Exponential Integration for Efficient and Accurate Multi-Body Simulation with Stiff Viscoelastic Contacts

Bilal Hammoud[1*], Luca Olivieri[2], Ludovic Righetti[1], Justin Carpentier[3] and Andrea Del Prete[2]

[1*]Tandon School of Engineering, New York University, New York, New York, USA.
[2]Department of Industrial Engineering, University of Trento, Via Sommarive 9, Trento, 38123, Italy.
[3]INRIA, Paris, France.

*Corresponding author(s). E-mail(s): bah436@nyu.edu;
Contributing authors: luca.olivieri-1@unitn.it; lr114@nyu.edu; justin.carpentier@inria.fr; andrea.delprete@unitn.it;

Abstract

The simulation of multi-body systems with frictional contacts is a fundamental tool for many fields, such as robotics, computer graphics, and mechanics. Hard frictional contacts are particularly troublesome to simulate because they make the differential equations stiff, calling for computationally demanding implicit integration schemes. We suggest to tackle this issue by using exponential integrators, a long-standing class of integration schemes (first introduced in the 60's) that in recent years has enjoyed a resurgence of interest. This scheme can be applied to multi-body systems subject to stiff viscoelastic contacts, leading to integration errors similar to implicit Euler, but at much lower computational costs (between 2 to 100 times faster). In our tests with quadruped and biped robots, our method demonstrated stable behaviors with large time steps (10 ms) and stiff contacts ($\mathbf{10^5}$ N/m). Its excellent properties, especially for fast and coarse simulations, make it a valuable

candidate for many applications in robotics, such as simulation, Model Predictive Control, Reinforcement Learning, and controller design. *

Keywords: keyword1, Keyword2, Keyword3, Keyword4

1 Introduction

The interest of the robotics community for fast and reliable methods to simulate multi-body systems subject to frictional contacts has been constantly growing in the last two decades [1–5]. This is reasonable considering that simulation is at the core of many robotics applications, such as the development and testing of novel controllers before deployment on hardware. Moreover, many advanced control and planning techniques, such as Model Predictive Control [6] (MPC) and Optimal Control [7], rely on the ability to predict the future behavior of the system. Finally, the current bottleneck of many learning algorithms [8, 9] is their need for huge amounts of data, which therefore can greatly benefit from fast and accurate simulation methods.

The simulation of articulated rigid multi-body systems without contacts is a solved problem [10]. The same is not the case for systems with stiff contacts, which can be treated in two ways, each leading to a hard (but different) numerical problem. The first approach, which more closely follows the physical phenomenon of contacts, consists in expressing contact forces as a function of the penetration between bodies. Often, linear spring dampers have been used in this context [5, 11]. This leads to stiff differential equations that are simple to evaluate, but difficult to integrate because of their *numerical stiffness* [12]. The second approach tries to circumvent these numerical challenges by assuming contacts to be *infinitely* rigid. This approach effectively gets rid of the numerical stiffness, but in exchange for *non-smoothness*. One method that has been particularly successful for dealing with the resulting non-smooth equations is velocity-impulse time-stepping [1, 4, 13, 14]. This has become the standard for robot simulation [15], demonstrating stable behaviors with large time steps (several ms)—even though it must solve a numerically hard Linear Complementarity Problem (LCP). Several authors have tried to improve this approach by getting rid of the strict complementarity constraints [2, 3, 16], which are the source of the numerical challenge. However, none of these approaches is currently widely accepted in the robotics community, mainly because of the unclear effects of the introduced numerical regularization/relaxations on the physics (i.e., relaxations can be interpreted as implicit spring/dampers but are not an explicit part of the modeling).

*This version of the article has been accepted for publication, after peer review and is subject to Springer Nature's AM terms of use, but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: http://dx.doi.org/10.1007/s11044-022-09818-z

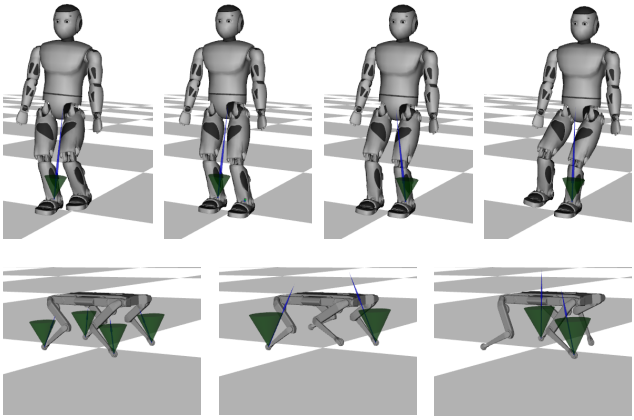


Fig. 1: Snapshots from our simulation tests with a biped and a quadruped robot.

The approach we advocate for in this article is based on a well-known soft contact model: the linear spring damper. Instead of using explicit integration schemes, which require small time steps, or implicit schemes, which require solving nonlinear systems of equations and introduce artificial viscosity leading to nonphysical behaviors, we use *Exponential Integrators* (EI) [17, 18]. EI are a long-standing class of integration schemes [19] that are particularly suited for stiff differential equations. EI were initially considered unpractical because of the computational challenges related to the matrix exponential [20]. However, novel numerical methods to compute the matrix exponential [21–23] have recently unlocked the potential of EI. This has already been used in computer graphics for simulating deformable objects, modeled as systems of particles [24–26]. This model is particularly suited for EI because the stiff part of the dynamics is linear, which however is not the case for articulated systems in contact with the environment.

Our main contribution is a simulation algorithm that exploits EI to simulate articulated robots in contact with a stiff visco-elastic environment. Particularly, this paper addresses the following questions. (1) Does the proposed simulation scheme provide an improvement in terms of speed vs accuracy when compared to classic explicit and implicit methods? (2) Is it possible to develop a simulation scheme that is less sensitive to the choice of contact stiffness and damping? (3) Is it possible to get stable simulations with increased time step (integration interval)? The last question is of particular importance for using MPC and reinforcement learning.

In order to address all the above questions in an efficient way, we apply EI only to the contact dynamics (which is stiff) while using an explicit Euler scheme for the remaining terms (which are not stiff). Our simulation results on quadruped and biped robots (see Fig. 1) show the superior performance of our method compared to standard integration schemes in terms of accuracy, speed

and stability. To our knowledge, this class of integrators has never been used before by the robotics community. The article is organized as follows. Section 2 introduces the problem of multi-body simulation and the basic theory of EI. Section 3 explains how EI can be used for multi-body simulation with bilateral contacts. This method is then extended to frictional contacts in Section 4. Section 5 discusses the implementation details of the algorithm. Section 6 presents the results and Section 7 concludes the article.

