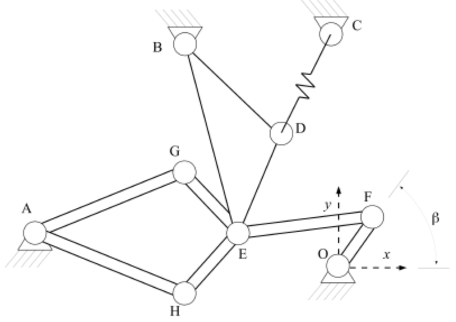


Figure 4.19: Andrew's mechanism *MSC Adams* model [27].

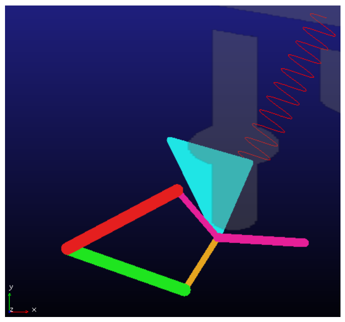


Figure 4.20: Andrew's mechanism diagram.

For this mechanism the data given in the literature by Schiehlen [5], González et al. [50] and Tagliapietra et al. [27], is given in terms of local coordinates in addition to an initial angle β and three global coordinates, points A, B and C. This format poses some difficulties in guaranteeing that the reproduced *MSC Adams* model, from which data relating to the frame orientation is taken to be used in the general-purposed tool model, is a reliable representation of the original problem.

As a consequence there was a need to find new sources in which the problem specifications were given in terms of global coordinates, giving a certain degree of confidence in the model reproduced in *MSC Adams*. For this, the work of Betsch et al. [56] was used and the model data is present in Tab. 4.12 and Tab. 4.13. Additionally, it is necessary to state that the CoM coordinates given from the source for the body 3 and 4 had to be corrected since their x value is symmetrical to what was presented. Finally, Tab. 4.14 presents the coordinates of the points at the ends of the spring, whose stiffness is equal to 4530 N/m and rest length is equal to 7.79×10^{-2} m.

ID	Body	CoM $[x, y, z]$ [m]	Mass [kg]	Inertia Tensor $[I_{xx}, I_{yy}, I_{zz}]$ [kgm^2]
1	Absolute Frame	$[0, 0, 0]$	0	$[0, 0, 0]$
2	OF	$[9.18 \times 10^{-4}, -5.7 \times 10^{-5}, 0]$	4.33×10^{-2}	$[0, 0, 2.19 \times 10^{-6}]$
3	FE	$[4.49 \times 10^{-3}, 2.8 \times 10^{-3}, 0]$	3.65×10^{-3}	$[0, 0, 4.41 \times 10^{-7}]$
4	BED	$[-1.8174 \times 10^{-2}, 2.048 \times 10^{-2}, 0]$	2.373×10^{-2}	$[0, 0, 5.26 \times 10^{-6}]$
5	EG	$[-3.02 \times 10^{-2}, 1.207 \times 10^{-2}, 0]$	7.06×10^{-3}	$[0, 0, 5.67 \times 10^{-7}]$
6	GA	$[-5.324 \times 10^{-2}, 1.663 \times 10^{-2}, 0]$	7.05×10^{-2}	$[0, 0, 1.17 \times 10^{-5}]$
7	AH	$[-5.926 \times 10^{-2}, -1.663 \times 10^{-2}, 0]$	5.498×10^{-2}	$[0, 0, 1.91 \times 10^{-5}]$
8	HE	$[-2.854 \times 10^{-2}, -1.072 \times 10^{-2}, 0]$	7.06×10^{-2}	$[0, 0, 5.667 \times 10^{-7}]$

Table 4.12: Global bodies modelling information, in global coordinates, relative to the Andrew's mechanism.

Type	Body 1 ID	Body 2 ID	Global Position $[x, y, z]$ [m]
Revolute	1	2	$[0, 0, 0]$
Revolute	2	3	$[6.99 \times 10^{-3}, -4.33 \times 10^{-4}, 0]$
Revolute	3	4	$[-2.096 \times 10^{-2}, 1.3 \times 10^{-3}, 0]$
Revolute	4	1	$[-3.635 \times 10^{-2}, 3.273 \times 10^{-2}, 0]$
Revolute	4	5	$[-2.096 \times 10^{-2}, 1.3 \times 10^{-3}, 0]$
Revolute	5	6	$[-3.4 \times 10^{-2}, 1.646 \times 10^{-2}, 0]$
Revolute	6	1	$[-6.934 \times 10^{-2}, -2.27 \times 10^{-3}, 0]$
Revolute	1	7	$[-6.934 \times 10^{-2}, -2.27 \times 10^{-3}, 0]$
Revolute	7	8	$[-3.163 \times 10^{-2}, -1.562 \times 10^{-2}, 0]$
Revolute	8	3	$[-2.096 \times 10^{-2}, 1.3 \times 10^{-3}, 0]$

Table 4.13: Global joints modelling information, in global coordinates, relative to the Andrew's mechanism.

Type	Body 1 ID	Body 2 ID	Point C	Point D
Spring	1	4	$[1.4 \times 10^{-2}, 7.2 \times 10^{-2}, 0]$	$[-1.05 \times 10^{-2}, 2.54 \times 10^{-2}, 0]$

Table 4.14: Global spring modelling information relative to the Andrew's mechanism.

Akin to the other problems, the simulation is left to run in *MSC Adams* for five milliseconds, $t_{sim} = 0.005$ s, with a time step $\Delta t = 3 \times 10^{-5}$ s, using the solver combination of WSTIFF + SI2 as determined in Tab. 4.5. In the general-purposed tool the simulation was left to run for the same duration, with a maximum integration step for the Adams-Bashforth-Moulton VSVO of $\Delta t_{max} = 3 \times 10^{-5}$ s, while requiring a point to be calculated also at each $\Delta t_{calc} = 3 \times 10^{-5}$ s. Finally, both programs used an integrator tolerance of 1×10^{-6}, and the convergence studies performed to choose the adequate penalty values (α, Ω and μ) are displayed in Tab. 4.15 and Tab. 4.16.

α	Condition Number	Difference in y displacement [%]		
		$\Omega = 10^6$	$\Omega = 10^5$	$\Omega = 10^4$
10^2	7.268×10^7	0.1230	0.073	10.04
10^3	7.268×10^8	0.1230	0.073	10.04
10^4	7.268×10^9	0.1226	0.073	10.04

Table 4.15: Difference (%) between the obtained solution for the Andrew's mechanism and the *MSC Adams* solution by varying the system numerical stiffness, Ω, and its convergence rate, α.

α	Condition Number	Computational Time [s]		
		$\Omega = 10^6$	$\Omega = 10^5$	$\Omega = 10^4$
10^2	7.268×10^7	1044.6	121.0	37.9
10^3	7.268×10^8	850.9	85.5	27.1
10^4	7.268×10^9	824.7	85.9	27.3

Table 4.16: *MATLAB* computational time obtained for the Andrew's Mechanism by varying the system numerical stiffness, Ω, and its convergence rate, α.

Following the convergence studies, the penalty parameters assume the values of $\alpha = 10^3$, $\Omega = 10^5$ and $\mu = 1$ for this simulation. Additionally, the graphics corresponding to the differences between the two solutions are presented for the monitored values of x and y displacements of point E, as depicted in Fig. 4.21 and Fig. 4.22. Alongside with its respective differences (%) in relation to the solution obtained from *MSC Adams*, presented in Fig. 4.23 and Fig. 4.24.

