Multibody 3D Simulation Tool for Vehicle Development

Tiago Miguel Serralha Pinheiro de Carvalho
tiago.serralha@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2022

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.

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

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 the 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. Therefore the objective of this work 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. The second inherent objective is to benchmark the developed code against a set of proposed benchmarking problems of MBS Simulation

tools [2], to identify program limitations and define future guidelines for its development.

2. Spatial Multibody Theory

A MBS 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. Its 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 and the local coordinate systems.

In MBSs, 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 CoM corresponds to its origin, and its axes are coincident with the principal axes of inertia of the corresponding body.

Therefore, defining methods that use the initial orientation of the local frames and consequently the orientation of the bodies is fundamental. Of all the analysed methods the robustness and effectiveness of the develop program benefits the most of the use of Euler parameters, whose application to multibody systems is extensively discussed in a series of articles written by Nikravesh ([3–5]), and arise from 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 $p = [e_0, e_1, e_2, e_3]$ that represent the rotation of the local frame around the orientational rotation axis, with the direction vector of this axis being $e = [e_1, e_2, e_3]$. These identities are defined by Eqs. (1) and (2) and are related to each other through Eq.(3). Its transformation matrix is also enunciated in Eq.(4) [7].

$$e_0 = cos(\frac{\phi}{2}) \tag{1}$$

$$\vec{e} = \vec{u}sin(\frac{\phi}{2}) \tag{2}$$

$$e_0^2 + e_1^2 + e_2^2 + e_3^2 = 1 \Leftrightarrow p^T p - 1 = 0 \tag{3}$$

$$A = 2\begin{bmatrix} e_0^2 + e_1^2 - \frac{1}{2} & e_1e_2 - e_0e_3 & e_1e_3 + e_0e_2 \\ e_1e_2 + e_0e_3 & e_0^2 + e_2^2 - \frac{1}{2} & e_2e_3 - e_0e_1 \\ e_1e_3 - e_0e_2 & e_2e_3 + e_0e_1 & e_0^2 + e_3^2 - \frac{1}{2} \end{bmatrix} \tag{4}$$

2.1. 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. In 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. The constraint equations are composed of three types of constraints, the joint constraint equations, the driving constraint equations and finally by the Euler parameters constraints, the latter due to its coordinates non-dependency.

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, 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, which 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]. The standard joint equations are represented in Tab. 1.

3. Kinematic Analysis

A kinematic simulation allows the user to have an overview of the motion of the MBS, without needing the data to fully characterise each body in the system, such as mass, moments of inertia and position of the centre of mass (CoM). As a result, during a kinematic analysis forces are not considered and the driving constraints fully define the motion of the system.

This simulation consists in the resolution of three sub-problems, the non-linear position analysis represented by Eq.(5), which comprises of calculating the final position of the system, based on the finite displacement input for the driving constraints and a given fixed initial position. Alongside with the remaining linear velocity and acceleration subproblems represented by Eqs. (6) and (7).

$$\begin{bmatrix} \Phi(q) \\ \Phi^{(d)}(q,t) \end{bmatrix} = 0 \tag{5}$$

Table 1: Standard constraint components for the Jacobian matrix and the right hand side acceleration analysis vector [6].

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	$\tilde{s}_i C_j$	$\tilde{s}_j h_j - \tilde{s}_j h_i - 2\tilde{\dot{s}}_i \tilde{s}_j$
$\Phi^{(p2,2)}$	$-\tilde{s}_i$	$-\tilde{s}_i B_i - \tilde{d} C_i$	$\tilde{s}_i$	$\tilde{s}_i B_j$	$-\tilde{s}_i(h_j^B - h_i^B) - \tilde{d} h_i - 2\tilde{\dot{s}}_i \dot{d}$
$\Phi^{(s-s,1)}$	$-2d^T$	$-2d^T B_i$	$2d^T$	$2d^T B_j$	$2d^T(h_i^P - h_j^P) - 2\dot{d}^T \dot{d}$
$\Phi^{(s,3)}$	I	C_i	$-I$	$-C_j$	$h_i^P - h_j^P$

$$\begin{bmatrix} \Phi_q \\ \Phi_q^{(d)} \end{bmatrix} \dot{q} = \begin{bmatrix} 0 \\ -\Phi_t^{(d)} \end{bmatrix} \tag{6}$$

$$\begin{bmatrix} \Phi_q \\ \Phi_q^{(d)} \end{bmatrix} \ddot{q} = \begin{bmatrix} -(\Phi_q \dot{q})_q \dot{q} \\ -(\Phi_q^{(d)} \dot{q})_q \dot{q} - 2\Phi_{qt}^{(d)} \dot{q} - \Phi_{tt}^{(d)} \end{bmatrix} \tag{7}$$

Since the finite displacement problem is non-linear, it requires the use of an iterative process for its solution. For this it is possible to use the non-linear solvers already available in *MATLAB* that are a product of the combination between the simpler Newton-Raphson method, and the steepest descent method. 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 [8] and Goharian [9], the trust-region Dogleg algorithm has a higher computational efficiency than the Levenberg-Marquardt method. 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 [9].