2 Background

2.1 Multi-Body Dynamics and Soft Contact Model

We want to simulate a multi-body mechanical system with the following dynamics [10]:

$$M(q)\dot{v} = u(q, v) + J(q)^\top \lambda, \tag{1}$$

where q is the robot joint configuration, v is the robot joint velocity, M is the joint space mass matrix, u contains gravity, nonlinear effects and actuator forces, J is the contact point Jacobian, and λ contains the stacked 3D contact forces. We assume a linear spring-damper contact model, which means that the contact forces λ are proportional to the inter-penetration of contacting bodies:

$$\lambda = -K\,(\underbrace{p(q) - p_0}_{\Delta p}) - B\,(\underbrace{\dot{p}(q, v) - \dot{p}_0}_{\Delta \dot{p}}), \tag{2}$$

where p and $\dot{p}$ contain the stacked 3D contact point positions and velocities, p_0 and $\dot{p}_0$ contain the stacked 3D anchor point positions and velocities, and K and B are the diagonal stiffness and damping matrices, respectively. The anchor point p_0 is a *virtual* point to which the *virtual* spring and damper are attached. It is typically set to the contact point location when contact is first detected, and as long as contacts are sticking $\dot{p}_0 = 0$. However, when slipping occurs then $\dot{p}_0 \neq 0$. A limitation of the "anchor point" model is that to generate tangential forces, some lateral motion of the contact point is always necessary. Consequently, pure static friction cannot be modeled with this approach, but it can be well approximated by using large lateral stiffnesses. Dependencies on q and v are dropped in the following to ease notation.

2.2 Explicit Integration Schemes

The classic approach to integrate this dynamical system starts by writing it in standard form. Defining the state as $x_q \triangleq (q, v)$, its dynamics is:

$$\underbrace{\frac{d}{dt}\begin{bmatrix} q \\ v \end{bmatrix}}_{\dot{x}_q} = \underbrace{\begin{bmatrix} v \\ M^{-1}(u + J^\top \lambda) \end{bmatrix}}_{f(x_q, u)} \tag{3}$$

We can integrate (3) with any numerical integration scheme, such as a high-order Runge-Kutta scheme, or even a simple explicit Euler [12] (very common in robotics):

$$x_q^+ = x_q + \delta t\, f(x_q, u) \tag{4}$$

where x_q^+ represents the next value of the state and δt is the integration time step. The problem with this approach is that for large values of K and B the differential equations (3) are *stiff* [12]. This means that they require very small integration steps for numerical stability. This is the main reason why soft contact models have been mostly abandoned in the last decade, in favor of complementarity-based models (and their relaxations) and time-stepping integration [2, 4, 27].

2.3 Exponential Integrators (EI)

EI [24–26] are integration schemes particularly suited for stiff dynamical systems for which the *stiffness* comes from a linear part of their dynamics:

$$\dot{x} = \underbrace{f(x)}_{\text{nonstiff nonlinear function}} + \underbrace{Ax}_{\text{stiff linear function}} \tag{5}$$

In this case, using an explicit integration scheme would result in the problems mentioned above. Instead, EI exploit the linearity of the stiff part of the dynamics, which can be solved *analytically* using the matrix exponential, thanks to the well-known solution of linear dynamical systems:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + b \\ x(t) &= e^{tA}x(0) + \int_0^t e^{\tau A}\, \mathrm{d}\tau\, b \end{aligned} \tag{6}$$

First-order EI apply the solution (6) to the nonlinear system (5), by interpreting $f()$ as b and assuming it remains constant during the integration step:

$$x(t) = e^{tA}x(0) + \int_0^t e^{\tau A} \mathrm{d}\tau\, f(x(0)) \tag{7}$$

Since the stiff part of the equations is integrated via the matrix exponential, large integration steps can be taken.

3 Bilateral Contacts

Our approach consists in using EI to simulate the system (3). The standard approach to apply EI to arbitrary dynamics is to use a 1-st order Taylor expansion:

$$\dot{x}_q(t_0 + t) \approx \dot{x}_q(t_0) + \frac{\partial f}{\partial x_q}\left(x_q(t_0 + t) - x_q(t_0)\right) \tag{8}$$

However, this would require two demanding computations: the dynamics Jacobian, and a matrix exponential with the size of x_q. In the following we present instead an approach that i) does not require the dynamics Jacobian, and ii) only computes a matrix exponential with twice the size of λ, which is typically smaller than x_q in legged locomotion.

To get a differential equation with the form (5) we start by projecting (1) in contact space pre-multiplying both sides by JM^{-1}:

$$J\dot{v} - \underbrace{JM^{-1}J^\top}_{\Upsilon}\lambda = J\underbrace{M^{-1}u}_{\dot{\bar{v}}} \tag{9}$$

Then we use the relationship $\ddot{p} = J\dot{v} + \dot{J}v$ to express the contact point accelerations as functions of the robot accelerations:

$$\begin{aligned} \ddot{p} - \Upsilon\lambda &= J\dot{\bar{v}} + \dot{J}v \\ \ddot{p} + \Upsilon K\Delta p + \Upsilon B\Delta\dot{p} &= \underbrace{J\dot{\bar{v}} + \dot{J}v}_{\ddot{\bar{p}}} \end{aligned} \tag{10}$$

Since for bilateral contacts $\dot{p}_0$ is always null, we have that $\ddot{p}_0 = 0$ and thus we can write the contact point dynamics as:

$$\frac{d}{dt}\begin{bmatrix}\Delta p \\ \Delta\dot{p}\end{bmatrix} = \underbrace{\begin{bmatrix} 0 & I \\ -\Upsilon K & -\Upsilon B\end{bmatrix}}_{A}\underbrace{\begin{bmatrix}\Delta p \\ \Delta\dot{p}\end{bmatrix}}_{x} + \underbrace{\begin{bmatrix}0 \\ \ddot{\bar{p}}\end{bmatrix}}_{b} \tag{11}$$

This dynamical system does not have the same form as (5) because Υ (and thus A) depends on q. However, Υ is typically a well-conditioned function, meaning that it changes little for small variations of q. The same holds for $\ddot{\bar{p}}$ (and thus b), which is why multi-body systems without contacts can typically be integrated with large time steps (≈5 ms). This means that we can approximate A and b as constants during the integration step, and therefore treat (11) as linear. We can now express the contact forces as:

$$\lambda(t) = \underbrace{\begin{bmatrix}-K & -B\end{bmatrix}}_{D} x(t) = De^{tA}x(0) + D\int_0^t e^{\tau A}\mathrm{d}\tau\, b \tag{12}$$