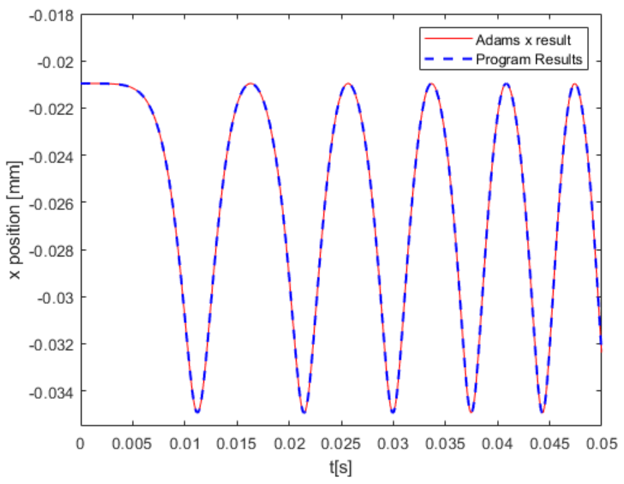


Figure 4.21: Andrew's Mechanism x position variation with time for point E.

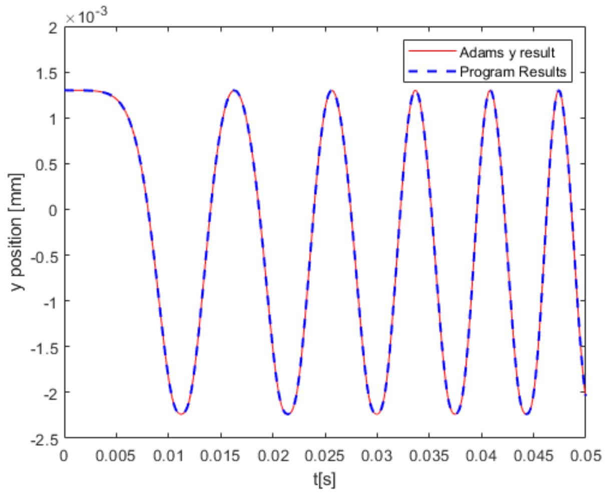


Figure 4.22: Andrew's Mechanism y position variation with time for point E.

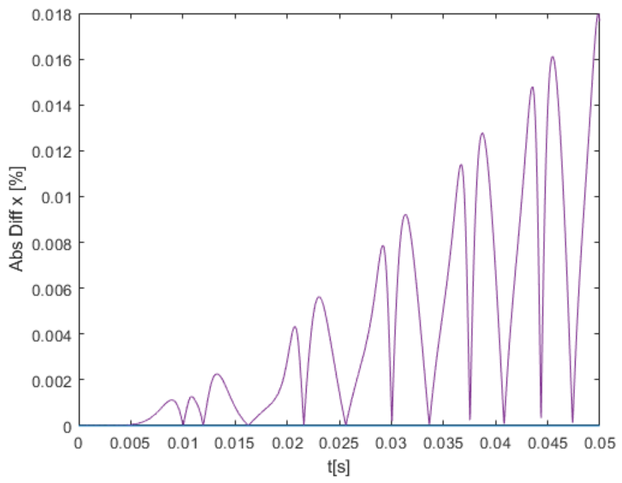


Figure 4.23: Andrew's mechanism x differences (%), between the obtained solution and the *MSC Adams* solution for point E.

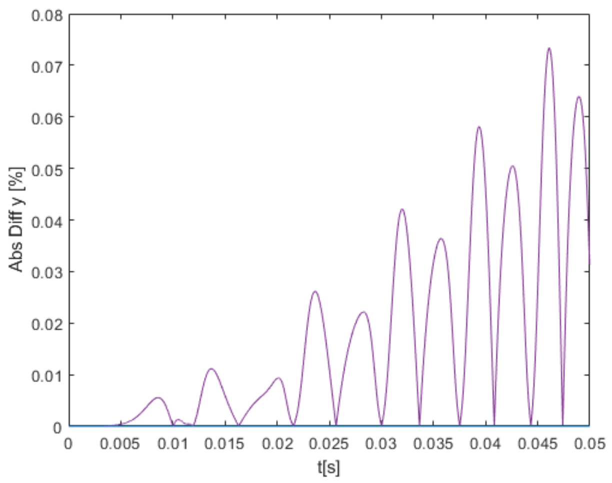


Figure 4.24: Andrew's mechanism y differences (%), between the obtained solution and the *MSC Adams* solution for point E.

For this mechanism, the general-purposed tool was able to provide an adequate result, being effective in registering values in intervals as small as one millisecond, 0.001 s. Furthermore, it is necessary to highlight that this model also tested other features of the tool, such as using torque inputs in the model's initial conditions to cause motion in the mechanism. As well as, test the subroutines used to model the spring force element, which in this case the connecting point with body 4 does not coincide with the CoM of the body and therefore the momentum created by its actuation also needs to be taken into account. All essential features to run successful forward dynamics simulations.

Finally, it is necessary to highlight that from the convergence studies it was possible to infer that if the penalty parameter, Ω, assumes lower values such as 10^4, the error sharply increases. This can be explained by the presence of the spring force element in this mechanism which causes the response to slow down and consequently get out of phase with the response obtain from *MSC Adams*, as can be inferred in the response of the mass-spring-damper system of A represented in Fig. A.4.

4.2.4 Bricard's Mechanism

Bricard's mechanism, represented in Fig. 4.25, is a 3D model whose objective is to test the code's ability to run problems that are overconstrained. This is extremely important for problems where a set of linearly dependent kinematic constraints cannot be identified during the modelling phase by the user, because it changes during its motion, as is the case of this example.

The mechanism is composed of five cylindrical rods, as depicted in Tab. 4.17, and by six revolute joints, Tab. 4.18, whose particular orientation results in the system having motion when subjected to external forces, despite having zero DOF through the mobility formula of Eq.(2.5). The system in this example is actuated by gravity, with the orientation depicted in Fig. 4.26, which will induce a motion that causes the redundant constraint to change, but the system does not reach any singular configurations.

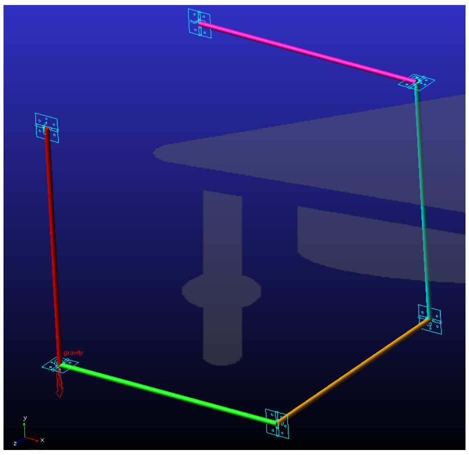


Figure 4.25: Bricard's mechanism *MSC Adams* model.

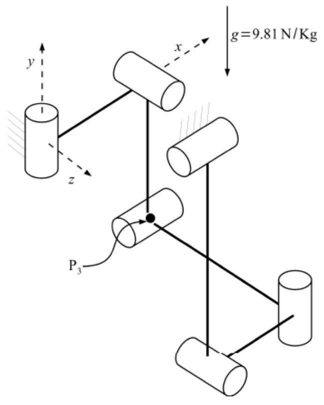


Figure 4.26: Bricard's mechanism diagram [27].

