PLOS ONE

RESEARCH ARTICLE

Algorithmic differentiation improves the computational efficiency of OpenSim-based trajectory optimization of human movement

Antoine Falisse[1☯*], **Gil Serrancolí**[2☯], **Christopher L. Dembia**[3], **Joris Gillis**[4,5], **Friedl De Groote**[1]

1 Department of Movement Sciences, KU Leuven, Leuven, Belgium, **2** Department of Mechanical Engineering, Universitat Politècnica de Catalunya, Barcelona, Catalunya, Spain, **3** Department of Mechanical Engineering, Stanford University, Stanford, California, United States of America, **4** Department of Mechanical Engineering, KU Leuven, Leuven, Belgium, **5** DMMS Lab, Flanders Make, Leuven, Belgium

☯ These authors contributed equally to this work.
* antoine.falisse@kuleuven.be

Citation: Falisse A, Serrancolí G, Dembia CL, Gillis J, De Groote F (2019) Algorithmic differentiation improves the computational efficiency of OpenSim-based trajectory optimization of human movement. PLoS ONE 14(10): e0217730. https://doi.org/10.1371/journal.pone.0217730

Editor: Manoj Srinivasan, The Ohio State University, UNITED STATES

Received: May 15, 2019

Accepted: September 19, 2019

Published: October 17, 2019

Data Availability Statement: All data, code, and materials used in this study are available at https://simtk.org/projects/algodiff.

Funding: This work was supported by the Research Foundation Flanders (https://www.fwo.be/) under Ph.D. grant 1S35416N and travel grant V441717N to AF, and research project grant G079216N to FDG, and by the NIH research infrastructure (https://orip.nih.gov/) under grant P2C HD065690 to the National Center for Simulation in Rehabilitation Research (NCSRR)

Abstract

Algorithmic differentiation (AD) is an alternative to finite differences (FD) for evaluating function derivatives. The primary aim of this study was to demonstrate the computational benefits of using AD instead of FD in OpenSim-based trajectory optimization of human movement. The secondary aim was to evaluate computational choices including different AD tools, different linear solvers, and the use of first- or second-order derivatives. First, we enabled the use of AD in OpenSim through a custom source code transformation tool and through the operator overloading tool ADOL-C. Second, we developed an interface between OpenSim and CasADi to solve trajectory optimization problems. Third, we evaluated computational choices through simulations of perturbed balance, two-dimensional predictive simulations of walking, and three-dimensional tracking simulations of walking. We performed all simulations using direct collocation and implicit differential equations. Using AD through our custom tool was between 1.8 ± 0.1 and 17.8 ± 4.9 times faster than using FD, and between 3.6 ± 0.3 and 12.3 ± 1.3 times faster than using AD through ADOL-C. The linear solver efficiency was problem-dependent and no solver was consistently more efficient. Using second-order derivatives was more efficient for balance simulations but less efficient for walking simulations. The walking simulations were physiologically realistic. These results highlight how the use of AD drastically decreases computational time of trajectory optimization problems as compared to more common FD. Overall, combining AD with direct collocation and implicit differential equations decreases the computational burden of trajectory optimization of human movement, which will facilitate their use for biomechanical applications requiring the use of detailed models of the musculoskeletal system.

that funded AF and GS as visiting scholars. CLD received a Stanford Bio-X Graduate Fellowship (https://biox.stanford.edu/). JG has benefited from KU Leuven-BOF PFV/10/002 Centre of Excellence: Optimization in Engineering (OPTEC, https://set.kuleuven.be/optec) and from Flanders Make (https://www.flandersmake.be) ICON (DriveTrainCodesign). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

Introduction

Combining musculoskeletal modeling and dynamic simulation is a powerful approach to study the mechanisms underlying human movement. In the last decades, researchers have primarily used inverse dynamic simulations to identify biomechanical variables (e.g., muscle forces and joint loads) underlying observed movements. Yet dynamic simulations can also be applied to generate novel movements. Such predictive simulations have the potential to reveal cause-effect relationships that cannot be explored based on inverse dynamic simulations that require movement kinematics as input. Novel movements can be generated by solving trajectory optimization problems. Generally, trajectory optimization consists of identifying a trajectory that optimizes an objective function subject to a set of dynamic and path constraints [1]. In the biomechanical field, researchers have used trajectory optimization for solving two main types of problems. In tracking problems, the objective function is the difference between a variable's measured and simulated value [2–4], whereas in predictive problems, the objective function represents a movement related performance criterion (e.g., minimizing muscle fatigue) [5–8]. However, the nonlinearity and stiffness of the dynamic equations characterizing the musculoskeletal system cause the underlying optimal control problems to be challenging to solve and computationally expensive [5,7,8]. For example, small changes in controls can cause large changes in kinematics and hence a foot to penetrate into the ground, drastically increasing ground reaction forces. These challenges have caused the biomechanics community to primarily perform studies based on inverse dynamic analyses of observed movements rather than trajectory optimization of novel movements.

Over the last decade, the increase in computer performance and the use of efficient numerical methods have equipped researchers with more efficient tools for solving trajectory optimization problems. In particular, direct collocation methods [4,6,8–11] and implicit formulations of the musculoskeletal dynamics [10,12] have become popular. Direct collocation reduces the sensitivity of the objective function to the optimization variables, compared to other methods such as direct shooting [5], by reducing the time horizon of the integration. Direct collocation converts optimal control problems into large sparse nonlinear programming problems (NLPs) that readily available NLP solvers (e.g., IPOPT [13]) can solve efficiently. Implicit formulations of the musculoskeletal dynamics improve the numerical conditioning of the NLP over explicit formulations by, for example, removing the need to divide by small muscle activations [10] or invert a mass matrix that is near-singular due to body segments with a large range of masses and moments of inertia [12]. In implicit formulations, additional controls are typically introduced for the time derivative of the states, which allows imposing the nonlinear dynamic equations as algebraic constraints in their implicit rather than explicit form (i.e., $\dot{y} = u$, $0 = f_i(y,u)$ instead of $\dot{y} = f_e(y)$).

Algorithmic differentiation (AD) is another numerical tool that can improve the efficiency of trajectory optimization [14,15]. AD is a technique for evaluating derivatives of functions represented by computer programs. It is, therefore, an alternative to finite differences (FD) for evaluating the derivative matrices required by the NLP solver, namely the objective function gradient, the constraint Jacobian, and the Hessian of the Lagrangian (henceforth referred to as simply Hessian). These evaluations are obtained free of truncation errors, in contrast with FD, and for a computational cost of the same order of magnitude as the cost of evaluating the original function. AD relies on the observation that any function can be broken down into a sequence of elementary operations, forming an expression graph (example in Fig 1). AD then relies on the chain rule of calculus that describes how to calculate the derivative of a composition of functions [15]. By traversing a function's expression graph while applying the chain rule, AD allows computing the function derivatives. Note that, like FD, AD can exploit the

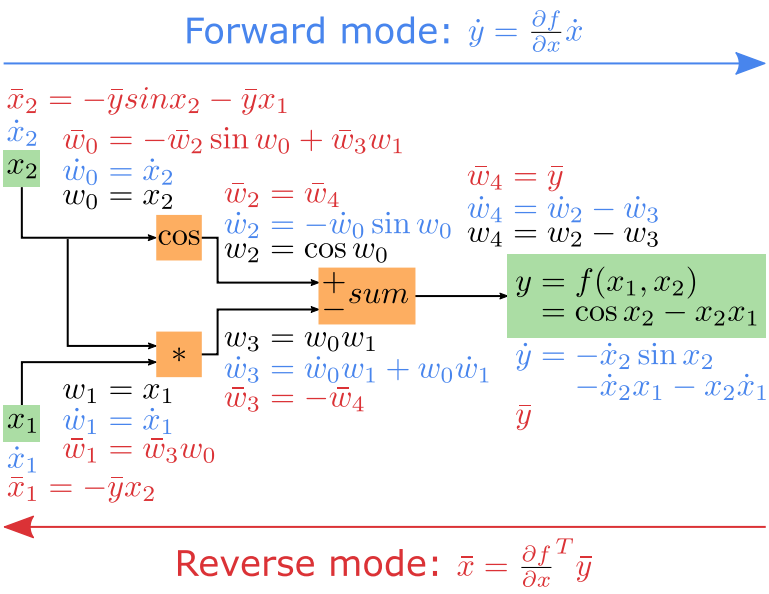

Fig 1. Example of AD forward and reverse modes. A function $y = f(x_1,x_2) = cos\ x_2 - x_2x_1$ is broken down into a sequence of elementary operations, forming an expression graph. In the forward mode, the forward seeds $\dot{x}_1$ and $\dot{x}_2$ are propagated from the inputs to the output, and the Jacobian $J = \partial f/\partial \boldsymbol{x}$ relates $\dot{x}_1$ and $\dot{x}_2$ to the forward sensitivity $\dot{y}$. In the reverse mode, the reverse seed $\bar{y}$ is propagated from the output to the inputs, and the transposed Jacobian J^T relates $\bar{y}$ to the reverse sensitivities $\bar{x}_1$ and $\bar{x}_2$.

https://doi.org/10.1371/journal.pone.0217730.g001

sparsity of the aforementioned derivative matrices resulting, for example, from applying direct collocation [16].

AD allows traversing the expression graph in two directions or modes: from the inputs to the outputs in its forward mode and from the outputs to the inputs in its reverse mode. This permits the evaluation of two types of directional derivatives: Jacobian-times-vector product and Jacobian-transposed-times-vector product in the forward and reverse mode, respectively. The computational efficiency of the AD mode depends on the problem dimensions. Consider the function $G : \mathbb{R}^n \rightarrow \mathbb{R}^m : y = G(x)$ describing the m NLP constraints y as a function of the n optimization variables x. The constraint Jacobian $J = \partial y/\partial x$ is a matrix with size m x n. In the forward mode, J relates forward seeds $\dot{x}$ to forward sensitivities $\dot{y}$: $\dot{y} = J\dot{x}$ (example in Fig 1). In the reverse mode, J^T relates reverse seeds $\bar{y}$ to reverse sensitivities $\bar{x}$: $\bar{x} = J^T\bar{y}$ (example in Fig 1). In the forward mode, the cost of evaluating J is proportional to n times the cost of evaluating G. In the reverse mode, the cost of evaluating J^T is proportional to m times the cost of evaluating G. If there are many more inputs n than outputs m, the reverse mode may drastically decrease the number of function evaluations required to evaluate J and highly reduce the computational time (CPU time) as compared to the forward mode [15,17].