Substituting (12) in (3) we can compute the robot accelerations:

$$\dot{v}(t) = M^{-1}(u + J^\top\lambda(t)) = \dot{\bar{v}} + M^{-1}J^\top Dx(t), \tag{13}$$

where we consider all terms constant during the integration step, except for $x(t)$. Now we can integrate to get the new velocities v^+:

$$\begin{aligned} v^+ &= v + \int_0^{\delta t} \dot{v}(\tau)\,\mathrm{d}\tau = \\ &= v + \delta t\,\dot{\bar{v}} + M^{-1}J^\top D \int_0^{\delta t} x(\tau)\,\mathrm{d}\tau \end{aligned} \tag{14}$$

Finally we integrate twice to get the new configuration q^+:

$$\begin{aligned} q^+ &= q + \int_0^{\delta t} v(\tau)\mathrm{d}\tau = \\ &= q + \delta t\, v + \frac{\delta t^2}{2}\dot{\bar{v}} + M^{-1}J^\top D \int_0^{\delta t}\int_0^{\tau} x(\tau_1)\,\mathrm{d}\tau_1\,\mathrm{d}\tau \end{aligned} \tag{15}$$

3.1 Integration of Matrix Exponentials

Eq. (14) and (15) are straightforward to compute, except for their last terms, which are:

$$\begin{aligned} x_{int}(t) &\triangleq \int_0^t x(\tau)\,\mathrm{d}\tau \\ x_{int2}(t) &\triangleq \int_0^t \int_0^{\tau} x(\tau_1)\,\mathrm{d}\tau_1\,\mathrm{d}\tau, \end{aligned} \tag{16}$$

where

$$x(t) = e^{tA}x(0) + \int_0^t e^{\tau A}\,\mathrm{d}\tau\, b \tag{17}$$

When A is invertible we can express the integral of e^{tA} as an algebraic function of e^{tA}:

$$\int_0^t e^{\tau A}\mathrm{d}\tau = A^{-1}(e^{tA} - I) \tag{18}$$

However, A is not invertible if the contact Jacobian J is not full-row rank. Luckily, the computation of integrals involving matrix exponentials has been thoroughly investigated [28, 29]. In Section 5 we show how to compute these integrals indirectly, by simply computing the matrix exponentials of an augmented system.

3.2 Extension to non-Euclidian spaces

When q does not belong to an Euclidian space (as in the case of legged robots, where q includes the orientation of the base link) the integration of q is slightly more complicated (while the integration of v remains unchanged). Given the following definition:

$$v_{mean} \triangleq v + \frac{\delta t}{2}\dot{\bar{v}} + \frac{1}{\delta t}M^{-1}J^\top D x_{int2}(\delta t) \tag{19}$$

The integration step of q is computed as:

$$q^+ = \text{integrate}(q, \delta t\, v_{mean}), \tag{20}$$

where the function *integrate(·)* performs integration in the non-Euclidian space of q.

4 Frictional Contacts

So far we have assumed that contact forces were bilateral. However, we typically want to simulate unilateral contacts, where forces oppose penetration but do not oppose detachment of bodies. Assuming that the contact forces are expressed in a local reference frame with the z direction aligned with the contact normal, unilateral forces must satisfy:

$$f_i^z \geq 0 \qquad \forall i \tag{21}$$

Moreover, tangential forces are typically limited as well. Assuming a Coulomb friction model we have:

$$\sqrt{(f_i^x)^2 + (f_i^y)^2} \leq \mu f_i^z \qquad \forall i, \tag{22}$$

where $\mu \in \mathbb{R}^+$ is the coefficient of friction[1]. We can represent the constraints (21) and (22) as $\lambda \in \mathcal{K}_\mu$, with $\mathcal{K}_\mu$ being a second-order cone.

4.1 Force Projection

To account for these constraints, when the value of $\lambda(t)$ computed by (12) is outside $\mathcal{K}_\mu$, we should project it on the boundaries of $\mathcal{K}_\mu$. However, we do not know how to check this constraint in continuous time. In the same spirit of time-stepping simulators [1], we suggest to check friction constraints on the average value of $\lambda(t)$ during the integration step, which is:

$$\bar{\lambda} \triangleq \frac{1}{\delta t} \int_0^{\delta t} \lambda(\tau)\, \mathrm{d}\tau = \frac{1}{\delta t} Dx_{int}(\delta t) \tag{23}$$

If $\bar{\lambda} \notin \mathcal{K}_\mu$, then we compute its projection on the boundaries of the friction cone $\lambda_{pr} = \text{proj}_{\mathcal{K}_\mu}(\bar{\lambda})$ and we use it to compute the next state:

$$\begin{aligned} \dot{v}_{pr} &\triangleq M^{-1}(u + J^\top \lambda_{pr}) \\ v^+ = v + \delta t\, \dot{v}_{pr}, \qquad q^+ &= q + \delta t\, v + \frac{\delta t^2}{2} \dot{v}_{pr} \end{aligned} \tag{24}$$

[1] We assume that static and dynamic coefficients of friction are equal.

Note that in case $\bar{\lambda} \in \mathcal{K}_\mu$, then $\lambda_{pr} = \bar{\lambda}$ and the velocity update in (24) is equivalent to (14). However, the position update in (24) approximates the double integral of $\lambda(t)$ assuming a constant force (λ_{pr}), and so it is not equivalent to (15) in general. In order to exploit also the double integral of $x(t)$, we can check the friction cone constraints on the average of the average $\lambda(t)$, computed as:

$$\bar{\bar{\lambda}} \triangleq \frac{2}{\delta t^2} \int_0^{\delta t} \int_0^{\tau} \lambda(\tau_1)\, \mathrm{d}\tau_1 \mathrm{d}\tau = \frac{2}{\delta t^2} Dx_{int2}(\delta t) \tag{25}$$

If $\bar{\bar{\lambda}} \notin \mathcal{K}_\mu$, then we project it on the boundaries of the friction cone $\lambda_{pr2} = \mathrm{proj}_{\mathcal{K}_\mu}(\bar{\bar{\lambda}})$ and we use it to compute the next position:

$$\begin{aligned} \dot{v}_{pr2} &\triangleq M^{-1}(u + J^\top \lambda_{pr2}) \\ q^+ &= q + \delta t\, v + \frac{\delta t^2}{2} \dot{v}_{pr2} \end{aligned} \tag{26}$$

Using (26) for the position update and (24) for the velocity update, both updates are equivalent to the original ones in case of no slippage.