ID	Body	CoM $[x, y, z]$ [m]	Mass [kg]	Inertia Tensor $[I_{xx}, I_{yy}, I_{zz}]$ [kgm^2]
1	Absolute Frame	$[0, 0, 0]$	0	$[0, 0, 0]$
2	Rod 1	$[0, 0.5, 0]$	0.94	$[9.14 \times 10^{-2}, 1.71 \times 10^{-4}, 9.15 \times 10^{-2}]$
3	Rod 2	$[0.5, 0, 0]$	0.94	$[4.71 \times 10^{-5}, 7.86 \times 10^{-2}, 7.87 \times 10^{-2}]$
4	Rod 3	$[1, 0, -0.5]$	0.94	$[7.86 \times 10^{-2}, 7.86 \times 10^{-2}, 4.71 \times 10^{-5}]$
5	Rod 4	$[1, 0.5, -1]$	0.94	$[7.86 \times 10^{-2}, 4.71 \times 10^{-5}, 7.86 \times 10^{-2}]$
6	Rod 5	$[0.5, 1, -1]$	0.94	$[4.71 \times 10^{-5}, 7.86 \times 10^{-2}, 7.87 \times 10^{-2}]$

Table 4.17: Global bodies modelling information relative to the Bricard's mechanism.

Type	Body 1 ID	Body 2 ID	Global Position $[x, y, z]$ [mm]
Revolute	1	2	$[0, 1, 0]$
Revolute	2	3	$[0, 0, 0]$
Revolute	3	4	$[1, 0, 0]$
Revolute	4	5	$[1, 0, -1]$
Revolute	5	6	$[1, 1, -1]$
Revolute	6	1	$[0, 1, -1]$

Table 4.18: Global joints modelling information relative to the Bricard's mechanism.

The simulation is run in both programs for ten seconds, $t_{sim} = 10$ s and an integrator tolerance of 1×10^{-6}. For *MSC Adams*, the solver configuration used is the GSTIFF + I3, as indicated in Tab. 4.5, with a time step of one millisecond, $\Delta t = 0.001$ s. The general-purposed tool maintains the same formalism as before, Adams-Bashforth-Moulton with the ALF method for the stabilisation of the kinematic constraint equations, with a maximum allowed integration step of one millisecond, $\Delta t_{max} = 0.001$ s, while a point is asked to be calculated at every one millisecond, $\Delta t_{calc} = 0,001$ s. The penalty parameters assume values of $\alpha = 1 \times 10^6$, $\Omega = 500$ and $\mu = 1$ following the convergence studies exhibited in Tab. 4.19 and Tab. 4.20.

α	Condition Number	Difference in y displacement [%]			
		$\Omega = 4000$	$\Omega = 2500$	$\Omega = 1000$	$\Omega = 500$
10^6	7.7×10^7	0.0192	0.0173	0.041	0.0544
10^7	7.7×10^8	0.008	0.0102	0.1680	DNC
10^8	7.7×10^9	0.0595	0.183	DNC	DNC

Table 4.19: Difference (%) between the obtained solution for the Bricard's mechanism and the *MSC Adams* solution by varying the system numerical stiffness, Ω, and its convergence rate, α. DNC - Did Not Converge.

α	Condition Number	Computational Time [s]			
		$\Omega = 4000$	$\Omega = 2500$	$\Omega = 1000$	$\Omega = 500$
10^6	7.7×10^7	527.7	270.6	128.6	86.5
10^7	7.7×10^8	549.6	295.5	142.7	DNC
10^8	7.7×10^9	550.8	298.7	DNC	DNC

Table 4.20: *MATLAB* computational time obtained for the Bricard's mechanism by varying the system numerical stiffness, Ω, and its convergence rate, α. DNC - Did Not Converge.

The monitored variables during the simulation are the x, y and z coordinates of point P_3, represented in Fig. 4.26, whose curves are represented in Fig. 4.27, Fig. 4.28 and Fig. 4.29, respectively. Alongside with its respective differences (%) in relation to the solution obtained from *MSC Adams*, presented in Fig. 4.30, Fig. 4.31 and Fig. 4.32.

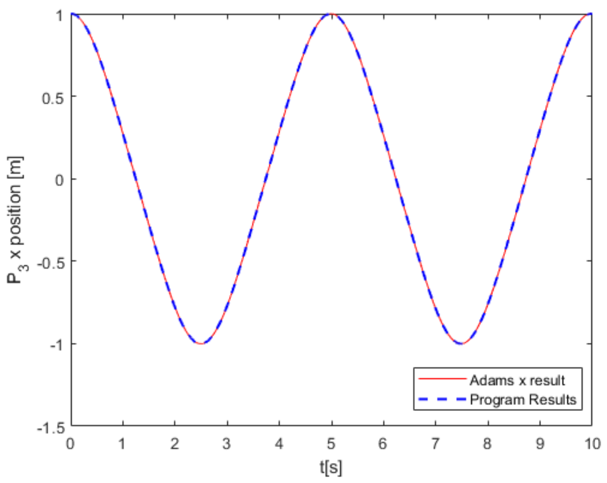


Figure 4.27: Bricard's Mechanism x position variation with time for point P_3.

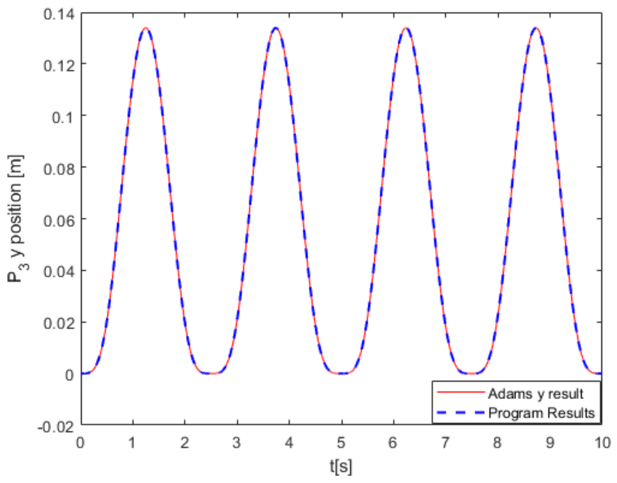


Figure 4.28: Bricard's Mechanism y position variation with time for point P_3.

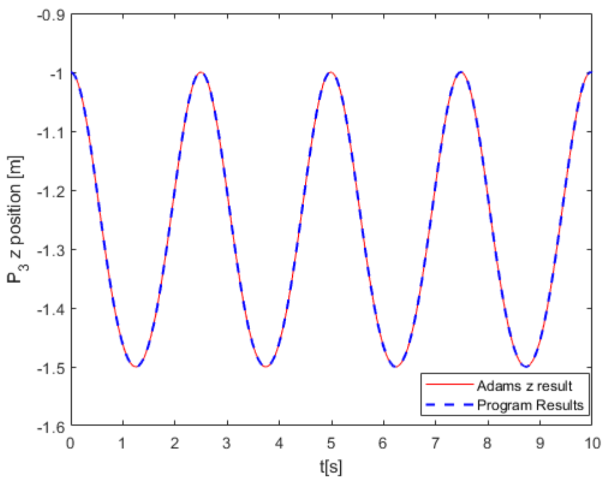


Figure 4.29: Bricard's Mechanism z position variation with time for point P_3.