Two main approaches exist for adding AD to existing software, namely operator overloading and source code transformation. Source code transformation is inherently faster than operator overloading but may not be readily available for all features of a programming language. In the operator overloading approach, AD's algorithms are applied after the evaluation of the original function using concrete numerical inputs. This is typically performed by introducing a new numerical type that stores information about partial derivatives as calculations proceed (e.g., through operator overloading in C++) [15,17]. Examples of AD tools using operator overloading in C++ are ADOL-C [18] and CppAD [19]. In the source code transformation approach, the AD tool analyzes a given function's source code and outputs a new function

that computes the forward or reverse mode of that function. Examples of AD tools using source code transformation are ADiGator for MATLAB [20] and CasADi that is available for C++, Python, and MATLAB [21]. CasADi is a modern actively developed tool for nonlinear optimization and AD that has many additional features (e.g., code generation) and interfaces with NLP solvers designed to handle large and sparse NLPs (e.g., IPOPT). CasADi provides a high-level, symbolic, way to construct an expression graph, on which source code transformation is applied. The resultant expression graph can be code-generated to achieve the computational efficiency of pure source code transformation.

AD has a long history [14] but has rarely been applied in biomechanics, likely because AD is relatively unknown in the field and is not integrated as part of widely used biomechanical software packages. In previous work, we solved muscle redundancy problems while exploiting AD [10,22]. For this purpose, we used GPOPS-II [23], a MATLAB software for solving optimal control problems with direct collocation, in combination with ADiGator. However, these problems were limited to models implemented in MATLAB, enabling the use of ADiGator. Generating simulations of human movement requires expanding these problems to account for the multi-body dynamics. OpenSim [24,25] and its dynamics engine Simbody [26] are widely used open-source software packages for musculoskeletal modeling and biomechanical dynamic simulation. These packages provide multi-body dynamics models and have been used for trajectory optimization of human gait [3,4,8,11]. Yet they currently do not leverage tools for AD. Moreover, they are written in C++, which would prevent the use of ADiGator.

AD is increasingly used for trajectory optimization in related fields such as rigid body dynamics for robotic applications and several software packages leverage AD tools [27]. RobCoGen is a modeling tool for rigid body dynamics that supports AD through source code transformation. Giftthaler *et al*. showed that trajectory optimization of gait for a quadrupedal robot modeled with RobCoGen was five times faster with AD than with FD [27]. Other packages for robotic applications with modules supporting AD include Drake [28], Robotran [29], MBSlib [30], and Pinocchio [31]. Drake is a collection of tools that relies on Eigen [32] for linear algebra. Eigen has a module supporting AD's forward mode using operator overloading. Robotran is a symbolic software to model multibody systems that can be interfaced with CasADi to solve optimal control problems. MBSlib is a multibody system library supporting AD through ADOL-C. Finally, Pinocchio is a software platform implementing algorithms for rigid body dynamics that can be interfaced with ADOL-C, CppAD, and CasADi. Note that AD is not exclusively used for trajectory optimization and is also applied in other related fields including deep learning with libraries such as TensorFlow [33] and Theano [34], and applications for robotic gait optimization (e.g., [35]).

The contribution of this study is threefold. First, we enabled the use of AD in OpenSim and Simbody (henceforth referred to as OpenSim). We compared two approaches: we incorporated the operator overloading AD tool ADOL-C and we developed our own AD tool Recorder that uses operator overloading to construct an expression graph on which source code transformation is applied using CasADi. Second, we interfaced OpenSim with CasADi, enabling trajectory optimization using OpenSim's multi-body dynamics models while benefitting from CasADi's efficient interface with NLP solvers. Third, we evaluated the efficiency of different computational choices based on trajectory optimization problems of varying complexity solved with IPOPT. We compared three different derivative scenarios: AD with ADOL-C, AD with Recorder, and FD. In addition, we compared different linear solvers and different Hessian calculation schemes within IPOPT, to aid users in choosing the most efficient solver settings. Primal-dual interior point methods such as IPOPT rely on linear solvers to solve the primal-dual system, which involves the Hessian, when computing the Newton step direction during the optimization [36]. The Hessian can be exact (i.e., based on second-order derivative

information) or approximated with a limited-memory quasi-Newton method (L-BFGS) that only requires first-order derivative information. We found that using AD through Recorder was more efficient than using FD or AD through ADOL-C, whereas the efficiency of the linear solver and Hessian calculation scheme was problem-dependent.

Materials and methods

Tools to enable the use of AD in OpenSim

We first incorporated the operator overloading AD tool ADOL-C in OpenSim. ADOL-C relies on the concept of active variables, which are variables that may be considered as differentiable quantities at some time during the execution of a computer program [18]. To distinguish these variables and store information about their partial derivatives, ADOL-C introduced the augmented scalar type *adouble* whose real part is of standard type *double*. All active variables should be of type *adouble*. To differentiate OpenSim functions using ADOL-C, we modified OpenSim's source code by replacing the type of potential active variables to *adouble* (example for SimTK::square() in Fig 2). We maintained a layer of indirection so that OpenSim could be compiled to use either *double* or *adouble* as the scalar type. We excluded parts of the code, such as numerical optimizers, that were not relevant to this study.

The limited computational benefits of using AD through ADOL-C led us to seek alternative AD strategies (see discussion for more detail). We developed our own tool, Recorder, which combines the versatility of operator overloading and the speed of source code transformation. Recorder is a C++ scalar type for which all operators are overloaded to generate an expression graph. When evaluating an OpenSim function numerically at a nominal point, Recorder generates the function's expression graph as MATLAB source code in a format that CasADi's AD algorithms can transform into C-code (see S1 Appendix for source code from the example of Fig 1). Note that this workflow is currently only practical when the branches (*if-tests*)

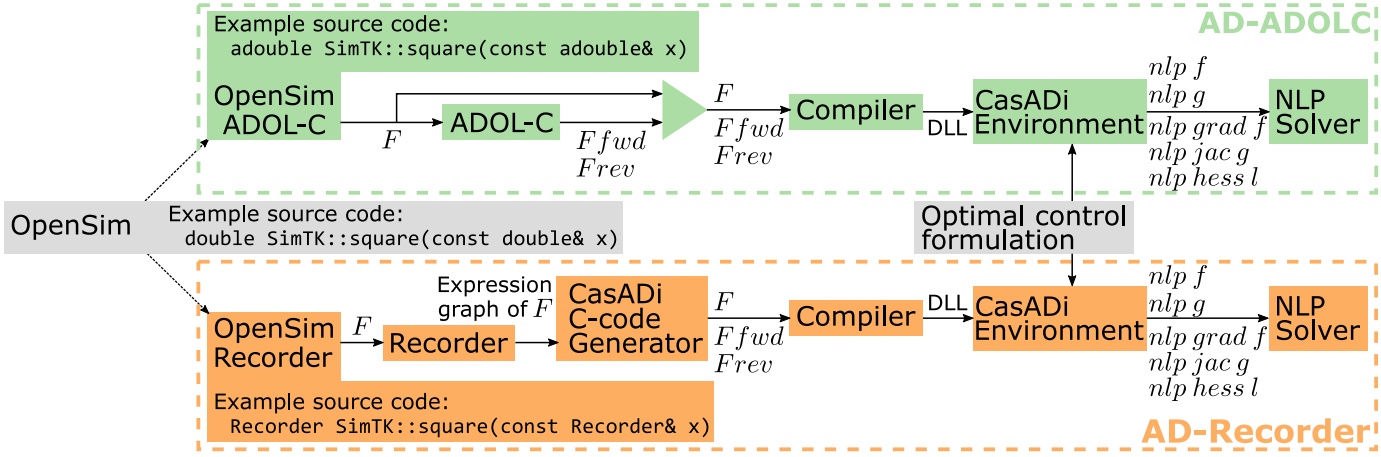

Fig 2. Flowchart depicting the optimal control framework. We developed two approaches (AD-ADOLC and AD-Recorder) to make an OpenSim function *F* and its forward (*F fwd*) and reverse (*F rev*) directional derivatives available within the CasADi environment for use by the NLP solver during the optimization. In the AD-ADOLC approach (top), ADOL-C's algorithms are used in a C++ code to provide *F fwd* and *F rev*. In the AD-Recorder approach (bottom), Recorder provides the expression graph of *F* as MATLAB source code from which CasADi's C-code generator generates C-code containing *F*, *F fwd*, and *F rev*. The AD-Recorder approach combines operator overloading, when generating the expression graph, and source code transformation, when processing the expression graph to generate C-code for *F*, *F fwd*, and *F rev*. In both approaches, the code comprising *F*, *F fwd*, and *F rev* is compiled as a Dynamic-link Library (DLL), which is imported as an external function within the CasADi environment. In our application, *F* represents the multi-body dynamics and is called when formulating the optimal control problem. The latter is then composed into a differentiable optimal control transcription using CasADi. During the optimization, CasADi provides the NLP solver with evaluations of the NLP objective function (*nlp f*), constraints (*nlp g*), objective function gradient (*nlp grad f*), constraint Jacobian (*nlp jac g*), and Hessian of the Lagrangian (*nlp hess l*). CasADi efficiently queries *F fwd* and *F rev* to construct the full derivative matrices.

https://doi.org/10.1371/journal.pone.0217730.g002

encountered at the nominal point remain valid for all evaluations encountered during the optimization.