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. 2 [8, 9].

With the non-linear solver selected, the final combination of the three sub-problems results in the algorithm presented in Alg. 1, which is applied to the general-purposed tool of this work.

Algorithm 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**

4. Dynamic Analysis

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 [10]. Therefore, the inertial characteristics of the system must be defined, such as the inertia tensor, mass, and position of the CoM.

In forward dynamics, the generalised positions q_i

Table 2: Comparison of the efficiency, through the simulation time of the developed general-purposed tool, between the Levenberg-Marquardt and Trust Region Dogleg algorithms.

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

and velocities $\dot{q}_i$ are determined through the numerical integration of the correspondent time accelerations $\ddot{q}_i$, which are determined with recourse to the equations of motion. For a constrained system, besides contemplating the translational and rotational component of these equations, it is necessary to consider the reaction forces of each joint. These can be expressed with the resource of the augmented formulation and the Lagrange multipliers (λ) method, which through some mathematical manipulation result in a $n+m$ system of differential algebraic-equations (DAE) of Eq.(8).

$$\begin{bmatrix} M & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ -\lambda \end{bmatrix} = \begin{bmatrix} g \\ \gamma \end{bmatrix} \tag{8}$$

The dynamic analysis starts with the consistency check of the initial conditions to ensure that there are no initial constraint violations. There are two methods that can be considered for the correction of the initial conditions and were studied in detail by Nikravesh [11]. 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_i. The independent variables are picked by the user and assumed constant for the calculation of the remaining unknown variables, done through Eqs. (9) and (10).

$$\Phi(q_d, q_i^e) = 0 \tag{9}$$

$$\begin{bmatrix} \Phi_{q(d)} & \Phi_{q(i)} \\ 0 & I_i \end{bmatrix} \begin{bmatrix} \dot{q}_d \\ \dot{q}_i \end{bmatrix} = \begin{bmatrix} 0 \\ \dot{q}_i^e \end{bmatrix} \tag{10}$$

In contrast, the objective of the second method is to find a set of $\Delta' s$, Eqs. (11) and (12), that adjust 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.

$$\sigma^i = \Phi(q^i) \tag{11}$$

$$\Delta q^i = -\Phi_q^{-1}\Phi_q^T(\Phi_q M^{-1}\Phi_q^T)^{-1}\sigma \tag{12}$$

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, due to its equations non-linearity. The velocity analysis $\Delta' s$ are found through Eqs. (13) and (14) after the position adjustment.

$$\varepsilon = \Phi_q \dot{q}^e \tag{13}$$

$$\Delta\dot{q} = -M^{-1}\Phi_q^T(\Phi_q M^{-1}\Phi_q^T)^{-1}\sigma \tag{14}$$

The latter method requires less programming effort than readapting the Jacobian to solve the system of Eq.(10). Additionally, because it does not require the selection of independent coordinates it is less susceptible to yielding an improper configuration far from the desired initial configuration of the system.

After the determination of the consistent initial conditions the dynamic analysis is followed by the resolution of an initial value problem composed by a first order differential equation (ODE) system which is the product of the mathematical manipulation of the DAE system of Eq.(8).

For the resolution of this problem it was necessary to study the already available *MATLAB* ode solvers. Of these, two conditionally stable integrators were analysed, the explicit single-step Runge-Kutta Dormand-Prince pair method, or *ode45* and the Adams-Bashforth-Moulton , or *ode113*, a multistep variable-step, variable-order (VSVO) implicit predictor-corrector integrator. *Ode45* requires the minimum amount of storage but a higher number of evaluations of the function to be solved, f, per time step, while *ode113* requires a large amount of storage to store the values of previous step evaluations, but maintains a fairly low number of evaluations per time step.

With the knowledge that the computational cost of the numerical integration process is mostly due to the number of function evaluations at each time step, the performance of both ode solvers was studied and the results obtained prove that *ode113* is the most efficient option for the integration of the equations of motion, as presented in Tab. 3.

However, the numerical error inherent to the integration of the $\ddot{q}$ and $\dot{q}$ values can lead to the quadratic accumulation of significant position-level constraint violations over time [11, 12]. This occurs

Table 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.

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

because system of Eq.(8) does not use the position and velocity constraint equations explicitly, allowing for its violation during the integration process [13].