4.2 Anchor point update

When slippage occurs, the tangent anchor point state $(p_0^t, \dot{p}_0^t)$ (where the index t indicates the tangent directions) changes, which has two main implications. First, the assumption $\ddot{p}_0 = 0$ that we took to write the contact point dynamics as (11) is no longer valid. This means that, during slippage, (11) is an approximation of the contact point dynamics, based on a "business as usual" assumption (i.e., that the anchor point p_0 continues slipping at constant velocity). Second, the anchor point state should be updated so that the contact forces at the end of the time step are inside the friction cones. When a contact is slipping, the tangent anchor point velocity converges to $\dot{p}^t$. We show it now for the case of a 2D contact, but a similar reasoning can be applied to the 3D case. While a contact is slipping, the tangential force λ^t remains on the boundary of the friction cone, so we have:

$$\begin{aligned} \dot{\lambda}^t &= \mu \dot{\lambda}^n \\ K^t(\dot{p}_0^t - \dot{p}^t) + B^t(\ddot{p}_0^t - \ddot{p}^t) &= \mu \dot{\lambda}^n \\ (\ddot{p}_0^t - \ddot{p}^t) &= -(B^t)^{-1} K^t (\dot{p}_0^t - \dot{p}^t) + \mu (B^t)^{-1} \dot{\lambda}^n \end{aligned} \tag{27}$$

The last equation shows that, if $\dot{\lambda}^n = 0$, we have an exponential convergence to zero of $(\dot{p}_0^t - \dot{p}^t)$, with rate $(B^t)^{-1}K^t$. Since typically $(B^t)^{-1}K^t$ is large, whereas $\mu(B^t)^{-1}\dot{\lambda}^n$ is small, we can expect this convergence to be fast. For instance, if $(B^t)^{-1}K^t = 10^3$ and $\dot{\lambda}^n = 0$, then after 3 ms $(\dot{p}_0^t - \dot{p}^t)$ will be 5% of its initial value. Given this fast convergence, we neglect the transient and

as soon as slippage starts we set $\dot{p}_0^t := \dot{p}^t$. Then, we compute p_0^t so that the contact force is on the boundary of the friction cone:

$$\begin{aligned} \lambda &:= \mathrm{proj}_{\mathcal{K}_\mu}(\lambda) \\ p_0^t &:= p^t + (K^t)^{-1}\lambda^t \end{aligned} \tag{28}$$

5 Computational Aspects

The computational bottleneck of the presented approach is the computation of x_{int} and x_{int2} defined in (16). This section shows how to compute these quantities with a matrix exponential, and how this computation can be sped up.

5.1 Computing x_{int} and x_{int2}

Using the results presented in [29] we can compute x_{int} and x_{int2} as:

$$\begin{bmatrix} x_{int}(t) & x_{int2}(t) \end{bmatrix} = \begin{bmatrix} I_n & 0_{n\times 3} \end{bmatrix} e^{t\bar{A}} \begin{bmatrix} 0_{(n+1)\times 2} \\ I_2 \end{bmatrix} \tag{29}$$

where n is the size of A, and $\bar{A} \in \mathbb{R}^{(n+3)\times(n+3)}$ is an augmented matrix:

$$\bar{A} \triangleq \begin{bmatrix} A & b & x(0) & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{30}$$

5.2 Computing the Matrix Exponential

Using (29) we have transformed the problem of computing (16) into a matrix exponential evaluation. Computing the matrix exponential is a challenging but well-understood numerical problem [21–23, 30]. We have used as starting point the scaling&squaring method, as revisited by Higham [21], a widely used method for computing the exponential of small-medium size dense matrices. The method scales the matrix by a power of 2 to reduce the norm to order 1, computes a Padé approximant to the matrix exponential, and then repeatedly squares to undo the effect of the scaling. A Padé approximant of a function is its "best" approximation achievable by a ratio of two polynomials $D_j(\cdot)$, $N_j(\cdot)$ of order j:

$$e^A \approx D_j(A)^{-1}\, N_j(A) \tag{31}$$

These approximants are only accurate around zero, so they cannot be used directly if $\|A\|$ is large. When that is the case, the scaling&squaring method is used to reduce $\|A\|$ by exploiting this property of the exponential:

$$e^A = (e^{A/(2^s)})^{2^s} \tag{32}$$

The integer scaling parameter s is chosen so that $\|e^{A/(2^s)}\|$ is sufficiently small.

5.3 Boosting the Matrix Exponential Computation

Our problem has two features that we can exploit to speed up computation:

1. We do not need double machine precision, i.e. $\approx 10^{-16}$, (which is the target of the algorithm of [21]) because we are typically fine with much larger numerical integration errors, e.g. $\approx 10^{-4}$.
2. We do not need the whole matrix exponential, but only its product with a 2-column matrix, as shown in (29).

The first point is easily exploitable. The choice of the scaling parameter s and the polynomial order j is usually optimized to achieve double machine precision with the minimum amount of matrix-matrix multiplications. We have empirically found that for our tests we can set $s = 0$ and use a relatively low order $j \in [1, 2, 3, 5, 7]$, corresponding to $[0, 1, 2, 3, 4]$ matrix-matrix multiplications, respectively. Which polynomial order is optimal depends on the specific test, and is discussed in the next section.

Regarding the second point, given a matrix V, we can directly compute the product $e^A V$ by performing operations in the following order:

$$\begin{aligned} V_1 &:= N_j(A)\, V \\ e^A V &:= D_j(A)^{-1}\, V_1 \end{aligned} \tag{33}$$

This is faster than computing e^A and then multiplying it times V because we have to solve the linear system with a much smaller right-hand-side (V_1 rather than N_j).

Finally, we have also observed that the preprocessing step suggested in [21], which uses *matrix balancing*, is extremely effective at reducing $\|A\|$ in our tests. This is crucial to achieve accurate results with low polynomial orders, therefore speeding up computations. Further details can be found in our open-source online repository[2].

6 Results

We assess the performance of our simulation algorithm (*Expo*) comparing it to Implicit Euler (*Eul-imp*), Runge-Kutta 4 (*RK4*) and explicit Euler (*Eul-exp*). Our implementation of *Eul-imp* is described in the Appendix. Our implementation of *RK4* is standard, whereas *Eul-exp* was implemented as follows:

$$v^+ = v + \delta t\, \dot{v}, \qquad q^+ = \text{integrate}(q, \delta t\, v + \frac{\delta t^2}{2}\, \dot{v}) \tag{34}$$

Our results try to answer to the following questions:

[2] https://github.com/andreadelprete/consim

Table 1: Controller time steps.

Test	δt_c [ms]	Test	δt_c [ms]
Solo-squat	40	Solo-trot	2
Solo-jump	10	Romeo-walk	40

1. Can our approach (compared to the others) achieve higher accuracy for equal computation time, or equal accuracy for smaller computation time? (Section 6.3)
2. How sensitive is the simulator accuracy to contact stiffness and damping? (Section 6.4)
3. What is the maximum integration time step that results in a *stable*[3] motion? (Section 6.5)
4. How accurately can (11) predict future contact forces when assuming constant A and b? (Section 6.6)
5. How much computation time is spent in the different operations of our simulator? Is there room for improvement? (Section 6.7)

6.1 Accuracy Metric

Following an approach similar to [15], we measure accuracy with a *local integration error*. We compute the ground truth trajectory $x_q(t)$ using the simulator under analysis with an extremely small time step $\delta t = 1/64$ ms. Let us define $\hat{x}_q(t;\ t - \delta t_c, x_q(t - \delta t_c))$ as the state at time t obtained by numerical integration starting from the ground-truth state $x_q(t - \delta t_c)$, where $\delta t_c (\geq \delta t)$ is the time step of the controller. We define the *local* integration error as the error accumulated over one control time step:

$$e(t) \triangleq \|x_q(t) \ominus \hat{x}_q(t;\ t - \delta t_c, x_q(t - \delta t_c)\|_\infty$$

where $\ominus$ is a difference operator on the space of x_q. In the numerical integration literature [12] the *local* integration (or truncation) error is typically defined using the integration step δt rather than the controller step δt_c. We chose to use the controller step to make errors comparable across tests with different integration steps (as in [15]).