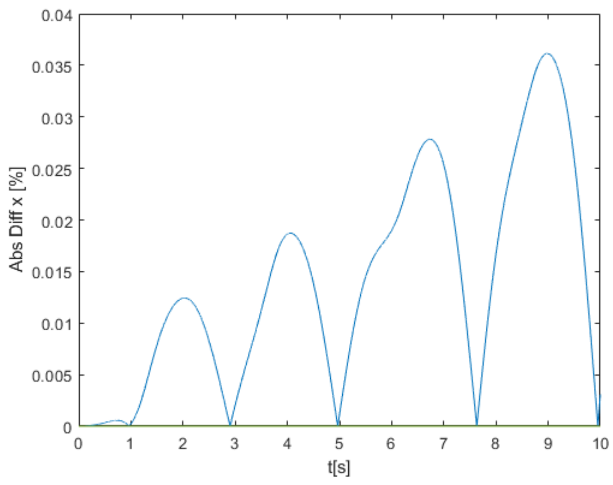


Figure 4.30: Bricard's mechanism x differences (%), between the obtained solution and the *MSC Adams* solution for point P_3.

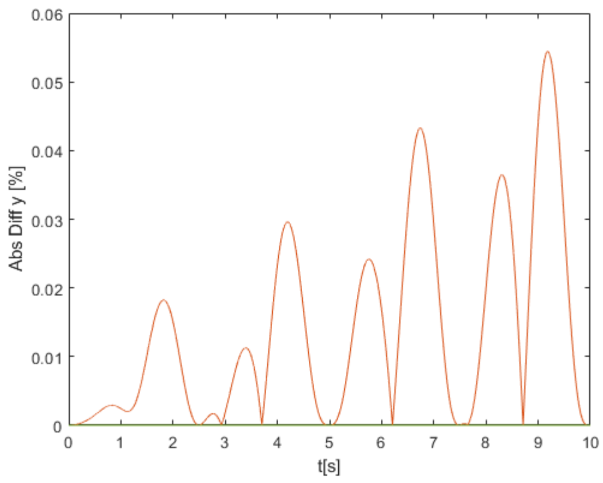


Figure 4.31: Bricard's mechanism y differences (%), between the obtained solution and the *MSC Adams* solution for point P_3.

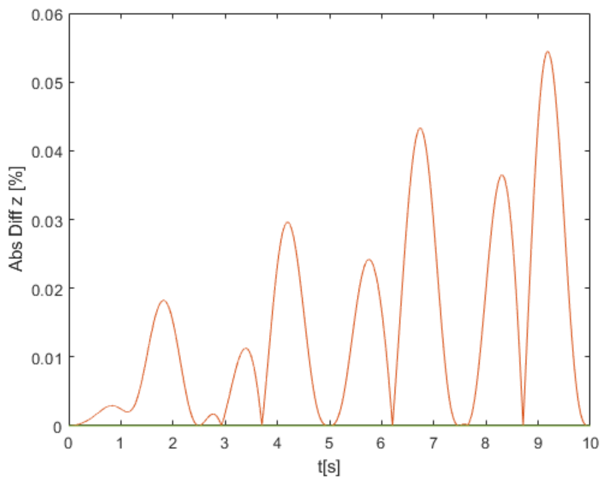


Figure 4.32: Bricard's mechanism z differences (%), between the obtained solution and the *MSC Adams* solution for point P_3.

From the graphics of Fig. 4.30, Fig. 4.31 and Fig. 4.32 it is possible to infer that the differences between the solutions are within the acceptable margin of 5 %. Therefore, it is possible to confirm that the ALF routine, as stated in Sec. 3.2.2, is more than capable of dealing with the rank deficient Jacobian matrices, Φ_q, created by the redundant constraints [55].

Furthermore, by analysing Tab 4.19 it is possible to infer, similarly to the N-Bar mechanism, that the α parameter does exhibit some unexpected influence in the differences (%) between the obtained solutions for the Bricard's mechanism. However, Tab. 4.20 allows the observation that this increase in α does not affect the computational time in the same order of magnitude of the N-Bar mechanism, in which the computational time almost doubles, as presented in Tab. 4.11. Similarly by observing Tab. 4.7 and 4.16 it is possible to state that the influence of α in the Bricard's mechanism computational time does not exhibit the same behaviour as in the simple pendulum and Andrew's mechanism.

The explanation for this can be found in the nature of the four problems discussed before, as well as, with recourse to the work developed by González et al. [55]. As stated before, Bricard's mechanism possesses a deficient Jacobian matrix due to redundant constraints but does not undergo motion in singular positions, this will generate some numerical difficulties which can explain the fact that the computational time remains almost constant for values of $\alpha = 10^7$ and $\alpha = 10^8$. However, González et al. states that the numerical difficulties are aggravated when the body undergoes motion in singular positions because impact forces are suddenly introduced into the constraint reactions produced by the penalty system of Eq.(3.46) [55]. These constraint reactions make the system solution convergence of the N-Bar mechanism more susceptible to the variation of the penalty parameters, explaining the doubling of the computational time when the condition number of the leading matrix increases, as a consequence of the α increase from 10^{10} to 10^{11}. In contrast, the simple pendulum and Andrew's mechanism are mechanisms that do not undergo any of these phenomena and as a result, the variation of the solutions differences (%) and computational time follow what was expected from the literature, as perceivable in Tab. 4.6, Tab. 4.15, Tab. 4.7 and Tab. 4.16 respectively [19, 55].

4.2.5 Flyball Governor Mechanism

The flyball governor mechanism, represented in Fig. 3.3, is the most complex system to be benchmarked in this work. As depicted in the figure, the system has an initial angular velocity of 2π rad/s induced in its base around the y axis and its movement is done under the effects of gravity, $g = 9.81$ m/s^2. Its main objective is to test the code's ability to solve stiff mechanical systems, which was obtained by replacing the coupler rods with spring-damper elements, whose stiffness coefficient, $k = 8 \times 10^5$ N/m, damping coefficient, $c = 4 \times 10^4$ Ns/m and rest length, $L_0 = 0.5$ m were adjusted for this purpose.

Using the specifications provided in *PDF* file by Tagliapietra et al., it was possible to produce the *MSC Adams* model of Fig. 4.33, whose modelling information is presented in Tab 4.21, Tab. 4.22, Tab. 4.23 and Tab. 4.24 [27].

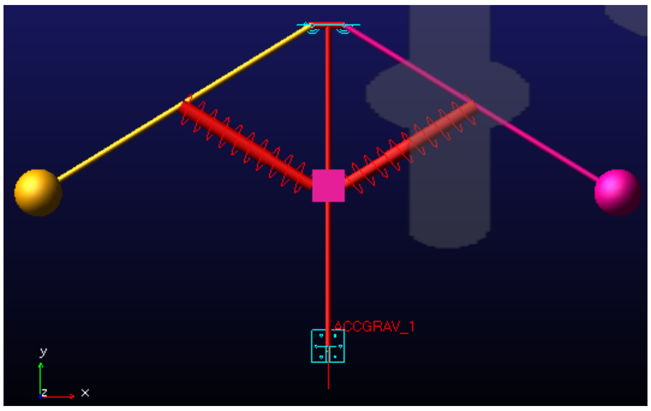


Figure 4.33: Flyball governor *MSC Adams* model.