To use Recorder with OpenSim, we relied on the code we had modified for incorporating ADOL-C but replaced *adouble* with the *Recorder* scalar type (example for SimTK::square() in Fig 2). This change required minimal effort but enabled Recorder to identify all differentiable variables when constructing the expression graphs.

Interface between OpenSim and CasADi

We enabled the use of OpenSim functions within the CasADi environment by compiling the functions and their derivatives as Dynamic-link Libraries that are then imported as external functions for use by CasADi (Fig 2). The function derivatives can be computed through ADOL-C (AD-ADOLC in Fig 2) or through Recorder (AD-Recorder in Fig 2).

Trajectory optimization problems to evaluate computational choices

We designed three example trajectory optimization problems to evaluate different computational choices (see Tables 1–3 for detailed formulations). The general formulation of the optimal control problems consists of computing the controls $\boldsymbol{u}(t)$, states $\boldsymbol{x}(t)$, and time-independent parameters $\boldsymbol{p}$ minimizing an objective functional:

$$J = \int_{t_i}^{t_f} L(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}) dt, \quad (1)$$

where t_i and t_f are initial and final times, and t is time [37]. This objective functional is subject

Table 1. Formulation of example 1.

Pendulum simulations			
Number of optimization variables	2 degree of freedom pendulum: 504 3 degree of freedom pendulum: 756 4 degree of freedom pendulum: 1008 5 degree of freedom pendulum: 1260 6 degree of freedom pendulum: 1512 7 degree of freedom pendulum: 3514 8 degree of freedom pendulum: 4016 9 degree of freedom pendulum: 4518 10 degree of freedom pendulum: 10020	**Number of equality constraints**	2 degree of freedom pendulum: 458 3 degree of freedom pendulum: 687 4 degree of freedom: pendulum 916 5 degree of freedom pendulum: 1145 6 degree of freedom pendulum: 1374 7 degree of freedom pendulum: 3178 8 degree of freedom pendulum: 3632 9 degree of freedom pendulum: 4086 10 degree of freedom pendulum: 9040
States $\boldsymbol{x}(t)$	Joint positions q and velocities v	**Controls** $\boldsymbol{u}(t)$	Derivatives of v (accelerations): u_{dv} Joint torques u_T
Bounds	$-3\pi = q_{lb} \le q \le q_{ub} = 3\pi$ $-20 = v_{lb} \le v \le v_{ub} = 20$ $-500 = u_{dv,lb} \le u_{dv} \le u_{dv,ub} = 500$ $-1000 = u_{T,lb} \le u_T \le u_{T,ub} = 1000$	**Scaling**	$s_q = 3;\ \tilde{q} = q/s_q;\ \tilde{q}_{lb} = q_{lb}/s_q;\ \tilde{q}_{ub} = q_{ub}/s_q$ $s_t = 0.2;\ \tilde{v} = v/(s_q/s_t);\ \tilde{v}_{lb} = v_{lb}/(s_q/s_t);\ \tilde{v}_{ub} = v_{ub}/(s_q/s_t)$ $\tilde{u}_{dv} = u_{dv}/(s_q/s_t^2);\ \tilde{u}_{dv,lb} = u_{dv,lb}/(s_q/s_t^2);\ u_{dv,ub} = u_{dv,ub}/(s_q/s_t^2)$ $s_T = 500;\ \tilde{u}_T = u_T/s_T;\ \tilde{u}_{T,lb} = u_{T,lb}/s_T;\ \tilde{u}_{T,ub} = u_{T,ub}/s_T$
Objective function	$L = \|\tilde{u}_T\|_2^2 + L_p$ $L_p = 0.1\|\tilde{u}_{dv}\|_2^2$	**Dynamic constraints**	$(dqdt)/s_q = v/s_q$ $(dv/dt)/(s_q/s_t) = u_{dv}/(s_q/s_t)$
Path constraints	$T = f_s(q, v, u_{dv})$ $T/s_T = \tilde{u}_T$ $\tilde{q}(0) = \tilde{q}(1) = \tilde{v}(0) = \tilde{v}(1) = 0$		

Controls: we introduced accelerations (time derivative of velocities) as controls (implicit formulations) in addition to joint torques. **Bounds**: *lb* and *ub* are for lower and upper bounds, respectively. **Scaling**: we used time scaling for the joint states and controls. **Objective function**: to avoid singular arcs, situations for which controls are not uniquely defined by the optimality conditions [37], we appended a penalty function L_p with the remaining controls to the objective function L. **Dynamic constraints** are scaled using the same scale factors as used for the states [37]. We used implicit formulations. **Path constraints**: $f_s(\cdot)$ computes net joint torques T according to the skeleton dynamics.

https://doi.org/10.1371/journal.pone.0217730.t001

Table 2. Formulation of example 2.

2D predictive simulations of walking			
Number of optimization variables	13807	**Number of constraints**	12857 equality constraints, 1800 inequality constraints
States $\boldsymbol{x}(t)$	Muscle activations a and tendon forces F_t Joint positions q and velocities v Trunk activations a_{trunk}	**Controls** $\boldsymbol{u}(t)$	Derivatives of a: u_{da} and F_t: u_{dF_t} Derivatives of v (accelerations): u_{dv} Trunk excitations e_{trunk}
Parameters $\boldsymbol{p}$	Half gait cycle duration t_f	**Scaling**	$s_q = \max(\text{abs}(q_{lb}), \text{abs}(q_{ub}));\ \tilde{q} = q/s_q;\ \tilde{q}_{lb} = q_{lb}/s_q;\ \tilde{q}_{ub} = q_{ub}/s_q$ $s_v = \max(\text{abs}(v_{lb}), \text{abs}(v_{ub}));\ \tilde{v} = v/s_v;\ \tilde{v}_{lb} = v_{lb}/s_v;\ \tilde{v}_{ub} = v_{ub}/s_v$ $s_{u_{dv}} = \max(\text{abs}(u_{dv,lb}), \text{abs}(u_{dv,ub}));\ \tilde{u}_{dv} = u_{dv}/s_{u_{dv}}$ $\tilde{u}_{dv,lb} = u_{dv,lb}/s_{u_{dv}};\ \tilde{u}_{dv,ub} = u_{dv,ub}/s_{u_{dv}}$ $s_{da} = s_{dF_t} = 100;\ s_{\text{trunk}} = 150$ $s_{F_t} = F_{t,ub};\ \tilde{F}_t = F_t/s_{F_t};\ \tilde{F}_{t,ub} = F_{t,ub}/s_{F_t}$
Bounds	$0 \le a \le 1;\ 0 \le F_t \le F_{t,ub} = 5;\ 0.1 \le t_f \le 1$ $-\tau_d/100 \le u_{da} \le \tau_a/100;\ \tau_d = 60\text{ms};\ \tau_a = 15\text{ms}$ $-1 \le u_{dF_t}, a_{\text{trunk}}, e_{\text{trunk}} \le 1$ $q_{lb,\text{man}} = q_{lb} \le q \le q_{ub} = q_{ub,\text{man}}$ $v_{lb,\text{man}} = v_{lb} \le v \le v_{ub} = v_{ub,\text{man}}$ $u_{dv,lb,\text{man}} = u_{dv,lb} \le u_{dv} \le u_{dv,ub} = u_{dv,ub,\text{man}}$		
Objective function	$L = (w_1\|a\|_3^3 + w_2\|e_{\text{trunk}}\|_2^2 + w_3\|\tilde{u}_{dv}\|_2^2 + L_p)/d$ $L_p = 0.001(\|u_{da}\|_2^2 + \|u_{dF_t}\|_2^2)$	**Dynamic constraints**	$da/dt = s_{da}u_{da}$ $(dF_t/dt)/s_{F_t} = (s_{dF_t}u_{dF_t})/s_{F_t}$ $(dq/dt)/s_q = v/s_q$ $(dv/dt)/s_v = u_{dv}/s_v$ $da_{\text{trunk}}/dt = (e_{\text{trunk}} - a_{\text{trunk}})/\tau;\ \tau = 35\text{ms}$
Path constraints	$0 \le s_{da}u_{da} + a/\tau_d$ $s_{da}u_{da} + a/\tau_a \le 1/\tau_a$ $f_c(a, F_t, u_{dF_t}) = 0$ $T = f_s(q, v, u_{dv})$ $T_{\text{pelvis}} = 0$	$T_{\text{ll}} = \sum_{m=1}^{M} MA_m F_{t,m}$ $T_{\text{trunk}}/s_{\text{trunk}} = a_{\text{trunk}}$ $\bar{x}(t_f) = \bar{x}(0)$ $(q_{\text{pelvis,for}}(t_f) - q_{\text{pelvis,for}}(0))/t_f = 1.33$	

Controls are introduced for the time derivative of the states (implicit formulations) in addition to trunk excitations. **Bounds** are manually (man) set for the joint states and controls; *lb* and *ub* are for lower and upper bounds, respectively. **Scaling**: joint states and controls, and tendon forces are scaled such that the lower and upper bounds are between -1 and 1. **Objective function** L is normalized by distance traveled d. To avoid singular arcs [37], a penalty function L_p (with low weight) with the remaining controls is appended to L. **Dynamic constraints** are scaled using the scale factors used for the states [37]. **Path constraints**: $f_s(\cdot)$ computes net joint torques T according to the skeleton dynamics, $f_c(\cdot)$ describes the Hill-type muscle contraction dynamics [10], MA_m is moment arm of muscle m, $\bar{x}(\cdot)$ contains all states except the pelvis forward position $q_{\text{pelvis,for}}$ (symmetry), and 1.33 m s^{-1} is the prescribed gait speed.

https://doi.org/10.1371/journal.pone.0217730.t002

to dynamic constraints:

$$\dot{x}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}), \tag{2}$$

and to algebraic path constraints:

$$g_{\text{min}} \le g(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}) \le g_{\text{max}}, \tag{3}$$

which are equality constraints if $g_{\text{min}} = g_{\text{max}}$. The optimization variables are typically bounded as follows:

$$x_{\text{min}} \le \boldsymbol{x}(t) \le x_{\text{max}}, \tag{4}$$

$$u_{\text{min}} \le \boldsymbol{u}(t) \le u_{\text{max}}, \tag{5}$$

$$p_{\text{min}} \le \boldsymbol{p} \le p_{\text{max}}. \tag{6}$$

In the first example, we perturbed the balance of nine inverted pendulums, with between two and 10 degrees of freedom, by applying a backward translation to their base of support. The optimal control problem identified the joint torques necessary to restore the pendulums'

Table 3. Formulation of example 3.