6.2 Test Description

To evaluate the trade-off between accuracy and computation time we tested each simulator with different time steps. For *Expo*, *RK-4*, *Eul-imp* we have started from $\delta t = 1/8$ ms up to the controller time step $\delta t = \delta t_c$ with a logarithmic step of 2 (i.e. 1/8, 1/4, ..., $\delta t_c/2$, δt_c). For *Eul-exp* we have used the same approach, but starting from a value of δt resulting in roughly the same computation time of *Expo*. For every test we have set δt_c to the largest value that still ensured control stability (see Table 1). Since our main interest lies in legged robots, our tests focused on quadrupeds and bipeds:

[3] We say that a simulation is "stable" if the robot state remains bounded.

- *Solo-squat*: Quadruped robot Solo [31] performing a squatting motion.
- *Solo-jump*: Quadruped robot Solo jumping in place.
- *Solo-trot*: Quadruped robot Solo trotting forward.
- *Romeo-walk*: Humanoid robot Romeo [32] taking two walking steps.

It is important to note that the quadruped Solo has a total of 13 links, 18 degrees of freedom, 12 of which are actuated revolute joints, and 4 contact points (one on each foot). As for the humanoid Romeo, it consists of 32 links, 37 degrees of freedom, 31 actuated revolute joints and a total of 8 contact points (four on each foot). In all tests, the control torques have been computed with a feedback controller, either a linear controller or a Task-Space Inverse Dynamics controller. If not specified otherwise, we have used a contact stiffness $K = 10^5$ N/m, and a contact damping $B = 300$ Ns/m, which are reasonable values for contacts with a hard floor. For homogeneity we have used the same value of friction coefficient $\mu = 1$ across all our tests, even though this large friction was only needed for control stability of the quadruped jumping motion. Besides testing the default *Expo* simulator, we also tested 5 other versions of the same scheme where we used a reduced polynomial order in the Padé approximant of the matrix exponential. This leads to a reduced number of matrix-matrix multiplications (mmm), between 0 and 4 (see Section 5.3). This results in a faster but potentially less accurate computation of the matrix exponential.

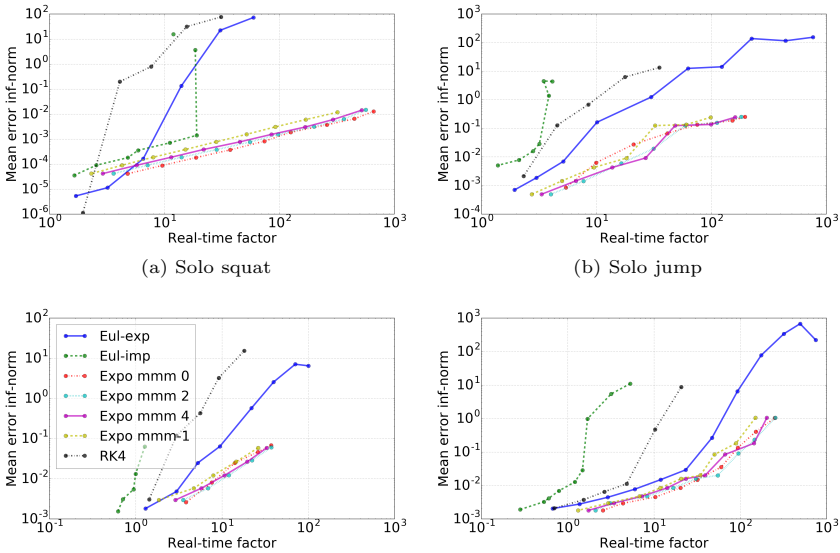
All the code has been implemented in C++ and binded with Python. For all dynamics computation we have used the Pinocchio library [33].

6.3 Accuracy-Speed Results

Fig. 2 and 3 summarize the results for the four tests. Fig. 2 plots *local errors* vs *real-time factor*, which measures how many times the simulation was faster than real time. Fig. 3 instead plots *local errors* vs *integration time step*. Even though our main interest is in the trade-off between computation time and accuracy, which is depicted in Fig. 2, we decided to report also the accuracy as a function of integration time in Fig. 3, to provide more information about the behavior of the different methods.

Overall, *Expo* outperformed the other methods in all tests, showing faster computation for equal accuracy, or greater accuracy for equal computation time. Surprisingly, the second best method overall was the simple *Eul-exp*, even though it was partially beaten by *Eul-imp* in "solo-squat". *RK4* was comparable to *Eul-exp* for small time steps, but surprisingly worse for large time steps. These results show a sudden increase of integration error of *Eul-exp* and *RK4* for large real-time factors—corresponding to large δt. This is because of the poor stability of explicit methods.

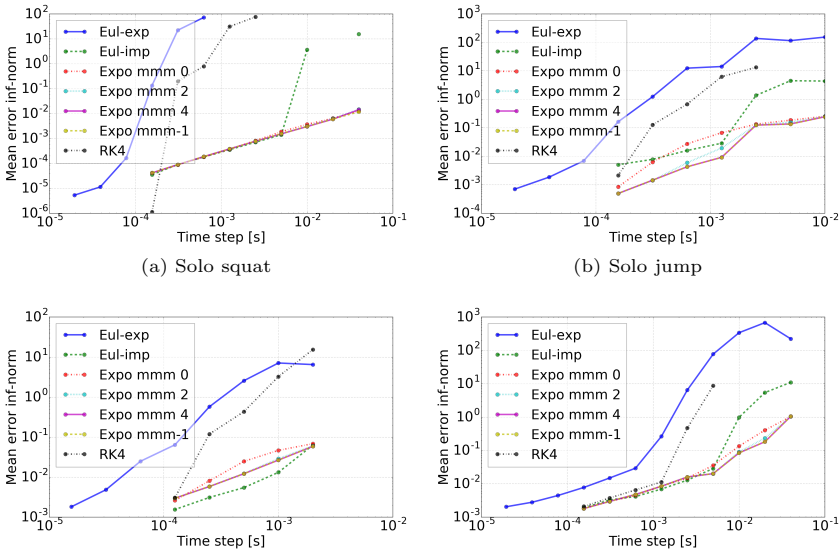
Eul-imp sometimes failed to converge to the desired error threshold (10^{-6}), in particular when using large integration steps. This is not surprising because the system dynamics is non-continuous at impacts, and *Eul-imp* uses a gradient-based method (Newton) that is suited for smooth systems. Despite