ID	Body	CoM $[x, y, z]$ [m]	Mass [kg]	Inertia Tensor $[I_{xx}, I_{yy}, I_{zz}]$ [kgm^2]
1	Absolute Frame	$[0, 0, 0]$	0	$[0, 0, 0]$
2	Base	$[0, 0.5171, 0]$	0.9758	$[8.662 \times 10^{-2}, 7.808 \times 10^{-5}, 8.662 \times 10^{-2}]$
3	Rod 1	$[0.901, 0.5088, 0]$	5.9425	$[1.0867 \times 10^{-2}, 3.5 \times 10^{-1}, 3.5 \times 10^{-1}]$
4	Rod 2	$[-0.901, 0.5088, 0]$	5.9425	$[1.0867 \times 10^{-2}, 3.5 \times 10^{-1}, 3.5 \times 10^{-1}]$
5	Spring/Damper Supp	$[0, 0.5, 0]$	3	$[5 \times 10^{-3}, 5 \times 10^{-3}, 5 \times 10^{-3}]$

Table 4.21: Global bodies modelling information relative to the Flyball governor mechanism.

Type	Body 1 ID	Body 2 ID	Global Position $[x, y, z]$ [m]
Revolute	1	2	$[0, 0, 0]$
Revolute	2	3	$[0.05, 1, 0]$
Revolute	2	4	$[-0.05, 1, 0]$
Prismatic	2	5	$[0, 0.5, 0]$

Table 4.22: Global joints modelling information relative to the Flyball governor mechanism.

Type	Body 1 ID	Body 2 ID	Point Body 1	Point Body 2
Spring	5	3	$[5 \times 10^{-2}, 0.5, 0]$	$[0.483, 0.750, 0]$
Spring	5	4	$[-5 \times 10^{2}, 0.5, 0]$	$[-0.483, 0.750, 0]$

Table 4.23: Global spring modelling information relative to the Flyball governor mechanism.

Type	Body 1 ID	Body 2 ID	Point Body 1	Point Body 2
Spring	5	3	$[5 \times 10^{-2}, 0.5, 0]$	$[0.483, 0.750, 0]$
Spring	5	4	$[-5 \times 10^{2}, 0.5, 0]$	$[-0.483, 0.750, 0]$

Table 4.24: Global dampers modelling information relative to the Flyball governor mechanism.

Analogously to the other problems, the monitored variables are indicated by González et al., which in this case correspond to the y position of the spring/damper support represented by the variable s in Fig. 3.3. For both programs the simulation was run first for a total time of ten seconds, $t_{sim} = 10$ s, and then for a total of five seconds, $t_{sim} = 5$ s, to confirm if the dynamic response was not degrading over time. The integrator tolerances in both programs were set to 1×10^{-6}, with a time step of one millisecond, $\Delta t = 0.001$ s. For the general-purposed tool, a point was calculated at every millisecond, $\Delta t_{calc} = 0.001$ s, and the penalty parameters values were accordingly to the convergence studies of Tab. 4.25 and Tab. 4.26 assuming the values of $\alpha = 10^8$, $\Omega = 10^2$ and $\mu = 1$. Finally, the results obtained for the monitored variables are depicted for $t_{sim} = 10$ s in Fig. 4.34, alongside its respective differences (%) in relation to the solution obtained from *MSC Adams*, presented in Fig 4.35.

α	Condition Number	Difference in y displacement [%]			
		$\Omega = 10^2$	$\Omega = 10^3$	$\Omega = 10^4$	$\Omega = 10^5$
10^7	7.5771×10^8	1.8796	2.6278	2.6359	2.6360
10^8	7.5771×10^9	1.8796	2.6278	2.6359	2.6360
10^9	7.5771×10^{10}	1.8796	2.6278	2.6359	DNC

Table 4.25: Difference (%) between the obtained solution for the Flyball governor and the *MSC Adams* solution by varying the system numerical stiffness, Ω, and its convergence rate, α. DNC - Did Not Converge.

α	Condition Number	Computational Time [s]			
		$\Omega = 10^2$	$\Omega = 10^3$	$\Omega = 10^4$	$\Omega = 10^5$
10^7	7.5771×10^8	263.46	294.48	765.86	13876
10^8	7.5771×10^9	266.16	263.57	654.67	13390
10^9	7.5771×10^{10}	262.85	311.67	691.61	DNC

Table 4.26: *MATLAB* computational time obtained for the Flyball governor by varying the system numerical stiffness, Ω, and its convergence rate, α. DNC - Did Not Converge.

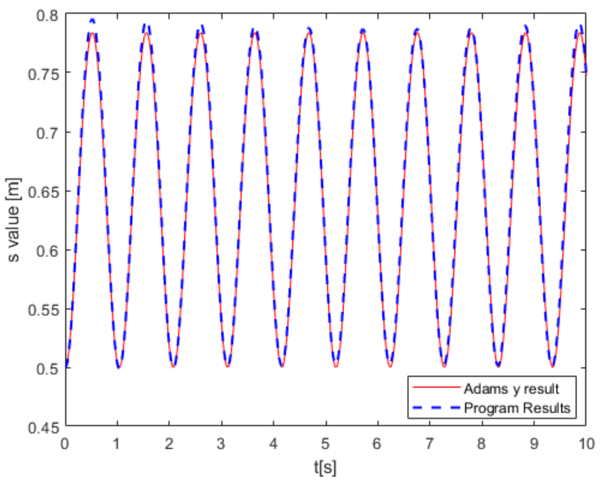


Figure 4.34: Flyball governor s values variation with time.

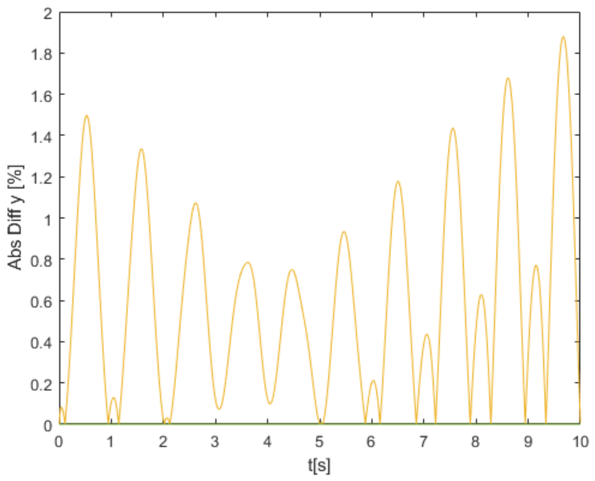


Figure 4.35: Flyball governor s value differences (%), between the obtained solution and the *MSC Adams* solution.

From the analysis of Tab. 4.25 it is possible to infer that the errors vary as expected with the increase or decrease of the penalty parameter α. However, its variation with the Ω parameter does not behave as expected, since it decreases from $\Omega = 10^3$ to $\Omega = 10^2$. This may be explained by the fact that increasing Ω will induce numerical stiffness in the problem and may, contrary to the other problems where stabilises the ALF routine, contribute to further increase the instability caused on the ode solver by the already stiff problem, however this increase in the differences is punctual and the maximum difference stabilises for the other orders of magnitude, $\Omega = 10^4$ and $\Omega = 10^5$ [49].

Since the system is classified as stiff, it is necessary to highlight that the chosen integrator Adams-Bashforth-Moulton, *ode113*, is classified as being conditionally stable as stated by da Silva, M. [38]. This implies that in some conditions, such as the ones created by a stiff system, the predictor-corrector routine can become unstable leading to a solution not being obtained or requiring the use of very small integration steps to obtain a valid solution leading to a high computational cost.