3D tracking simulations of walking			
Number of optimization variables	61318	**Number of constraints**	56050 equality constraints, 9200 inequality constraints
States $\boldsymbol{x}(t)$	Muscle activations a and tendon forces F_t Joint positions q and velocities v Arm activations a_{arms}	**Controls** $\boldsymbol{u}(t)$	Derivatives of a: u_{da} and F_t: u_{dF_t} Derivatives of v (accelerations): u_{dv} Arm excitations e_{arms}
Parameters $\boldsymbol{p}$	Contact sphere transversal plane locations p_{cl} Contact sphere radii p_{cr}	**Scaling**	$s_q = \max(\text{abs}(q_{lb}), \text{abs}(q_{ub}));\ \tilde{q} = q/s_q;\ \tilde{q}_{lb} = q_{lb}/s_q;$ $\tilde{q}_{ub} = q_{ub}/s_q$ $s_v = \max(\text{abs}(v_{lb}), \text{abs}(v_{ub}));\ \tilde{v} = v/s_v;\ \tilde{v}_{lb} = v_{lb}/s_v;$ $\tilde{v}_{ub} = v_{ub}/s_v$ $s_{u_{dv}} = \max(\text{abs}(u_{dv,lb}), \text{abs}(u_{dv,ub}));\ \tilde{u}_{dv} = u_{dv}/s_{u_{dv}}$ $\tilde{u}_{dv,lb} = u_{dv,lb}/s_{u_{dv}};\ \tilde{u}_{dv,ub} = u_{dv,ub}/s_{u_{dv}}$ $s_{da} = s_{dF_t} = 100;\ s_T = s_{\text{arms}} = 150$ $s_{F_t} = F_{t,ub};\ \tilde{F}_t = F_t/s_{F_t};\ \tilde{F}_{t,ub} = F_{t,ub}/s_{F_t}$ $\tilde{p}_{cl,v} = 1/(p_{cl,ub} - p_{cl,lb});\ \tilde{p}_{cl,r} = 0.5 - p_{cl,ub}/(p_{cl,ub} - p_{cl,lb})$ $\tilde{p}_{cr,v} = 1/(p_{cr,ub} - p_{cr,lb});\ \tilde{p}_{cr,r} = 0.5 - p_{cr,ub}/(p_{cr,ub} - p_{cr,lb})$ $\tilde{p}_{cl,lb} = \tilde{p}_{cr,lb} = -0.5;\ \tilde{p}_{cl,ub} = \tilde{p}_{cr,ub} = 0.5$
Bounds	$0 \le a \le 1;\ 0 \le F_t \le F_{t,ub} = 5$ $-\tau_d/100 \le u_{da} \le \tau_a/100;\ \tau_d = 60\text{ms};\ \tau_a = 15\text{ms}$ $-1 \le u_{dF_t}, a_{\text{arms}}, e_{\text{arms}} \le 1$ $\widehat{q}_{\min} - \widehat{q}_r = q_{lb} \le q \le q_{ub} = \widehat{q}_{\max} + \widehat{q}_r$ $\widehat{v}_{\min} - \widehat{v}_r = v_{lb} \le v \le v_{ub} = \widehat{v}_{\max} + \widehat{v}_r$ $\widehat{u}_{dv,\min} - \widehat{u}_{dv,r} = u_{dv,lb} \le u_{dv} \le u_{dv,ub} = \widehat{u}_{dv,\max} + \widehat{u}_{dv,r}$ $\widehat{p}_{ci} - 0.025 = p_{cl,lb} \le p_{cl} \le p_{cl,ub} = \widehat{p}_{ci} + 0.025$ $\widehat{p}_{cr} - 0.5\widehat{p}_{cr} = p_{cr,lb} \le p_{cr} \le p_{cr,ub} = \widehat{p}_{cr} + 0.5\widehat{p}_{cr}$		
Objective function	$L = w_1\|\|a\|\|_2^2 + w_2\|\|q - \widehat{q}\|\|_2^2 + w_3\|\|GRF - \widehat{GRF}\|\|_2^2 + w_4\|\|GRT - \widehat{GRT}\|\|_2^2 + w_5\|\|T_{\text{ll,trunk,arms}} - \widehat{T}_{\text{ll,trunk,arms}}\|\|_2^2 + L_p$ $L_p = 0.001\ (\|\|u_{da}\|\|_2^2 + \|\|u_{dF_t}\|\|_2^2 + \|\|u_{dv}\|\|_2^2)$	**Dynamic constraints**	$da/dt = s_{da}u_{da}$ $(dF_t/dt)/s_{F_t} = (s_{dF_t}u_{dF_t})/s_{F_t}$ $(dq/dt)/s_q = v/s_q$ $(dv/dt)/s_v = u_{dv}/s_v$ $da_{\text{arms}}/dt = (e_{\text{arms}} - a_{\text{arms}})/\tau;\ \tau = 35\text{ms}$
Path constraints	$0 \le s_{da}u_{da} + a/\tau_d$ $s_{da}u_{da} + a/\tau_a \le 1/\tau_a$ $f_c(a, F_t, u_{dF_t}) = 0$ $T = f_s(q, v, u_{dv})$	$T_{\text{pelvis}}/s_T = \widehat{T}_{\text{pelvis}}/s_T$ $T_{\text{ll,trunk}} = \sum_{m=1}^{M} MA_m F_{t,m} + T_p$ $T_{\text{arms}}/s_{\text{arms}} = a_{\text{arms}}$	

Controls are introduced for the state derivatives in addition to arm excitations. **Bounds** of joint states and controls are based on measured data ($\widehat{q}$, $\widehat{u}$, $\widehat{u}_{dv}$); min and max are for minimum and maximum values, respectively; *r* is range of motion; *lb* and *ub* are for lower and upper bounds, respectively; bounds of contact parameters (p_{cl}, p_{cr}) are based on [4]. **Scaling**: joint states and controls, and tendon forces are scaled such that the lower and upper bounds are between -1 and 1; contact parameters are scaled such that their lower and upper bounds are -0.5 and 0.5, respectively.

Objective function L tracks measured joint positions ($\widehat{q}$), ground reaction forces ($\widehat{GRF}$) and torques ($\widehat{GRT}$), and joint torques of the lower limbs, trunk, and arms ($\widehat{T}_{\text{ll,trunk,arms}}$). A penalty function L_p is appended to L. **Dynamic constraints** are scaled using scale factors used for the states [37]. **Path constraints**: $f_s(\cdot)$ computes net joint torques T according to the skeleton dynamics, $f_c(\cdot)$ describes the Hill-type muscle contraction dynamics [10], MA_m is moment arm of muscle m, and T_p are passive torques [5].

https://doi.org/10.1371/journal.pone.0217730.t003

Table 4. Numerical tools.

NLP solver	Linear solvers		AD approaches
IPOPT	Mumps		Operator overloading (ADOL-C)
	HSL collection	ma27 ma57 ma77 ma86 ma97	Source code transformation (Recorder)

https://doi.org/10.1371/journal.pone.0217730.t004

upright posture within one second while minimizing the actuator effort (i.e., squared joint torques) and satisfying the pendulum dynamics.

In the second example, we performed predictive simulations of walking with a two-dimensional (2D) musculoskeletal model (10 degrees of freedom, 18 muscles actuating the lower limbs, one ideal torque actuator at the trunk, and two contact spheres per foot [24]). We identified muscle excitations and half walking cycle duration that minimized a weighted sum of muscle fatigue (i.e., muscle activations at the third power [6]) and joint accelerations subject to constraints describing the musculoskeletal dynamics, imposing left-right symmetry, and prescribing gait speed (i.e., distance travelled by the pelvis divided by gait cycle duration). Imposing left-right symmetry allowed us to only optimize for half a gait cycle.

In the third example, we performed tracking simulations of walking with a three-dimensional (3D) musculoskeletal model (29 degrees of freedom, 92 muscles actuating the lower limbs and trunk, eight ideal torque actuators at the arms, and six contact spheres per foot [4,24,38]) while calibrating the foot-ground contact model. We identified muscle excitations and contact sphere parameters (locations and radii) that minimized a weighted sum of muscle effort (i.e., squared muscle activations) and the difference between measured and simulated variables (joint angles and torques, and ground reaction forces and torques) while satisfying the musculoskeletal dynamics. Data collection was approved by the Ethics Committee at UZ / KU Leuven (Belgium).

In these examples, we modeled pendulum/skeletal movement with Newtonian rigid body dynamics and, for the walking simulations, compliant Hunt-Crossley foot-ground contact [24,26]. We created a continuous approximation of a contact model from Simbody to provide twice continuously differentiable contact forces, which are required when using second-order gradient-based optimization algorithms [39]. We performed the approximations of conditional *if-tests* using hyperbolic tangent functions. For the muscle-driven walking simulations, we described muscle activation and contraction dynamics using Raasch's model [9,40] and a Hill-type muscle model [10,41], respectively. We defined muscle-tendon lengths, velocities, and moment arms as a function of joint positions and velocities using polynomial functions [42]. We optimized the polynomial coefficients to fit muscle-tendon lengths and moment arms (maximal root mean square deviation: 3 mm; maximal order: ninth) obtained from OpenSim for a wide range of joint positions.