(a) Solo squat (b) Solo jump

(c) Solo trot (d) Romeo walk

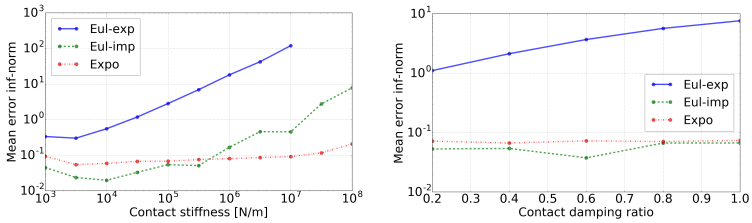
Fig. 2: Local integration errors vs real-time factors. The label *mmm-1* in the legend corresponds to using the default number of matrix-matrix multiplications (mmm) in the computation of the matrix exponential.



(a) Solo squat (b) Solo jump

(c) Solo trot (d) Romeo walk

Fig. 3: Local integration errors vs integration time step.



(a) Varying contact stiffness (with fixed damping ratio of 0.5)

(b) Varying contact damping ratio (with fixed stiffness of 10^5 N/m).

Fig. 4: Local integration errors vs contact stiffness and damping ratio for the "solo trot" test using a fixed integration time step for *Eul-exp* (1/2 ms), *Expo* (2 ms) and *Eul-imp* (2 ms).

this, Fig. 3 shows that in most cases *Eul-imp* gives integration errors similar to *Expo* for the same integration time step. More precisely, *Eul-imp* is almost indistinguishable from *Expo* in "solo-squat" for $\delta t \leq 5$ ms, whereas it is unstable for larger δt. *Eul-imp* is a bit worse than *Expo* in "solo-jump" and "romeo-walk" (but only for large time steps), and slightly better in "solo-trot". Overall, *Eul-imp* performed similarly to *Expo* for the same δt, but resulted in much larger computation times, making it often the worst one in terms of accuracy-speed trade-off.

Expo instead shows a graceful degradation of accuracy for large real-time factors, making it an excellent candidate for fast low-accuracy simulations, which are typically desirable in MPC. In general, the *Expo* versions using a reduced number of mmm outperformed the standard *Expo*, but which number of mmm is optimal depends on the specific test and time step. As expected, when δt is smaller we can use a lower number of mmm. Automatically finding the optimal number of mmm is an interesting direction for future work.

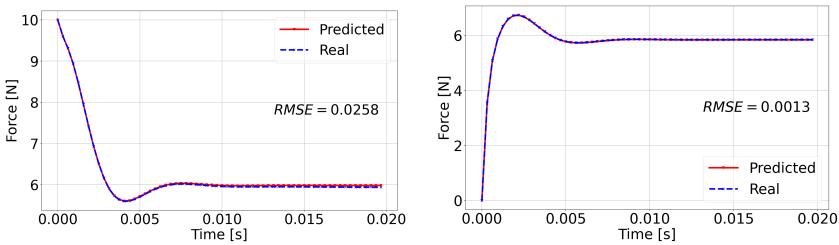
6.4 Stiffness and Damping

This subsection investigates the sensitivity to contact stiffness and damping ratio of *Expo*, *Eul-exp* and *Eul-imp*. The damping ratio is defined as $\frac{B}{2\sqrt{K}}$. A damping ratio of 1 corresponds to a *critically damped* contact. These results are based on the "solo-trot" scenario. For *Eul-exp* we have used δt=1/2 ms (real-time factor ≈50). Then, we have set δt=2 ms for *Expo* so that it had roughly the same computation time, and δt=2 ms for *Eul-imp*, so that it performed similarly to *Expo* (even though with much larger computation times).

Fig. 4 shows the local integration error as we vary the contact stiffness (with fixed damping ratio) and the damping ratio (with fixed contact stiffness). *Expo* performs consistently as damping ratio and stiffness increase up to $K = 10^8$, which roughly corresponds to a ground penetration of 0.01 mm for 100 kg of weight on a single contact point. The error of *Eul-exp* instead is highly affected by both stiffness and damping. *Eul-imp* performed slightly better than *Expo*

Table 2: Maximum integration time steps to achieve a stable motion.

Test	*Expo* δt [ms]	*Eul-exp* δt [ms]	*Eul-imp* δt [ms]
Solo-squat	40	$40/512 \approx 0.08$	$40/2 = 20$
Solo-jump	10	$10/64 \approx 0.16$	$10/2 = 5$
Solo-trot	2	$2/16 \approx 0.13$	2
Romeo-walk	$40/8 = 5$	$40/32 = 1.25$	$40/4 = 10$



(a) Contact velocity at impact: 0.1 m/s. (b) Zero contact velocity at impact.

Fig. 5: Comparison of contact forces in normal direction with forces predicted using matrix exponential.

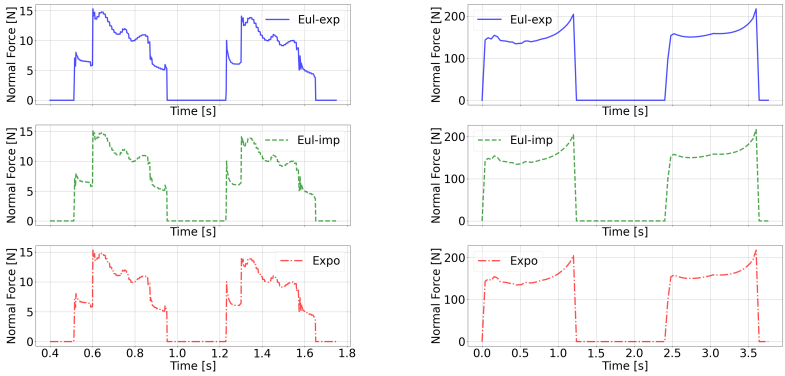
in most cases (but at the cost of being 50 times slower), except for very stiff contacts ($K \geq 10^6$), where it led to larger errors.

6.5 Stability

To test the stability of the simulators we have repeated the previous tests, but without resetting the state to the ground truth after every control loop. Table 2 reports, for *Expo*, *Eul-exp* and *Eul-imp*, the largest integration time step for which the system remained stable. *Expo* and *Eul-imp* showed similar stability, both remaining stable for large time steps, mostly between 5 and 40 ms. In the tests "solo-squat" and "solo-jump" *Expo* was stable even with a larger time step than *Eul-imp*, whereas the opposite happened in "romeo-walk". *Eul-exp* instead showed poor stability, needing a time step between 4 and 512 times smaller than *Expo* to remain stable.