In this case, the solver proves itself capable of solving the stiff equations of the problem, however, this success may be punctual as stated by Fernández de Bustos et al. who further clarifies that this problem still can be solved by an explicit method, such as the predictor routine Adams-Bashforth [57]. Nonetheless, in the face of problems in which small integration steps are not enough to avoid instability issues, the general-purposed tool is still compatible with other stiff ODE solvers provided by *MATLAB* such as the *ode23s*, based upon a modified Rosenbrock formula, or the VSVO *ode15s*, based on the numerical differentiation formulas, this compatibility can be achieved by changing one line of code allowing the tool to have some flexibility when trying to solve this type of problems.

Additionally, by analysing Fig. 4.34 and Fig. 4.35 it is possible to conclude that the response given by the general-purposed tool is within the acceptable margin of 5 % and that it is capable of solving systems which involve several non-zero initial conditions inputs alongside with different types of joints and force elements, with the latter not being centred in the connecting bodies CoM. Finally, by analysing the maximum differences between the 5 s and 10 s simulations it is possible to infer that it increases with the simulation time, $\Delta max_{diff} = t_{10s} - t_{5s} = 1.8796 - 1.4983 = 0.3813$ %, which implies that for longer simulations of this type the solution may deteriorate to a point where it is unreliable.

4.2.6 CPU Time comparison

In Sec. 4.1, it was stated that one of the disadvantages of this program was the fact that it ran in *MATLAB* which is an interpreted programming language, and therefore would require more computational time than the one used by *MSC Adams*. In Tab. 4.7, Tab. 4.11, Tab. 4.16, Tab. 4.20 and Tab. 4.26 a series of computational times are registered for each simulation using *MATLAB*'s *tic* function, which uses the clock time, to indicate to the user the elapsed computational time. However, to compare *MSC Adams* with *MATLAB*, it is best to resort to the *cputime* function, which indicates the CPU Time that can also be retrieved in *MSC Adams* and solely considers the processing time taken to run the simulations and not any idle-time in which the software is waiting for hardware processes.

The obtained results are represented in Tab. 4.27 and corroborate the previous statement.

ID	*MATLAB* [s]	*MSC Adams* [s]
A01	72.7	10.2
A02	503.1	36.5
A03	281.6	18.4
A04	264.9	4.5
A05	910.8	21.3

Table 4.27: Comparison of the CPU time used by the general-purposed tool and the *MSC Adams* to solve each of the proposed problems. DNC - Did Not Converge

Chapter 5

Conclusions and Future Work

This work aimed to use the already existent Excel interface of the failed 3D kinematic simulator, *perfectkinematics*, to produce a general-purposed tool capable of solving position, velocity and acceleration analysis for kinematic and forward dynamic problems, in the context of vehicle dynamics. It was also intended that this tool should work based upon *MATLAB*, *SolidWorks* and *Excel*, all software's to which the team members are familiar.

To produce this tool first a 3D Kinematic program was formulated, using Euler parameters to avoid singularities. From this, it was possible to program all the necessary algebraic formalisms and allow for an easier debugging of the kinematic constraint equations. This verification was made with recourse to a slider-crank model, which passes through a kinetic singularity, and the *MSC Adams* software which allowed the comparison of solutions.

Subsequently, the 3D forward dynamics program was developed upon the constraint equations and algebraic formalisms of the kinematic program. Since one of the objectives was to produce a stable algorithm, in-depth research was done on *MATLAB*'s ode solver options and the mitigation of the constraints violations that occur during the integration process. From this, it was picked for the algorithm the Adams-Bashforth-Moulton, *ode113* integrator for non-stiff equations, and the index-1 Augmented Lagrangian Formula, whose combination was determined to be capable of solving singular configurations and overconstrained systems.

During the development of the program special attention was given to how the integration of its subroutines, namely, the actuation of forces and torques on the bodies CoM, and force elements could be made to allow future routines to be easily implemented, without overhauling the programs code, such as tire models. Accordingly, the code was developed to be as modular as possible and some work has already been developed to give the user the ability to use non-linear functions to describe the inputted functions to the bodies, and to use non-linear branch function handles to allow the elements to function as step functions, these pieces of code that were not fully verified due to the large scope of the thesis can be used to establish other subroutines that can be useful for future applications.

Since the 3D kinematic program can be seen as an intermediate step to the 3D forward dynamic program, the benchmarking done focused itself on testing the capabilities of the latter. To that end, the work of González et al. was used, in which the five proposed models aimed to test specific characteristics encountered in MBS [51]. The evaluated characteristics are the ability to solve motions that occur near singular positions, the N-four-bar mechanism, which takes place in very small time scales, the Andrew's mechanism, which occurs in the presence of redundant constraints, the Bricard's mechanism, and finally the study of the ability of the program to solve stiff equations, the Flyball governor mechanism.

These five problems allowed several conclusions from the program operation. Firstly, it is possible to conclude that the program was successful in producing results for the five problems, whose results are all within the acceptable margin of 5 % difference from the *MSC Adams* solution. In some cases, the differences between solutions were so insignificant that the factor influencing the choice of penalty parameters, that resulted from the convergence studies, was the computational time of the program, which was, as expected, higher than the ones produced by *MSC Adams*.

The penalty values assumed for these problems also demonstrated that the program suffered from some numerical instability that had to be mitigated through high levels of induced numerical stiffness, Ω, which in turn induced higher α values to guarantee convergence in useful time, in comparison to the typical range of values identified in the literature Eq.(3.46). However, these high values were expected for the Bricard's mechanism, overconstrained system, and the N-four-bar mechanism, singular positions, which both constitute motions that induce numerical difficulties in the ALF routine. Furthermore, even though the choice of penalty values is usually empiric, requiring the realisation of various convergence studies for each case, the already performed studies allowed for the creation of a library that will allow future users of this tool to gain sensibility to the evolution of these parameters with each multibody model. In fact, from these studies, it is not only possible to highlight situations where the ALF is pushed to its limits, but it is also possible to draw attention to the fact that always using high penalty values may not be the best strategy for the use of this tool, since this large values will lead to ill-conditioned matrices that consequently can cause a failure of the simulation or loss of precision, producing an unprecise solution.

Additionally, it is also possible to infer from problems A02, A03 and A05 that the program correctly accepts consistent dynamic inputs as initial conditions, such as forces, torques and initial velocities, as well as, being capable of solving motions of mechanical systems that have bodies connected to force elements, namely springs and dampers. From problem A05, it is still possible to show that the *ode113* can still solve some problems that are stiff, but also that the program has the flexibility to provide different solvers that may be more adequate to stiff systems such as *ode15s* and *ode23s*, therefore increasing the scope of multibody models that the program may solve.

Accordingly, it is possible to state, that two of the objectives proposed in Chapter 1 were met by producing a program whose interface is more simple and therefore more intuitive than *MSC Adams*, while also being stable and reliable so that it can be used as a base for future developments. Finally, due to the final complexity of the program and corresponding difficulties encountered in the final debug phase of the benchmarking models, the last objective was not complete at the time of the delivery of this work and the program did not yet run a 3D suspension model, whose system as far more elements than any of the evaluated benchmarking problems. However, this should not discredit the results obtained in this work since its evaluation represents a vital step in the development of programs of this nature and the results give confidence that the general-purposed tool is capable of solving more complex 3D problems.

5.1 Future Work

Regarding the future development of the program, the next steps should be to improve the numerical stability of the forward dynamic problem, by introducing clean-ups of the round-off errors induced by *MATLAB* during the algebraic operations. Followed by efforts, to finish the verification and integration of more stabilisation methods such as the Baumgarte stabilisation and direct correction method which are already programmed but not fully verified. As well as, the expansion of the driver joints inputs to accept branch functions as it is already possible for the force elements, allowing the test of step driver inputs such as bumps.