We transcribed each optimal control problem into a NLP using a third order Radau quadrature collocation scheme. We formulated each problem in MATLAB using CasADi and IPOPT. We imposed an NLP relative error tolerance of 1×10^{-6} and used an adaptive barrier parameter update strategy. We selected a number of mesh intervals for each problem such that the results were qualitatively similar when using a mesh twice as fine. We used 10 and three initial guesses for the pendulum and walking simulations, respectively. We ran all simulations on a single core of a standard laptop computer with a 2.9 GHz Intel Core i7 processor.

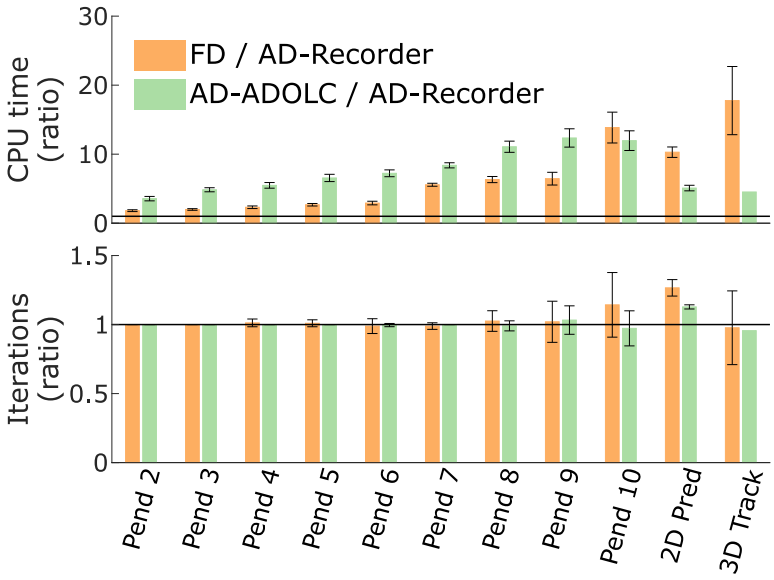

Fig 3. Comparison of computational time (top) and number of iterations (bottom) between FD, AD-ADOLC, and AD-Recorder. The comparisons are expressed as ratios and averaged over results from different initial guesses (error bars represent ± one standard deviation). The horizontal lines indicate 1:1 ratios. Ratios larger than one indicate slower convergence (top) and more iterations (bottom) with FD or AD-ADOLC as compared to AD-Recorder. Pend indicates pendulum simulations with the number being the number of degrees of freedom; Pred and Track indicate predictive and tracking simulations, respectively. The results were obtained using mumps and an approximated Hessian.

https://doi.org/10.1371/journal.pone.0217730.g003

Results analysis

We compared CPU time and number of iterations required to solve the problems using the different computational choices. First, we compared AD, using the Recorder approach, with FD. Second, we compared the AD approaches, namely AD-Recorder and AD-ADOLC. We performed these two comparisons using the linear solver mumps [43], which CasADi provides, and an approximated Hessian. Third, we compared different linear solvers, namely mumps with the collection of solvers from HSL (ma27, ma57, ma77, ma86, and ma97) [44],

Table 5. Comparison of computational time, number of iterations, and computational time per iteration between linear solvers.

Solver * vs mumps	Pendulum simulations			2D predictive simulations			3D tracking simulations		
	CPU time	Iteration Number	CPU time per Iteration	CPU time	Iteration Number	CPU time per Iteration	CPU time	Iteration Number	CPU time per Iteration
*ma27	0.6 ± 0.1	1.0 ± 0.0	0.6 ± 0.1	1.0 ± 0.3	1.4 ± 0.3	0.7 ± 0.1	0.7 ± 0.2	0.9 ± 0.3	0.7 ± 0.0
*ma57	0.6 ± 0.0	1.0 ± 0.0	0.6 ± 0.0	2.4 ± 2.5	2.8 ± 2.8	0.8 ± 0.0	/	/	/
*ma77	0.9 ± 0.1	1.0 ± 0.0	0.9 ± 0.0	1.3 ± 0.0	1.2 ± 0.0	1.1 ± 0.0	0.5 ± 0.2	0.7 ± 0.3	0.7 ± 0.0
*ma86	1.1 ± 0.1	1.0 ± 0.0	1.1 ± 0.1	2.1 ± 0.5	1.1 ± 0.1	1.8 ± 0.3	2.3 ± 0.0	1.9 ± 0.0	1.2 ± 0.0
*ma97	0.7 ± 0.0	1.0 ± 0.0	0.7 ± 0.0	1.2 ± 0.3	1.2 ± 0.1	1.0 ± 0.2	/	/	/

The comparisons are expressed as ratios (mean ± one standard deviation; results obtained with solver from the HSL collection over results obtained with mumps

* indicates ma27, ma57, ma77, ma86, or ma97).

The ratios are averaged over results from different initial guesses. Ratios larger than one indicate faster convergence, fewer iterations, or less time per iteration with mumps. The use of the solvers ma57 and ma97 led to memory issues for the 3D tracking simulations and these cases were therefore excluded from the analysis. The simulations were run using AD-Recorder and an approximated Hessian.

https://doi.org/10.1371/journal.pone.0217730.t005

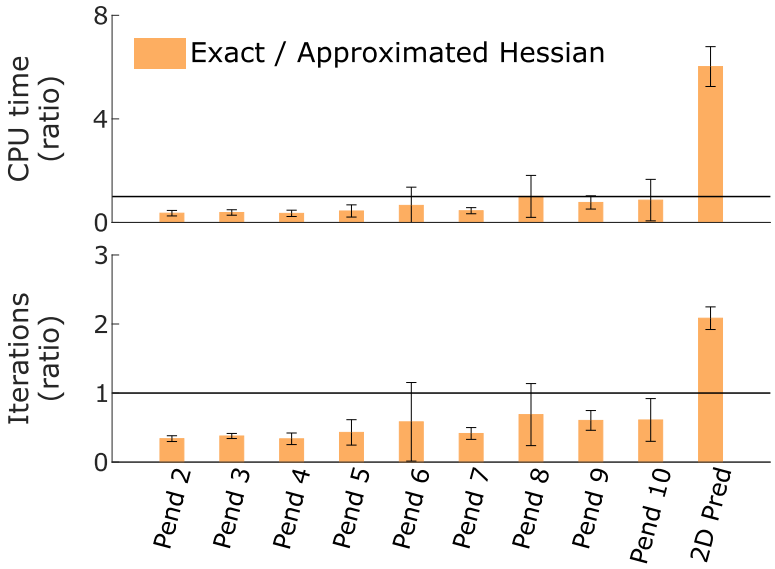

Fig 4. Comparison of computational time (top) and number of iterations (bottom) between exact and approximated Hessian. The comparisons are expressed as ratios and averaged over results from different initial guesses (error bars represent ± one standard deviation). The horizontal lines indicate 1:1 ratios. Ratios larger than one indicate slower convergence (top) and more iterations (bottom) with an exact versus an approximated Hessian. Pend indicates pendulum simulations with the number being the number of degrees of freedom; Pred and Track indicate predictive and tracking simulations, respectively. The results were obtained using all solvers and AD-Recorder.

https://doi.org/10.1371/journal.pone.0217730.g004

while using AD-Recorder and an approximated Hessian. Finally, we compared the use of approximated and exact Hessians. For this last comparison, we used AD-Recorder and tested all linear solvers. In all cases, we ran simulations from different initial guesses and compared results from simulations that started from the same initial guess and converged to similar optimal solutions. Table 4 distinguishes the numerical tools used in our analyses.

Results

Using AD-Recorder was computationally more efficient than using FD or AD-ADOLC (Fig 3). The CPU time decreased when using AD-Recorder as compared to FD (between 1.8 ± 0.1 and 17.8 ± 4.9 times faster with AD-Recorder) and AD-ADOLC (between 3.6 ± 0.3 and 12.3 ± 1.3 times faster with AD-Recorder). CPU time spent in evaluating the objective function gradient accounted for 95 ± 10% (average ± standard deviation) of the difference in CPU time between AD-Recorder and FD. The difference in CPU time spent in evaluating the constraint Jacobian accounted for 89 ± 6% of the difference in CPU time between AD-Recorder and AD-ADOLC. The number of iterations was similar when using AD-Recorder, FD, and AD-ADOLC. For the 2D predictive and 3D tracking simulations, one and two cases, respectively, out of nine (three derivative scenarios and three initial guesses) were excluded from the comparison as they converged to different solutions.