6.6 Force Prediction

To gain some insights into the internal computations of *Expo*, we show in Fig. 5 the normal contact forces predicted with (11) assuming constant A and b—which is the key assumption of our method. Since A and b depend on q and v, which vary during the time step, one could expect that neglecting their variations would result in significant force prediction errors. However, Fig. 5 shows that the force prediction can be accurate over a rather long time horizon (20 ms). These forces were generated at the beginning of the "solo-squat" test, using different initial velocities. Since a linear spring-damper model is used, a sudden discontinuity in the contact force is expected when a point reaches



(a) Quadruped Trotting. (b) Biped Walking.

Fig. 6: Normal contact force during quadruped trotting and biped walking.

Table 3: Computation times of *Expo* for "solo-trot", using zero mmm for the matrix exponential (in parentheses the values using the standard matrix exponential routine).

Operation	Mean Time [μs]	Percentage of Total Time [%]
step	39 (94)	100
computeIntegrals	13 (67)	33 (72)
prepareExpLDS	13	33 (14)
computeContactForces	8	20 (8)
Eul-exp step	9	-

contact with a non-zero velocity. To demonstrate this, the normal contact force at a single contact point is plotted for the cases of the trotting quadruped and the walking biped in Fig. 6. Depending on the velocity at contact time, a finite jump in the contact force is observed.

6.7 Computation Times

We report here a breakdown of the computation time of our method. The times shown in Tab. 3 are for the "solo-trot" test, which means that $v \in \mathbb{R}^{18}$ and most of the times $\lambda \in \mathbb{R}^6$. Most computation time (86%) is spent in three operations: `computeIntegrals`, `prepareExpLDS`, and `computeContactForces`. `computeIntegrals` boils down to computing a matrix exponential. This takes 72% of the total time when using a standard `expm` routine (without balancing and reduced matrix-matrix multiplications), but it goes down to 33% with our optimized version using zero matrix-matrix multiplications—we have

seen in Fig. 2 that often this results in only a small loss of integration accuracy. The preparation of the linear dynamical system (11) (`prepareExpLDS`, which includes the computations of $h(q, v)$ with RNEA, $M(q)$ with CRBA, and Υ with a custom sparse Cholesky decomposition) takes an equal amount of time: 13 μs on average, namely 33% of the total time. The third operation (`computeContactForces`) takes 20% of the total time, and it includes the computation of all kinematic quantities (contact point positions, velocities, accelerations, Jacobian) and the contact detection.

We believe that computation times could be improved, especially for the first two operations. In `computeIntegrals` we could test novel techniques [23] to compute the matrix exponential, exploit the sparse structure of the matrix A, and warm-start the computation using quantities computed at the previous cycle. In `prepareExpLDS`, the inverse contact-space inertia matrix Υ could be computed faster using a customized algorithm, rather than with products between J, M^{-1} and $J^\top$ [34]. Overall, it seems impossible to reach the same efficiency of a simple *Eul-exp* step (9 μs), but we think we could reach computation times in the range [20, 30] μs.

7 Conclusions

This paper has presented a new approach to simulate articulated systems subject to stiff visco-elastic frictional contacts. The novelty of the approach lies in the numerical integration, which applies a first-order Exponential Integrator scheme to the contact point dynamics to obtain a time-varying expression of the contact forces. These contact forces are then integrated analytically, exploiting theoretical results on the integrals of the matrix exponential [29], and advanced numerical algorithms for its fast computation [21]. Comparison with standard integration schemes, both implicit and explicit, highlighted the benefits of the proposed approach in terms of speed-accuracy trade off, and stability. Overall, the proposed approach performed similarly to an implicit scheme in terms of stability and accuracy, but without the excessive computational burden.

Given its good behavior in the high-speed/low-accuracy regime, we believe that this simulation technique could be an excellent candidate for MPC. To do that, we will need to differentiate the integration scheme, which should be feasible. We also plan to investigate the improvement, in terms of computational efficiency, of the needed dynamics quantities and the matrix exponential.

Appendix A Implicit Euler

This Appendix reports some details about our implementation of implicit Euler. The state of our robots lies in a Lie group, so we have taken into account the derivatives of the `integrate` and `difference` functions, which need to be used in place of the simple $+$ and $-$ operators:

$$\text{difference}(x^+, \text{integrate}(x, \delta t f(x^+, u))) = 0 \tag{A1}$$

We have computed the dynamics Jacobian using the appropriate functions of the *Pinocchio* library. In particular, we have used the analytical derivatives of the `ABA` function (28 μs for the Solo quadruped robot), and we have added to it the derivatives of our contact model (which are fast to compute because they depend on already computed quantities). For the line search we have used the `ABA` function (8 μs for the Solo quadruped robot). This resulted in a rather efficient implementation of implicit Euler.

For solving the nonlinear equations we have implemented Newton's method with line search and regularization. For solving the linear system of equations we have used the LU decomposition with partial pivoting provided by the *Eigen* library. We initialized the search with the solution of an explicit Euler integration (which proved to be better than initializing with the current state).

We have measured where computation time is spent in implicit Euler, to make sure that our implementation was efficient and to reason about potential improvements. For the "solo-squat" test, with an integration step $\delta t = 10$ ms, the average computation time was 0.92 ms. *Expo* was roughly 10 times faster for the same test, and gave roughly the same integration error. The Newton's method took on average 8 iterations to converge to an error norm below 10^{-6}. Computation time was distributed in this way[4]:

- Prepare linear system: 51%
  - Dynamics Jacobian: 29%
    - ABA derivatives: 24%
    - Contact model derivatives: 4%
  - Lie-group derivatives: 21%
- Line search: 29%
- Solve linear system: 12%
- Other: 8%

Most of the time was spent preparing the linear system (51%) and performing the line search (29%). Given the high efficiency of the *Pinocchio* library, we believe that there is not much room for improvement in the steps "ABA derivatives" and "Line search" (which mainly consists of calls to `ABA`, `integrate` and `difference`). The only significant room for improvement is in the "Lie-group derivatives" step, which includes the two matrix-matrix multiplications between the dynamics Jacobian and the sparse Jacobians of the `integrate` and `difference` functions. Since these Jacobians are sparse, these multiplications could be computed more efficiently than what we are currently doing. However, this could lead to a computational gain of at most 15%. Therefore, there seems to be little hope to make implicit Euler 10 times faster (i.e. as efficient as *Expo*).

[4] Percentages do not add up to 100% because of rounding errors.