It is also necessary to further develop the program to meet the functionalities exposed by Kortüm et al. for its application in the vehicle dynamics context [1]. Namely, the implementation of an unconditionally stable integrator, such as the Hilber-Hughes-Taylor integrator, which would allow the expansion of the complexity of the models solved by the program and allow the resolution of stiff equations.

Furthermore, routines for the modelling of tires, as well as, tracks and other vehicle dynamics characteristic subroutines should be developed and integrated into the program. Finally, it would be interesting to explore more advanced subjects, mainly the integration of FEM elements to the program, which would allow a better load calculation and the implementation of clearance and friction in the program joints, alongside track profiles.

Bibliography

[1] W. Kortüm, W. O. Schiehlen, and M. Arnold. Software tools: From multibody system analysis to vehicle system dynamics. *Mechanics for a New Mellennium*, pages 225–238, 2006. doi: 10.1007/0-306-46956-1_15.

[2] H. AB. Automated dynamic analysis of mechanical systems, 2017. URL `https://hexagon.com/company/divisions/manufacturing-intelligence/msc-software`.

[3] MathWorks. Optimumkinematics, 2012. URL `https://optimumg.com/product/optimumkinematics/`.

[4] D. S. S. Corp. Multibody dynamics simulation, 2005. URL `https://www.3ds.com/products-services/simulia/products/multibody-system-simulation/`.

[5] W. Schiehlen. *Multibody systems handbook*, volume 6. Springer Berlin Heidelberg, 1990.

[6] P. E. Nikravesh. *Computer-aided analysis of mechanical systems*. Prentice-Hall, Inc., 1988.

[7] R. N. Jazar. *Advanced dynamics: Rigid body, multibody, and aerospace applications*. John Wiley & Sons, 2011.

[8] L. H. V. B. Marques. Optimal Lap Time for a Race Vehicle. Master's thesis, Instituto Superior Técnico, University of Lisbon, Lisbon,Portugal, 2016.

[9] A. A. Shabana. *Dynamics of multibody systems*. Cambridge university press, 2003.

[10] P. Flores. *Concepts and formulations for spatial multibody dynamics*. Springer, 2015.

[11] A. B. Bhat and A. Naik. Suspension Steady-State Kinematics and Compliance Analysis Based on Linear Bushing Model. Master's thesis, Chalmers University of Technology, Göteborg,Sweden, 2020.

[12] L. Perumal. Quaternion and its application in rotation using sets of regions. *International Journal of Engineering and Technology Innovation*, 1(1):35, 2011.

[13] P. E. Nikravesh, R. A. Wehage, and O. K. Kwon. Euler Parameters in Computational Kinematics and Dynamics. Part 1. *Journal of Mechanisms, Transmissions, and Automation in Design*, 107(3): 358–365, 09 1985. ISSN 0738-0666. doi: 10.1115/1.3260722. URL `https://doi.org/10.1115/1.3260722`.

[14] P. E. Nikravesh, R. A. Wehage, and O. K. Kwon. Euler Parameters in Computational Kinematics and Dynamics. Part 2. *Journal of Mechanisms, Transmissions, and Automation in Design*, 107(3): 366–369, 09 1985. ISSN 0738-0666. doi: 10.1115/1.3260723. URL https://doi.org/10.1115/1.3260723.

[15] P. E. Nikravesh and I. S. Chung. Application of Euler Parameters to the Dynamic Analysis of Three-Dimensional Constrained Mechanical Systems. *Journal of Mechanical Design*, 104(4):785–791, 10 1982. ISSN 0161-8458. doi: 10.1115/1.3256437. URL https://doi.org/10.1115/1.3256437.

[16] S. Sarabandi and F. Thomas. A Survey on the Computation of Quaternions From Rotation Matrices. *Journal of Mechanisms and Robotics*, 11(2), 03 2019. ISSN 1942-4302. doi: 10.1115/1.4041889. URL https://doi.org/10.1115/1.4041889. 021006.

[17] J. Dye. *Development and application of computational dynamic and kinematic constrained multi-body system simulations in MATLAB*. PhD thesis, Wichita State University, 2011.

[18] M. Blundell and D. Harty. *Multibody systems approach to vehicle dynamics*. Elsevier, 2004.

[19] J. G. De Jalon and E. Bayo. *Kinematic and dynamic simulation of multibody systems: the real-time challenge*. Springer Science & Business Media, 2012.

[20] P. Flores and H. M. Lankarani. *Contact force models for multibody dynamics*, volume 226. Springer, 2016.

[21] MathWorks. Solve systems of linear equations ax = b for x, 2012. URL https://www.mathworks.com/help/matlab/ref/mldivide.html#responsive_offcanvas.

[22] Yoon, Sugjoon. *Real-time simulation of constrained dynamic systems*. PhD thesis, 1990. URL https://hdl.handle.net/2027.42/104438.

[23] Uchida, Thomas Kenji. *Real-time Dynamic Simulation of Constrained Multibody Systems using Symbolic Computation*. PhD thesis, 2011. URL http://hdl.handle.net/10012/5844.

[24] M. Wojtyra. Joint reaction forces in multibody systems with redundant constraints. *Multibody System Dynamics*, 14(1):23–46, 2005.

[25] P. E. Nikravesh. *Planar Multibody Dynamics: Formulation, Programming with MATLAB®, and Applications*. CRC press, 2018.

[26] P. E. Nikravesh. Initial condition correction in multibody dynamics. *Multibody System Dynamics*, 18 (1):107–115, 2007.

[27] L. Tagliapietra, M. Vivian, M. Reggiani, and E. Ceseracciu. Multibody systems benchmark in opensim (mbs-bos)-user manual. 2015.

[28] S. D. Conte and C. De Boor. *Elementary numerical analysis: an algorithmic approach*. SIAM, 2017.

[29] P. Deuflhard. A modified newton method for the solution of ill-conditioned systems of nonlinear equations with application to multiple shooting. *Numerische Mathematik*, 22(4):289–315, 1974.

[30] J. Lambers. Lecture 10 notes, the method of steepest descent, June 2011-2012.

[31] J. C. Meza. Steepest descent. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(6): 719–722, 2010.

[32] M. Grasmair. Minimizers of optimization problems. *To appear*, 2015.

[33] M. Lourakis and A. Argyros. Is levenberg-marquardt the most efficient optimization algorithm for implementing bundle adjustment? In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1526–1531 Vol. 2, 2005. doi: 10.1109/ICCV.2005.128.

[34] H. P. Gavin. The levenberg-marquardt algorithm for nonlinear least squares curve-fitting problems. *Department of Civil and Environmental Engineering, Duke University*, 19, 2019.

[35] Y.-x. Yuan. A review of trust region algorithms for optimization. In *Iciam*, volume 99, pages 271–282, 2000.

[36] M. Goharian, A. Jegatheesan, and G. R. Moran. Dogleg trust-region application in electrical impedance tomography. *Physiological Measurement*, 28(5):555–572, apr 2007. doi: 10.1088/0967-3334/28/5/009. URL `https://doi.org/10.1088/0967-3334/28/5/009`.

[37] F. V. Berghen. Levenberg-marquardt algorithms vs trust region algorithms. *IRIDIA, Université Libre de Bruxelles*, 1, 2004.