The solvers from the HSL collection were on average more efficient (faster with a similar number of iterations) than mumps for the pendulum simulations, but the efficiency varied for the 2D predictive and 3D tracking simulations (Table 5). The solver ma27 was on average faster than mumps in all cases although ma27 required more iterations for the 2D predictive simulations. The other solvers from the HSL collection were on average slower than mumps for the 2D predictive simulations. For the 3D tracking simulations, the solvers ma77 and ma86 were faster and slower, respectively, than mumps. The solvers ma57 and ma97 failed to solve

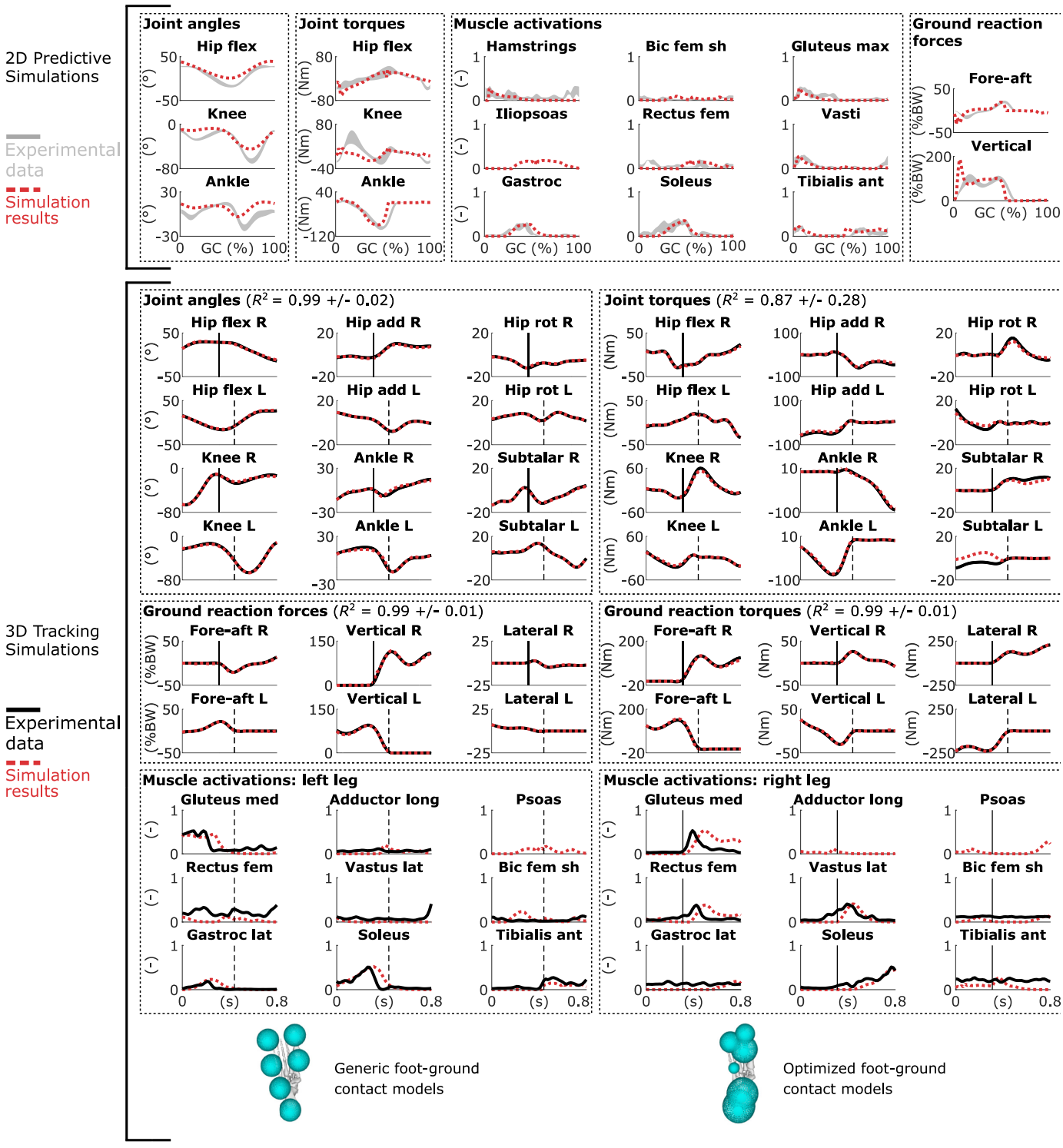

Fig 5. Results from trajectory optimization of walking. (Top) Results from 2D predictive simulations of walking (joint angles: flex is flexion, GC is gait cycle; muscle activations: bic is biceps, fem is femoris, sh is short head, max is maximus, gastroc is gastrocnemius, ant is anterior; ground reaction forces: BW is body weight). Experimental data are shown as mean ± two standard deviations. (Bottom) Results from 3D tracking simulations of walking (joint angles: R is right, L is left, add is adduction, rot is rotation; muscle activations: med is medialis, long is longus, lat is lateralis). The vertical lines indicate right heel strike (solid) and left toe-off (dashed); only part of the gait cycle, when experimental ground reaction forces are available, is tracked. The experimental electromyography data is normalized to peak muscle activations. The foot diagrams depict a down-up view of the configuration of the contact spheres of the right foot pre-calibration (left: generic) and post-calibration (right: optimized). The coefficient of determination R^2 is given for the tracked variables.

https://doi.org/10.1371/journal.pone.0217730.g005

the 3D tracking simulations due to memory issues. For all simulations, the solvers from the HSL collection except ma86 (and ma77 for the 2D predictive simulations) required less CPU time per iteration than mumps. For the 2D predictive and 3D tracking simulations, one case out of 18 (six solvers and three initial guesses) and four cases out of 12 (four solvers and three initial guesses), respectively, were excluded from the comparison as they converged to different solutions.

Using an exact Hessian was more efficient than using an approximated Hessian for the pendulum simulations but not for the 2D predictive simulations (Fig 4). The exact Hessian required less CPU time and fewer iterations than the approximated Hessian for the pendulum simulations (average 2.4 ± 1.2 times faster and 2.5 ± 0.9 times fewer iterations). By contrast, the exact Hessian required more CPU time and iterations than the approximated Hessian for the 2D predictive simulations (average 6.0 ± 0.8 times slower and 2.1 ± 0.2 times more iterations). For the pendulum simulations, 27 cases out of 540 (nine pendulums, six solvers, and 10 initial guesses) were excluded from the comparison as they converged to different solutions with the two Hessian settings. One case was also excluded as it had not converged after 3000 iterations with the exact Hessian but converged in 209 iterations with the approximated Hessian. For the 2D predictive simulations, only results obtained with the solvers ma86 and ma97 were included, since the use of the other solvers led to memory issues. Further, four cases out of six (two solvers and three initial guesses) were excluded from the comparison as they converged to different solutions with the two Hessian settings. Finally, the 3D tracking simulations were not included for this comparison as the large problem size induced memory issues with the exact Hessian.

In the different analyses, we examined the cases that we excluded from the comparison because of convergence to different solutions but we did not find that one derivative scenario, solver, or initial guess consistency led to a local optimum with a lower cost.

The pendulum simulations required at most 21 s and 366 iterations to converge (results obtained with AD-Recorder, mumps, and an approximated Hessian); CPU time and number of iterations depended on the number of degrees of freedom (S1 Movie).

The 2D predictive simulations reproduced salient features of human gait but deviated from experimental data in three noticeable ways (Fig 5; S2 Movie). First, the predicted knee flexion during mid-stance was limited, resulting in small knee torques. Second, the simulations produced less ankle plantarflexion at push-off. Third, the vertical ground reaction forces exhibited a large peak at impact. The simulations converged in less than one CPU minute (average over solutions starting from three initial guesses: 36 ± 17 s and 247 ± 143 iterations; results obtained with AD-Recorder, mumps, and an approximated Hessian).

The 3D tracking simulations accurately tracked the experimental walking data (average coefficient of determination R^2: 0.95 ± 0.17; Fig 5; S3 Movie). Simulated muscle activations also qualitatively resembled experimental electromyography data, even though electromyography was not tracked (Fig 5). The configuration of the contact spheres differed from the generic model after the calibration. The simulations converged in less than 20 CPU minutes (average over simulations starting from two initial guesses: 19 ± 7 minutes and 493 ± 151 iterations; results obtained with AD-Recorder, mumps, and an approximated Hessian).

Discussion

We showed that the use of AD over FD improved the computational efficiency of OpenSim-based trajectory optimization of human movement. Specifically, AD drastically decreased the CPU time spent in evaluating the objective function gradient. This time decrease results from AD's ability to evaluate a Jacobian-transposed-times-vector product through its reverse mode.

The objective function gradient has many inputs (all optimization variables) but only one output. It can thus be evaluated in only one reverse sensitivity sweep; the computational cost is hence proportional to the cost of evaluating the objective function. By contrast, with FD, the computational cost is proportional to the number of optimization variables times the cost of evaluating the objective function. The efficiency benefit of AD also increased with the complexity of the problems. This is expected, since the number of optimization variables increases with problem size; FD thus requires more objective function evaluations, whereas AD still requires only one reverse sweep. In our problems, AD did not outperform FD when evaluating the constraint Jacobian. Yet we expect that AD will be more efficient than FD for trajectory optimization problems in which the number of optimization variables largely exceeds the number of constraints, thereby resulting in faster constraint Jacobian evaluations with AD's reverse mode.

The choice of the objective function influences CPU time. As an illustration, we added a term representing the metabolic energy rate [45] to the objective function of the 2D predictive simulations. Minimizing metabolic energy rate is common in predictive studies of walking [5,7,39]. Solving the resulting optimal control problem was about 60 times faster with AD-Recorder than with FD (although FD required fewer iterations), whereas AD-Recorder was only about 10 times faster than FD without incorporating the metabolic energy rate in the objective function. This increased time difference can be explained by our use of computationally expensive hyperbolic tangent functions to make the metabolic energy rate model twice continuously differentiable, as required when using second-order gradient-based optimization algorithms [39]. Overall, AD reduces the number of function evaluations, which has an even larger effect if these functions are expensive to compute.

The implementation of AD was computationally more efficient through Recorder than through ADOL-C. Specifically, Recorder decreased the CPU time by a factor 4–12 compared to ADOL-C. ADOL-C records all calculations involving differential variables on a sequential data set called a tape [18], which is then evaluated by ADOL-C's *virtual machine*. By contrast, Recorder generates plain C-code. The factor 4–12 is the difference between a *virtual machine* interpreting a list of instructions (ADOL-C) and machine code performing these instructions directly (Recorder).

The effort required to enable the use of AD through Recorder was minimal once OpenSim's source code had been modified for use with the ADOL-C libraries. Indeed, Recorder relies on operator overloading for constructing the expression graphs, which is similar to ADOL-C. The only required change was to replace the *adouble* scalar type (ADOL-C) by the *Recorder* scalar type. Recorder also facilitates the interface with CasADi, since it generates expression graphs in a format from which CasADi can directly generate C-code. This code can then be compiled as a Dynamic-link Library and imported in the CasADi environment without any scripting input required from the user (Fig 2). Using ADOL-C's AD algorithms with CasADi necessitates manually writing C++ code to provide forward and reverse directional derivatives using ADOL-C's drivers in a format recognized by CasADi, which might be prone to errors (Fig 2). Note that the manual effort required for using Recorder or ADOL-C is independent of problem complexity. Overall, using Recorder is more efficient but also simpler than using ADOL-C when solving trajectory optimization problems with CasADi.