As a result it is necessary to implement methods that are capable of stabilising or eliminating these constraint violations. Three different methods were analysed: the simplistic Baumgarte stabilisation, the augmented Lagrange formulation (ALF) and the geometric direct correction method. Of these, only the geometric direct correction method is capable of completely eliminate the constraint violations, since both Baumgarte and ALF use the principles of control theory to stabilise these violations, allowing for a small deviation to occur before correcting it.

The implementation of these algorithms is straightforward. In this aspect, the Baumgarte stabilisation is the simplest to implement since it does not require the use of iterative processes as required by the ALF and the direct correction. However, in terms of robustness both the Baumgarte and the direct correction methods rely on the leading matrix of the classic Lagrange multipliers method, Eq.(8), 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.(15) [14, 15].

$$\begin{cases} 10^4 \leq \alpha \leq 10^7 \\ 0 \leq \mu \leq 1 \\ 0 \leq \omega \leq 10 \end{cases} \tag{15}$$

From these points , and the comparative work done by Marques *et al.* [16], 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, the formulation of this method is represented in Eq.(16). ALF's algorithm is presented in Alg. 2.

$$\begin{aligned} &[M + \Phi_q^T \alpha \Phi_q] \ddot{q}_{(i+1)} = \\ &= M\ddot{q}_i + \Phi_q^T \alpha(\gamma_i - 2\omega\mu(\Phi_q \dot{q}_i - \upsilon_i) - \omega^2 \Phi_i) \end{aligned} \tag{16}$$

5. Benchmarking and Results

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. 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. 1.

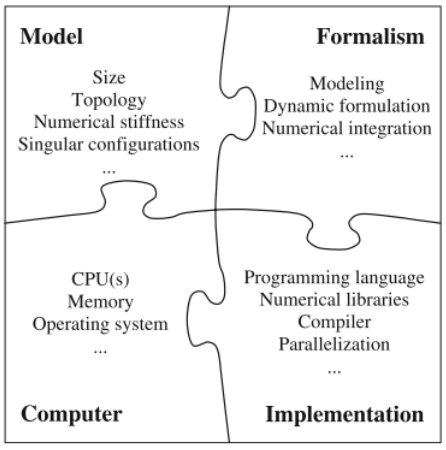

Figure 1: Puzzle of the factors that influence the benchmark of a MBS general code [18].

Of the aforementioned factors, it is necessary to

Algorithm 2: *ALF dynamic algorithm [14, 17]*

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. (5) and (6).

2: Start the initial value problem solver, *ode113*, that uses the VSVO Adams-Bashforth-Moulton predictor-corrector method.

3: Assemble the global mass matrix M, evaluate the Jacobian matrix Φ_q, construct the constraint equations Φ, determine the right-hand side of the velocities ν and accelerations γ, 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 $\ddot{q}$.

emphasise that there is no optimal formalism for all kind of MBS problems, as a result this program formalism, introduced before, is not capable of running, for example, real-time hardware loop simulations since this would require non-iterative and non-explicit methods. Furthermore, it is important to note that the developed code is run in *MATLAB* which is a interpreted language, thus requiring more memory and more computation time than the *Fortran* and *C++* languages in which the *MSC Adams* solver runs.

Since it is impossible to compare the performance and results of a code if the model used for the comparison and the error tolerance required for the simulation is not the same for the code and the reference solution, in this case for the *MSC Adams*, it was necessary to resort to a standardise benchmarking process. For this, the work developed by González *et al.* was used, which is composed by a set of five basic 2D and 3D problems that are designed to test the ability of MBS codes to solve problems with specific extreme MBS characteristics. These problems are presented in Tab. 4.

To uniformise the benchmarking process, a corresponding *MSC Adams* model was built for each of these test problems, based upon the specifications available online in a correspondent *PDF file*. Allowing for the confirmation of the literature solutions and to obtain all the data needed for the modelling of the problem in the general-purposed tool.

The results obtained from these models are summarised in Tab. 5 and Tab. 6, with all problems being successfully simulated by the program with results more than satisfactory, being all within an acceptable margin of 5 % difference in comparison to the *MSC Adams* results. However, for these problems it was also possible to infer that the ALF subroutine of the program required penalty values that are higher than the ones indicated in Eq.(15), which in general can be attributed to a combination of the high condition numbers of the leading matrix of the problems and the round-off errors that occur during the *MATLAB* algebraic operations.

This increase in the penalty values only affects the α and ω parameters which are both responsible for the convergence rate of the ALF and by the natural frequency of the penalty system of the ALF, respectively. The latter, when increased results in an overall increase in the numerical stiffness of the models which in turn improves the numerical stability of the ALF subroutine but requires the increase of the convergence rate to produce results in useful time, due to an increase in computational effort.