[38] da Silva, M. *Human Motion Analysis Using Multibody Dynamics and Optimization Tools*. PhD thesis, 2003. URL `http://web.ist.utl.pt/ist13915/docs/theses/PhD_Thesis_MiguelTavaresSilva_2003.pdf`.

[39] E. J. Haug. *Computer aided analysis and optimization of mechanical system dynamics*, volume 9. Springer Science & Business Media, 2013.

[40] G. Gadisa and H. Garoma. Comparison of higher order taylor's method and runge-kutta methods for solving first order ordinary differential equations. *Journal of Computer and Mathematical Sciences*, 8(1):12–23, 2017.

[41] L. F. Shampine. Some practical runge-kutta formulas. *Mathematics of computation*, 46(173):135–150, 1986.

[42] J. R. Dormand and P. J. Prince. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

[43] L. F. Shampine and M. W. Reichelt. The matlab ode suite. *SIAM journal on scientific computing*, 18 (1):1–22, 1997.

[44] F. Marques, A. P. Souto, and P. Flores. On the constraints violation in forward dynamics of multibody systems. *Multibody System Dynamics*, 39(4):385–419, 2017.

[45] E. Bayo and R. Ledesma. Augmented lagrangian and mass-orthogonal projection methods for constrained multibody dynamics. *Nonlinear dynamics*, 9(1):113–130, 1996.

[46] S. Yoon, R. Howe, and D. Greenwood. Geometric elimination of constraint violations in numerical simulation of lagrangian equations. 1994.

[47] C. P. Hansen. Regularization Tools - A Matlab Package for Analysis and Solution of Discrete Ill-Posed Problems. *Numerical Algorithms*, 6(March):1–35, 2008. URL `http://www2.compute.dtu.dk/~pcha/Regutools/RTv4manual.pdf`.

[48] I. Houtzager and P. Gebraad. Pbsid-toolbox regress function. `https://github.com/jwvanwingerden/PBSID-Toolbox/blob/master/private/regress.m`, 2017.

[49] E. Bayo and A. Avello. Singularity-free augmented lagrangian algorithms for constrained multibody dynamics. *Nonlinear Dynamics*, 5(2):209–231, 1994.

[50] M. González, F. González, A. Luaces, and J. Cuadrado. A collaborative benchmarking framework for multibody system dynamics. *Engineering with Computers*, 26:1–9, 2 2010. ISSN 01770667. doi: 10.1007/s00366-009-0139-0.

[51] M. González, D. Dopico, U. Lugrís, and J. Cuadrado. A benchmarking system for mbs simulation software: Problem standardization and performance measurement. *Multibody System Dynamics*, 16(2):179–190, 2006.

[52] IFtoMM. Multibody benchmark, 2015. URL `https://www.iftomm-multibody.org/benchmark/browse/`.

[53] MathWorks. Solve nonstiff differential equations - variable order method, 2006. URL `https://www.mathworks.com/help/matlab/ref/ode113.html#bu5ypth_sep_shared-tspan`.

[54] W. Blajer. Elimination of constraint violation and accuracy aspects in numerical simulation of multibody systems. *Multibody System Dynamics*, 7(3):265–284, 2002.

[55] F. González, D. Dopico, R. Pastorino, and J. Cuadrado. Behaviour of augmented lagrangian and hamiltonian methods for multibody dynamics in the proximity of singular configurations. *Nonlinear Dynamics*, 85(3):1491–1508, 2016.

[56] P. Betsch, C. Becker, M. Franke, Y. Yang, and A. Janz. A comparison of dae integrators in the context of benchmark problems for flexible multibody dynamics. *Journal of Mechanical Science and Technology*, 29(7):2653–2661, 2015.

[57] I. Fernández de Bustos, H. Uriarte, G. Urkullu, and V. García-Marina. A non-damped stabilization algorithm for multibody dynamics. *Meccanica*, 57(2):371–399, 2022.

Appendix A

Mass-Spring-Damper System

The MSC Adams model presented in Fig. A.1 is an example of the debug done where 3 cases were studied all affected by gravity. In the first case no spring was present and the y position was expected to vary linearly with the linear damper, as presented in Fig. A.2. On the second case the spring and damping coefficient were set to provoke a damped response from the body, as represented by Fig. A.3. Finally, the third case presented a system modelled without the presence of a damper and the mass was left to be actuated by gravity. Fig. A.4 shows the obtained position curves where no damping behaviour was observed.

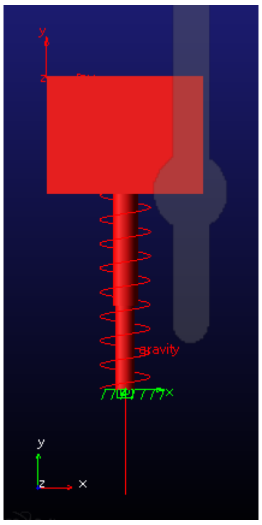


Figure A.1: Mass-spring-damper *MSC Adams* model.

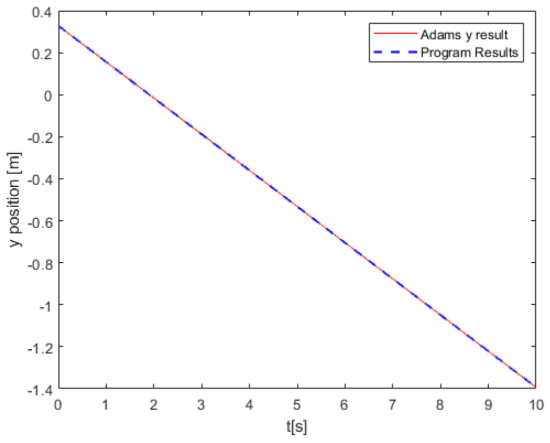


Figure A.2: Mass-damp y position variation.

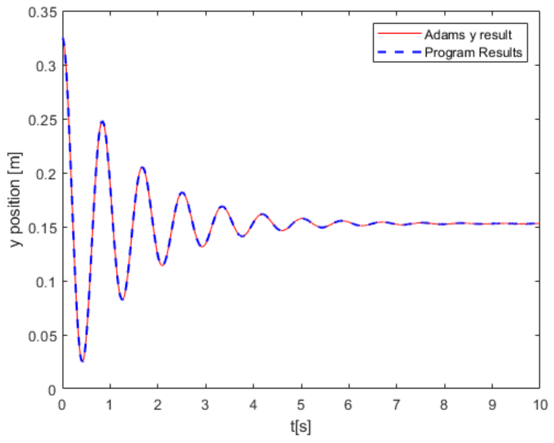


Figure A.3: Mass-spring-damper y position variation.

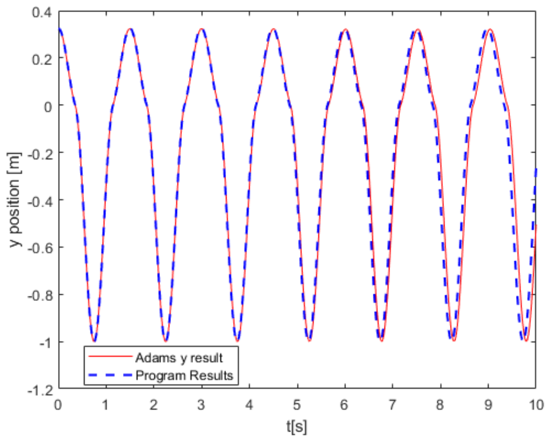


Figure A.4: Mass-spring y position variation.