References

[1] Anitescu, M., Hart, G.D.: A constraint-stabilized time-stepping approach for rigid multibody dynamics with joints, contact and friction. International Journal for Numerical Methods in Engineering **60**(Jan 2003), 2335–2371 (2004)

[2] Todorov, E.: Implicit nonlinear complementarity: A new approach to contact dynamics. In: 2010 IEEE International Conference on Robotics and Automation, pp. 2322–2329 (2010). IEEE

[3] Todorov, E.: Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo. In: Proceedings - IEEE International Conference on Robotics and Automation, pp. 6054–6061 (2014)

[4] Hwangbo, J., Lee, J., Hutter, M.: Per-Contact Iteration Method for Solving Contact Dynamics. IEEE Robotics and Automation Letters (RAL) **3**(2), 0–8 (2018)

[5] Drumwright, E.: An Unconditionally Stable First-Order Constraint Solver for Multibody Systems. arXiv preprint arXiv:1905.10828 (2019) https://arxiv.org/abs/1905.10828

[6] Tassa, Y., Erez, T., Todorov, E.: Synthesis and stabilization of complex behaviors through online trajectory optimization. In: Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference On, pp. 4906–4913 (2012)

[7] Von Stryk, D.M.O.: Numerical solution of optimal control problems by direct collocation. In: Optimal Control, pp. 129–143. Springer, ??? (1993)

[8] Mansard, N., Del Prete, A., Geisert, M., Tonneau, S., Stasse, O.: Using a Memory of Motion to Efficiently Warm-Start a Nonlinear Predictive Controller. In: IEEE International Conference on Robotics and Automation, pp. 2986–2993 (2018)

[9] Viereck, J., Kozolinsky, J., Herzog, A., Righetti, L., Aug, R.O.: Learning a Structured Neural Network Policy for a Hopping Task. IEEE Robotics and Automation Letters (RAL) **3**(4) (2018) https://arxiv.org/abs/arXiv:1710.00022v2

[10] Featherstone, R.: Rigid Body Dynamics Algorithms. Springer, New York, NY (2014)

[11] Yamane, K., Nakamura, Y.: Stable penalty-based model of frictional contacts. Proceedings - IEEE International Conference on Robotics and

Automation **2006**(January), 1904–1909 (2006)

[12] Ascher, U.M., Petzold, L.R.: Computer Methods for Ordinary Differential Equations and Differential-algebraic Equations. Siam, Philadelphia, PA (1998)

[13] Anitescu, M.: A Fixed Time-Step Approach for Multibody Dynamics with Contact and Friction. In: IEEE International Conference on Intelligent Robots and Systems, vol. 4, pp. 3725–3731 (2003)

[14] Peiret, A., Andrews, S., Kövecses, J., Kry, P.G., Teichmann, M.: Schur complement-based substructuring of stiff multibody systems with contact. ACM Trans. Graph. **38**(5) (2019). https://doi.org/10.1145/3355621

[15] Erez, T., Tassa, Y., Todorov, E.: Simulation Tools for Model-Based Robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX. In: International Conference on Robotics and Automation (2015)

[16] Drumwright, E., Shell, D.A.: Modeling contact friction and joint friction in dynamic robotic simulation using the principle of maximum dissipation. In: Springer Tracts in Advanced Robotics, vol. 68, pp. 249–266 (2010)

[17] Loffeld, J., Tokman, M.: Comparative performance of exponential, implicit, and explicit integrators for stiff systems of ODEs. Journal of Computational and Applied Mathematics (2013)

[18] Chen, Y.J.E., Sheen, S.H., Ascher, U.M., Pai, D.K.: Siere: A hybrid semi-implicit exponential integrator for efficiently simulating stiff deformable objects. ACM Trans. Graph. **40**(1) (2020). https://doi.org/10.1145/3410527

[19] Certaine, J.: The solution of ordinary differential equations with large time constants. Mathematical methods for digital computers **1**, 128–132 (1960)

[20] Moler, C., Van Loan, C.: Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later. SIAM Review **45**(1), 3–49 (2003) https://arxiv.org/abs/arXiv:1011.1669v3

[21] Higham, N.J.: The scaling and squaring method for the matrix exponential revisited. SIAM Journal on Matrix Analysis and Applications **26**(4) (2005)

[22] Al-Mohy, A.H., Higham, N.J.: Computing the Action of the Matrix Exponential, with an Application to Exponential Integrators. SIAM Journal of Scientific Computing **33**(2), 488–511 (2011)

[23] Sastre, J., Ibáñez, J., Defez, E.: Boosting the computation of the matrix exponential. Applied Mathematics and Computation **340**(August), 206–220 (2019)

[24] Michels, D.L., Sobottka, G.A., Weber, A.G.: Exponential integrators for stiff elastodynamic problems. ACM Transactions on Graphics **33**(1) (2014)

[25] Chen, Y.J., Ascher, U.M., Pai, D.K.: Exponential Rosenbrock-Euler Integrators for Elastodynamic Simulation. IEEE Transactions on Visualization and Computer Graphics **24**(10), 2702–2713 (2018)

[26] Luan, V.T., Michels, D.L.: Explicit Exponential Rosenbrock Methods and their Application in Visual Computing. arXiv preprint arXiv:1805.08337, 1–18 (2018) https://arxiv.org/abs/1805.08337

[27] Todorov, E., Erez, T., Tassa, Y.: MuJoCo: A physics engine for model-based control. In: Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference On (2012)

[28] Van Loan, C.F.: Computing Integrals Involving the Matrix Exponential. IEEE Transactions on Automatic Control **23**(3), 395–404 (1978)

[29] Carbonell, F., Jímenez, J.C., Pedroso, L.M.: Computing multiple integrals involving matrix exponentials. Journal of Computational and Applied Mathematics **213**(1), 300–305 (2008)

[30] Al-Mohy, A.H., Higham, N.J.: A New Scaling and Squaring Algorithm for the Matrix Exponential. SIAM Journal on Matrix Analysis and Applications **31**(3) (2010)

[31] Grimminger, F., Meduri, A., Khadiv, M., Viereck, J., Wuthrich, M., Naveau, M., Berenz, V., Heim, S., Widmaier, F., Flayols, T., Fiene, J., Badri-Sprowitz, A., Righetti, L.: An Open Torque-Controlled Modular Robot Architecture for Legged Locomotion Research. IEEE Robotics and Automation Letters **5**(2), 3650–3657 (2020) https://arxiv.org/abs/1910.00093

[32] Project Romeo http://projetromeo.com

[33] Carpentier, J., Saurel, G., Buondonno, G., Mirabel, J., Lamiraux, F., Stasse, O., Mansard, N.: The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. Proceedings of the 2019 IEEE/SICE International Symposium on System Integration, SII 2019, 614–619 (2019)

[34] Featherstone, R.: Exploiting sparsity in operational-space dynamics. The

Exponential Integration for Efficient Multi-Body Simulation with Stiff Viscoelastic Contacts

International Journal of Robotics Research **29**(10), 1–21 (2010)