Furthermore, from the performed convergence studies, it was demonstrated that in contrast to the statements in the literature that the α parameter only influences the rate of convergence of the ALF

Table 4: List of problems used for the benchmarking of the general-purposed tool [19].

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

subroutine, this parameter can also influence the difference (%) of the results if the problem in study is prone to numerical instability, such as problem A02. For this problem, in which motion in singular configurations occur, the α parameters demonstrated a positive influence in the order of 0.03 % when α was increased from 10^{10} to 10^{11}.

The convergence studies also exhibited the influence that these penalty values have in the performance of the program. From the simplistic problem A01, it is possible to infer that the α, Ω and μ parameters all impact the program in accordance with the literature, in which the increase of α reduces the computational time but does not affect the differences (%) between the solutions, and the increase of ω results in a decrease in the differences (%), until it converges, but causes an increase in computational time.

However, for the problems that undergo motions that cause a rank-deficient Jacobian, such as A02 and A04, the influence of the penalty parameters changes due to the numerical difficulties induced by these situations. In these cases, it was possible to observe that the high α values can cause an increase in the computational time, this can be explained due to the fact that the condition number increases with the α value which can aggravate the already present numerical instabilities and may require the ALF to conduct more iterations to converge for a solution, resulting in an increased computational time.

It was also possible to infer, as exemplified in Fig. 2, that the differences (%) increase linearly with the elapsed simulation time. This was proven to be, in the conservative models, due to a natural small phase shift that occurs due to the numerical energy dissipation created by the round-off errors of the integration. Further conclusions, for non-conservative models, A03 and A05, were not taken, since the total mechanical energy becomes an unreliable measurement of these dissipation's.

Finally, with recourse to Tab. 6 it is possible to confirm that the use of *MATLAB* is a limitation of the program, which causes a sharp increase in the CPU-time for the resolution of the problems tested.

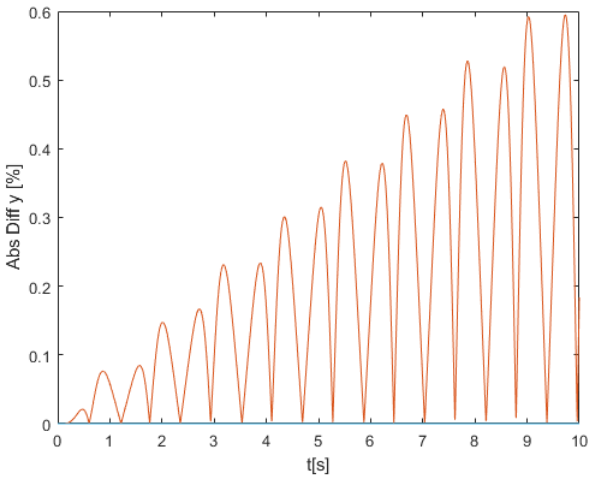

Figure 2: Evolution of the y difference (%) between the obtained solution and *MSC Adams* solution with time of the simple pendulum spherical mass.

6. Conclusions and future developments

The main goal of this work to produce a general-purposed tool capable of solving 3D kinematic and 3D foward dynamics simulations was accomplished. The objective of the tool to work based upon software's to which the team members of FST Lisboa are used to was also reached, with the tool working is based upon *MATLAB*, *SolidWorks* and *Excel*.

To reach the second inherent objective, the work of González *et al.* was used, in which the five proposed models aimed to test specific characteristics encountered in MBS. 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*.

Table 5: Summary of the results obtained for the benchmarking process. DNC - Did Not Converge

ID	Integrator Error Tolerance	Time step Δt [s]	Simulation length t_{sim} [s]	y Difference [%]
A01	1×10^{-6}	0.005	10	0.5878
A02	1×10^{-6}	0.001	10	0.1219
A03	1×10^{-6}	3×10^{-5}	0.05	0.073
A04	1×10^{-6}	0.001	10	0.0544
A05	1×10^{-6}	0.001	10	1.8796

Table 6: 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

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

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.(15). 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 **??** 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.

6.1. Future Work

Regarding the future development of the program, the next step 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 the 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.

References

[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] 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.

[3] 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`.

[4] 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`.

[5] 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`.

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

[7] 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.

[8] 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.

[9] 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`.

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

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

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

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

[14] da Silva, Miguel P T. *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`.

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

[16] 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.

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

[18] M. González, F. González, A. Luaces, and J. Cuadrado. A collaborative benchmarking framework for multibody system dynamics. *Engineering with Computers*, 26(1):1–9, 2010.

[19] 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.