The process of converting OpenSim's source code to code that compiles with the AD tools (ADOL-C and Recorder) was a considerable but one-time effort. OpenSim-based trajectory optimization problems can now be solved through the proposed framework while benefiting from AD and without any additional developments. We made our OpenSim-based AD framework available so that others can build upon our work. Importantly, using AD does not increase the complexity for the end user as compared to using FD. Indeed, the simulation

framework relies on CasADi that provides evaluations of function derivatives to the NLP solver. Hence, the user does not need to re-implement AD's forward and reverse algorithms. It is also worth mentioning that, in this study, we used Recorder to enable the use of AD with OpenSim. However, Recorder is a general C++ class that could be applied to any other C++ code for use with CasADi. Compiling existing source code with Recorder would require replacing the scalar type of active variables (i.e., differentiable quantities) with the *Recorder* scalar type. Our study suggests that this programming effort might be particularly valuable when the goal is to solve complex trajectory optimization problems. Specifically, our results showed that the difference between AD and FD increased with problem size. Users might thus consider the programming effort only when the aim is to solve multiple complex problems and when they are not satisfied with the computational performance obtained with FD.

It is difficult to provide guidelines for the linear solver selection based on our results, as their efficiency was problem-dependent. In contrast with mumps, the solvers from the HSL collection do not freely come with CasADi and are only free to academics. Hence, our study does not support the extra effort to obtain them since they did not consistently outperform mumps in our applications. Yet an in-depth analysis of the solvers' options and underlying mathematical details should be considered in future work.

The use of an exact Hessian, rather than an approximated Hessian, improved the computational efficiency for the pendulum simulations but not for the walking simulations. For the 2D walking simulations, using an exact Hessian required more CPU time but also more iterations. This might seem surprising, since an exact Hessian is expected to provide more accurate information and, therefore, lead to convergence in fewer iterations. However, IPOPT requires the Hessian to be positive definite when calculating a Newton step to guarantee that the step is in the descent direction. When this is not the case, the Hessian is approximated with a positive definite Hessian by adding the identity matrix multiplied by a regularization term to the Hessian [36]. We observed that for the 2D predictive simulations, the magnitude of the regularization term was much greater than for the pendulum simulations. Yet excessive regularization might degrade the performance of the algorithm, as regularization alters the second-order derivative information and causes IPOPT to behave more like a steepest-descent algorithm [46]. The approximated Hessian requires no regularization, which likely explains the difference in number of iterations. Overall, convexification of the currently non-convex optimal control problems is expected to further improve the computational efficiency [9].

Our comparison of derivative scenarios (AD-ADOLC, AD-Recorder, and FD), linear solvers (mumps and the HSL collection), and Hessian calculation schemes was based on several specific choices. First, we solved all problems using the NLP solver IPOPT, whereas other solvers compatible with CasADi, such as SNOPT [47] and KNITRO [48] (see [21] for detailed list), might behave differently. We selected IPOPT since it is open-source (SNOPT and KNITRO are commercial products), widely used, and well suited for large and very sparse NLPs [21]. Second, we transcribed the optimal control problems into NLPs using a third order Radau quadrature collocation scheme, whereas different orders, schemes (e.g., Legendre), and transcription methods (e.g., trapezoidal and Hermite-Simpson) might lead to different results. We selected quadrature collocation methods as they achieve exponential convergence if the underlying function is sufficiently smooth [1,49]. Third, we used specific models of muscle activation dynamics, contraction dynamics, and compliant contacts, whereas other models might behave differently. We selected models that were continuously differentiable for use with gradient-based optimization algorithms. Finally, our focus was on solving trajectory optimization problems for biomechanical applications with OpenSim. We chose OpenSim as it is an open-source and widely used software package in biomechanics. The difference in computational performance between AD and FD might thus vary with other software packages and applications.

Investigating all these other modeling and computational choices was out of the scope of this study but might be useful for helping users select the best settings for their applications. Overall, our study underlined the computational benefit of using AD over FD for trajectory optimization in biomechanics, which is in agreement with previous research in robotics (e.g., [27]).

The 2D predictive and 3D tracking simulations produced realistic movements although deviations remain between simulated and measured data. Modeling choices rather than local optima likely explain these deviations. These choices have a greater influence on the predictive simulations, since deviations from measured data are minimized in tracking simulations, whereas only the motor task goal is specified in the objective function of predictive simulations. Several modeling choices might explain the main deviations for the predictive simulations. First, we did not model stability requirements, which might explain the limited knee flexion during mid-stance [6,39]. Instead, we included muscle activity in the cost function, which might explain why reducing knee torques and, therefore, knee extensor activity was optimal. Second, the model did not include a metatarsophalangeal joint, which might explain the limited ankle plantarflexion at push-off; similar ankle kinematics have indeed been observed experimentally when limiting the range of motion of the metatarsophalangeal joint [50]. Third, the lack of knee flexion combined with the simple trunk model (i.e., one degree of freedom controlled by one ideal torque actuator) might explain the high vertical ground reaction forces at impact [6]. Finally, the goal of the motor task (i.e., minimizing muscle fatigue) likely does not fully explain the control strategies governing human walking. In this study, the focus was on evaluating different computational choices but future work should exploit the improved computational efficiency to explore how modeling choices affect the correspondence between simulated and measured quantities.

Our results indicate that AD is particularly beneficial with increasingly complex models. Hence, our OpenSim-based AD framework might allow researchers to rely on complex models, such as three-dimensional muscle-driven neuro-musculoskeletal models, in their studies. This model complexity might be highly desirable when studying, for instance, the impact of treatment on gait performance in patients with neuro-musculoskeletal disorders. Indeed, in such cases, the model should be complex enough to describe the musculoskeletal structures and motor control processes underlying gait that may be affected by treatment. Previous studies based on predictive models reported high computational times and were therefore limited to few predictions when relying on complex musculoskeletal models [5,8,51]. Using AD has the potential to drastically decrease the computational time of such predictive simulations, thereby extending their application.

Conclusions

In this study, we enabled the use of AD when performing OpenSim-based trajectory optimization of human movement. We showed that using AD drastically improved the computational efficiency of such simulations. This improved efficiency is highly desirable for researchers using complex models or aiming to implement such models in clinical practice where time constraints are typically more stringent than in research context. Overall, the combination of AD with other efficient numerical tools such as direct collocation and implicit differential equations allows overcoming the computational roadblocks that have long limited the use of trajectory optimization for biomechanical applications. In the future, we aim to exploit this computational efficiency to design optimal treatments for neuro-musculoskeletal disorders, such as cerebral palsy.

Supporting information

S1 Appendix. Example source code. Recorder provides the expression graph of the function to differentiate as MATLAB source code in a format that CasADi's AD algorithms can then

transform into C-code. This file provides MATLAB and C source code resulting from applying these two steps on the example function from Fig 1.
(PDF)

S1 Movie. Pendulum-based simulations of perturbed balance. The pendulums have between two and 10 degrees of freedom. The playback speed is 0.2 times real-time.
(AVI)

S2 Movie. 2D muscle-driven predictive simulation of walking (1.33 m s^{-1}). The playback speed is 0.2 times real-time.
(MP4)

S3 Movie. 3D muscle-driven tracking simulation of walking. The pink model (tracking simulation results) tracks the motion of the white model (experimental data). The playback speed is 0.2 times real-time.
(MP4)

Acknowledgments

The authors would like to thank Michael Sherman for helpful technical discussions.

Author Contributions

Conceptualization: Antoine Falisse, Gil Serrancolí, Friedl De Groote.

Data curation: Antoine Falisse, Gil Serrancolí.

Formal analysis: Antoine Falisse, Gil Serrancolí.

Funding acquisition: Antoine Falisse, Gil Serrancolí, Friedl De Groote.

Investigation: Antoine Falisse, Gil Serrancolí, Christopher L. Dembia.

Methodology: Antoine Falisse, Gil Serrancolí, Christopher L. Dembia, Joris Gillis, Friedl De Groote.

Project administration: Friedl De Groote.

Resources: Antoine Falisse, Gil Serrancolí, Christopher L. Dembia, Joris Gillis.

Software: Antoine Falisse, Gil Serrancolí, Christopher L. Dembia, Joris Gillis.

Supervision: Friedl De Groote.

Validation: Antoine Falisse, Gil Serrancolí.

Visualization: Antoine Falisse, Gil Serrancolí.

Writing – original draft: Antoine Falisse.

Writing – review & editing: Antoine Falisse, Gil Serrancolí, Christopher L. Dembia, Joris Gillis, Friedl De Groote.

References

1. Kelly M. An introduction to trajectory optimization: how to do your own direct collocation. SIAM Rev. 2017; 59(4):849–904.
2. van den Bogert AJ, Hupperets M, Schlarb H, Krabbe B. Predictive musculoskeletal simulation using optimal control: Effects of added limb mass on energy cost and kinematics of walking and running. P I Mech Eng P-J Spo. 2012; 226(2):123–33.

3. Meyer AJ, Eskinazi I, Jackson JN, Rao A V., Patten C, Fregly BJ. Muscle synergies facilitate computational prediction of subject-specific walking motions. Front Bioeng Biotechnol. 2016; 4:77. https://doi.org/10.3389/fbioe.2016.00077 PMID: 27790612
4. Lin Y-C, Pandy MG. Three-dimensional data-tracking dynamic optimization simulations of human locomotion generated by direct collocation. J Biomech. 2017; 59:1–8. https://doi.org/10.1016/j.jbiomech.2017.04.038 PMID: 28583674
5. Anderson FC, Pandy MG. Dynamic optimization of human walking. J Biomech Eng. 2001; 123(5):381–90. https://doi.org/10.1115/1.1392310 PMID: 11601721
6. Ackermann M, van den Bogert AJ. Optimality principles for model-based prediction of human gait. J Biomech. 2010; 43(6):1055–60. https://doi.org/10.1016/j.jbiomech.2009.12.012 PMID: 20074736
7. Song S, Geyer H. A neural circuitry that emphasizes spinal feedback generates diverse behaviours of human locomotion. J Physiol. 2015; 593(16):3493–511. https://doi.org/10.1113/JP270228 PMID: 25920414
8. Lin Y-C, Walter JP, Pandy MG. Predictive simulations of neuromuscular coordination and joint-contact loading in human gait. Ann Biomed Eng. 2018; 46(8):1216–27. https://doi.org/10.1007/s10439-018-2026-6 PMID: 29671152
9. De Groote F, Pipeleers G, Jonkers I, Demeulenaere B, Patten C, Swevers J, et al. A physiology based inverse dynamic analysis of human gait: potential and perspectives. Comput Methods Biomech Biomed Engin. 2009; 12(5):563–74. https://doi.org/10.1080/10255840902788587 PMID: 19319704
10. De Groote F, Kinney AL, Rao AV, Fregly BJ. Evaluation of direct collocation optimal control problem formulations for solving the muscle redundancy problem. Ann Biomed Eng. 2016; 44(10):2922–36. https://doi.org/10.1007/s10439-016-1591-9 PMID: 27001399
11. Lee L-F, Umberger BR. Generating optimal control simulations of musculoskeletal movement using OpenSim and MATLAB. PeerJ. 2016; 4:e1638. https://doi.org/10.7717/peerj.1638 PMID: 26835184
12. van den Bogert AJ, Blana D, Heinrich D. Implicit methods for efficient musculoskeletal simulation and optimal control. Procedia IUTAM. 2011; 2:297–316. https://doi.org/10.1016/j.piutam.2011.04.027 PMID: 22102983
13. Wächter A, Biegler LT. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math Program. 2006; 106(1):25–57.
14. Griewank A, Walther A. Evaluating derivatives: principles and techniques of algorithmic differentiation. 2nd ed. SIAM; 2008. 448 p.
15. Nocedal J, Wright SJ. Calculating Derivatives. In: Mikosch T V., Resnick S I, Robinson SM, editors. Numerical Optimization. 2nd ed. Springer-Verlag New York; 2006. p. 193–219.
16. Gebremedhin AH, Manne F, Pothen A. What color is your Jacobian? Graph coloring for computing derivatives. SIAM Rev. 2005; 47(4):629–705.
17. Andersson JAE. A general-purpose software framework for dynamic optimization. KU Leuven; 2013.
18. Walther A, Griewank A. Getting started with ADOL-C. In: Naumann U, Schenk O, editors. Combinatorial Scientific Computing. Chapman & Hall/CRC Computational Science; 2012. p. 181–202.
19. CppAD: A package for differentiation of C++ algorithms. https://projects.coin-or.org/CppAD.
20. Weinstein MJ, Rao A V. A source transformation via operator overloading method for the automatic differentiation of mathematical functions in MATLAB. ACM Trans Math Softw. 2016; 42(2):11:1–11:44.
21. Andersson JAE, Gillis J, Horn G, Rawlings JB, Diehl M. CasADi: a software framework for nonlinear optimization and optimal control. Math Program Comput. 2019; 11(1):1–36.
22. Falisse A, Van Rossom S, Jonkers I, De Groote F. EMG-driven optimal estimation of subject-specific Hill model muscle-tendon parameters of the knee joint actuators. IEEE Trans Biomed Eng. 2017; 64(9):2253–62. https://doi.org/10.1109/TBME.2016.2630009 PMID: 27875132
23. Patterson MA, Rao A V. GPOPS-II: a MATLAB software for solving multiple-phase optimal control problems using hp-adaptive Gaussian quadrature collocation methods and sparse nonlinear programming. ACM Trans Math Softw. 2014; 41(1):1:1–1:37.
24. Delp SL, Anderson FC, Arnold AS, Loan P, Habib A, John CT, et al. OpenSim: open-source software to create and analyze dynamic simulations of movement. IEEE Trans Biomed Eng. 2007; 54(11):1940–50. https://doi.org/10.1109/TBME.2007.901024 PMID: 18018689
25. Seth A, Hicks JL, Uchida TK, Habib A, Dembia CL, Dunne JJ, et al. OpenSim: simulating musculoskeletal dynamics and neuromuscular control to study human and animal movement. PLOS Comput Biol. 2018; 14(7):e1006223. https://doi.org/10.1371/journal.pcbi.1006223 PMID: 30048444
26. Sherman MA, Seth A, Delp SL. Simbody: multibody dynamics for biomedical research. Procedia IUTAM. 2011; 2:241–61. PMID: 25866705

27. Giftthaler M, Neunert M, Stäuble M, Frigerio M, Semini C, Buchli J. Automatic differentiation of rigid body dynamics for optimal control and estimation. Adv Robot. 2017; 31(22):1225–37.
28. Tedrake R. and the Drake Development Team. Drake: model-based design and verification for robotics. https://drake.mit.edu. 2019.
29. Docquier N, Poncelet A, Fisette P. ROBOTRAN: A powerful symbolic gnerator of multibody models. Mech Sci. 2013; 4(1):199–219.
30. Wojtusch J, Kunz J, Stryk O Von. MBSlib-An efficient multibody systems library for kinematics and dynamics simulation, optimization and sensitivity analysis. IEEE Robot Autom Lett. 2016; 1(2):954–60.
31. Carpentier J, Saurel G, Buondonno G, Mirabel J, Lamiraux F, Stasse O, et al. The Pinocchio C ++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. 2019 IEEE/SICE Int Symp Syst Integr. 2019;614–9.
32. Guennebaud G, Jacob B, Others. Eigen v3. http://eigen.tuxfamily.org. 2010.
33. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv [Preprint] arXiv:160304467 Available online at: https://arxiv.org/abs/160304467. 2016;
34. The Theano Development Team, Al-Rfou R, Alain G, Almahairi A, Angermueller C, Bahdanau D, et al. Theano: A Python framework for fast computation of mathematical expressions. arXiv [Preprint] arXiv:160502688 Available online at: https://arxiv.org/abs/160502688. 2016;1–19.
35. Degrave J, Hermans M, Dambre J, Wyffels F. A differentiable physics engine for deep learning in robotics. Front Neurorobot. 2019; 13(March):1–9.
36. Nocedal J, Wright SJ. Interior-point methods for nonlinear programming. In: Mikosch T V., Resnick S I, Robinson SM, editors. Numerical Optimization. 2nd ed. Springer-Verlag New York; 2006. p. 563–97.
37. Betts JT. The optimal control problem. In: Practical Methods for Optimal Control and Estimation Using Nonlinear Programming. 2nd ed. Philadelphia: SIAM; 2010. p. 123–218.
38. Hamner SR, Seth A, Delp SL. Muscle contributions to propulsion and support during running. J Biomech. 2010; 43(14):2709–16. https://doi.org/10.1016/j.jbiomech.2010.06.025 PMID: 20691972
39. Koelewijn AD, Dorschky E, van den Bogert AJ. A metabolic energy expenditure model with a continuous first derivative and its application to predictive simulations of gait. Comput Methods Biomech Biomed Engin. 2018; 21(8):521–31. https://doi.org/10.1080/10255842.2018.1490954 PMID: 30027769
40. Raasch CC, Zajac FE, Ma B, Levine WS. Muscle coordination of maximum-speed pedaling. J Biomech. 1997; 30(96):595–602.
41. Zajac FE. Muscle and tendon: properties, models, scaling, and application to biomechanics and motor control. Crit Rev Biomed Eng. 1989; 17(4):359–411. PMID: 2676342
42. van den Bogert AJ, Geijtenbeek T, Even-Zohar O, Steenbrink F, Hardin EC. A real-time system for biomechanical analysis of human movement and muscle function. Med Biol Eng Comput. 2013; 51 (10):1069–77. https://doi.org/10.1007/s11517-013-1076-z PMID: 23884905
43. Amestoy PR, Duff IS, L'Excellent J-Y. Multifrontal parallel distributed symmetric and unsymmetric solvers. Comput Methods Appl Mech Eng. 2000; 184(2–4):501–20.
44. HSL. A collection of Fortran codes for large scale scientific computation. http://www.hsl.rl.ac.uk/.
45. Bhargava LJ, Pandy MG, Anderson FC. A phenomenological model for estimating metabolic energy consumption in muscle contraction. J Biomech. 2004; 37(1):81–8. https://doi.org/10.1016/s0021-9290(03)00239-2 PMID: 14672571
46. Nocedal J, Wright SJ. A regularization procedure. In: Mikosch T V., Resnick S I, Robinson SM, editors. Numerical Optimization. 2nd ed. Springer-Verlag New York; 2006. p. 635–6.
47. Gill PE, Murray W, Saunders MA. SNOPT: An SQP algorithm for large-scale constrained optimization. SIAM Rev. 2005; 47(1):99–131.
48. Byrd RH, Nocedal J, Waltz RA. KNITRO: an integrated package for nonlinear optimization. In: Di Pillo G, Roma M, editors. Large Scale Nonlinear Optimization Nonconvex Optimization and Its Applications. Boston: Springer; 2006.
49. Limebeer DJN, Rao A V. Faster, higher, and greener: Vehicular optimal control. IEEE Control Syst Mag. 2015; 35(2):36–56.
50. Hall C, Nester CJ. Sagittal plane compensations for artificially induced limitation of the first metatarsophalangeal joint: a preliminary study. J Am Podiatr Med Assoc. 2004; 94(3):269–74. PMID: 15153589
51. Miller RH. A comparison of muscle energy models for simulating human walking in three dimensions. J Biomech. 2014; 47(6):1373–81. https://doi.org/10.1016/j.jbiomech.2014.01.049 PMID: 24581797