Algorithmic differentiation improves the computational efficiency of OpenSim-based optimal control simulations of movement

Antoine Falisse¹*, Gil Serrancoli²*, Christopher L. Dembia³, Joris Gillis⁴,⁵, Friedl De Groote¹

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

* Corresponding author
E-mail: antoine.falisse@kuleuven.be (AF)

*These authors contributed equally to this work.
Abstract

Algorithmic differentiation (AD) is an alternative to finite differences (FD) for evaluating function derivatives. The primarily aim of this study was to demonstrate the computational benefits of using AD instead of FD in OpenSim-based optimal control simulations. 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 perform optimal control simulations. 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 optimal control simulations as compared to more common FD. Overall, combining AD with direct collocation and implicit differential equations decreases the computational burden of optimal control simulations, which will facilitate their use for biomechanical applications.
Introduction

Combining musculoskeletal modeling and dynamic simulation is a powerful approach to study the mechanisms underlying human movements. There exist two types of model-based dynamic simulations. Inverse simulations identify controls (e.g., muscle excitations) underlying a given movement whereas forward simulations generate the movement based on the controls. Forward simulations are commonly combined with numerical optimization techniques to compute optimal controls. In the resultant optimal control problems, an objective function is minimized subject to constraints describing the musculoskeletal dynamics. Researchers have used such optimal control simulations for tracking measured movements [1–3] or for predicting de novo movements [4–7]. Forward simulations have the potential to reveal cause-effect relationships that cannot be explored through inverse simulations. Nevertheless, the nonlinearity and stiffness of the equations describing the dynamic constraints cause the underlying optimal control problems to be challenging to solve and computationally expensive [4,6,7]. 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 use simpler inverse simulations.

Over the last decade, the increase in computer performance and the use of efficient numerical methods have equipped researchers with better tools for generating computationally efficient optimal control simulations. In particular, direct collocation methods [3,5,7–10] and implicit formulations of the musculoskeletal dynamics [9,11] have become popular. Direct collocation reduces the sensitivity of the objective function to the optimization variables, compared to other methods such as direct shooting [4], by reducing the time horizon of the integration. Direct collocation converts optimal control problems into large sparse nonlinear programming problems (NLP) that readily available NLP solvers (e.g., IPOPT [12]) 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 [9] or invert a mass matrix that is near-singular due to
body segments with a large range of masses and moments of inertia [11]. 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 \[ y = f(y). \])

Algorithmic differentiation (AD) is another numerical tool that can improve the efficiency of
optimal control simulations [9]. Yet it has been underused in biomechanics. AD is a technique for
evaluating derivatives of functions represented by computer programs [13]. 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 [13]. By traversing a function’s expression graph while applying the chain rule, AD allows
computing the function derivatives.

**Fig 1. Example of AD forward and reverse modes.** A function \( y = f(x_1, x_2) = \cos x_2 - x_1 x_2 \) 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
\( J = \frac{\partial f}{\partial x} \) relates \( \dot{x}_1 \) and \( \dot{x}_2 \) and 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 \( \bar{y} \) and reverse
sensitivities \( \bar{x}_1 \) and \( \bar{x}_2 \).
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 modes depends on the problem dimensions. Consider the function $G : \mathbb{R}^n \rightarrow \mathbb{R}^m$ describing the $m$ NLP constraints $y$ as a function of the $n$ optimization variables $x$. The constraint Jacobian $J = \frac{\partial y}{\partial x}$ is a matrix with size $m \times n$. In the forward mode, $J$ relates forward seeds $\dot{x}$ and forward sensitivities $\dot{y} : \dot{y} = J \dot{x}$ (example in Fig 1). In the reverse mode, $J^T$ relates reverse seeds $\bar{x}$ and reverse sensitivities $\bar{y} : \bar{y} = J^T \bar{x}$ (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 (CPU) time as compared to the forward mode [13,14].

OpenSim [15,16] and its dynamics engine Simbody [17] are widely used open-source software packages for musculoskeletal modeling and dynamic simulation but do not leverage tools for AD. Two main approaches exist for adding AD to existing software. In the operator overloading (OO) approach, the AD 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 OO in C++) [13,14]. In the source code transformation (SCT) 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. SCT is inherently faster than OO, but may not be readily available for all features of a programming language.

Once a computer program supports AD, the program can be differentiated using the AD algorithms to evaluate the derivative matrices required by the NLP solver. Note that, like FD, AD algorithms can also exploit the sparsity of these matrices resulting from using direct collocation [18]. Of the many open-source AD frameworks, CasADi [14,19] is a modern actively developed tool with
many additional features (e.g., code generation) and also 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 SCT is applied. The resultant expression graph can be code-generated to
achieve the computational efficiency of pure SCT.

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
OO AD tool ADOL-C [20] and we developed our own AD tool Recorder that uses OO to construct an
expression graph on which SCT is applied. Second, we interfaced OpenSim with CasADi, enabling
optimal control simulations 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 optimal control simulations 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 [21]. 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 OO 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 [20]. 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.

**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 ($F_{\text{fwd}}$) and reverse ($F_{\text{rev}}$) 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 $F_{\text{fwd}}$ and $F_{\text{rev}}$. 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 $F$, $F_{\text{fwd}}$, and $F_{\text{rev}}$. The AD-Recorder approach combines OO, when generating the expression graph, and SCT, when processing the expression graph to generate C-code for $F$, $F_{\text{fwd}}$, and $F_{\text{rev}}$. In both approaches, the code comprising $F$, $F_{\text{fwd}}$, and $F_{\text{rev}}$ 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 $F_{\text{fwd}}$ and $F_{\text{rev}}$ to construct the full derivative matrices.

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 OO and the speed of SCT. 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 of example
from Fig 1). Note that this workflow is currently only practical when the branches (if-tests) 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 made utilizable an OpenSim function within the CasADi environment by compiling the
function and its derivatives as a Dynamic-link Library (DLL) that then is imported as an external function
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).

**Optimal control simulations to evaluate computational choices**

We designed three example optimal control simulations to evaluate different computational
choices (see Table 1 for detailed formulations). The simulations are optimal control problems whose
general formulation consists of computing the controls $u(t)$, states $x(t)$, and time-independent
parameters $p$ minimizing an objective functional:

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

where $t_i$ and $t_f$ are initial and final times, and $t$ is time. This objective functional is subject to dynamic
constraints:

$$ \dot{x}(t) = f(x(t), u(t), p), \quad (2) $$
and to algebraic path constraints:

$$g_{\min} \leq g(x(t), u(t), p) \leq g_{\max}$$  \hspace{1cm} (3)$$

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

$$x_{\min} \leq x(t) \leq x_{\max}$$  \hspace{1cm} (4)$$

$$u_{\min} \leq u(t) \leq u_{\max}$$  \hspace{1cm} (5)$$

$$p_{\min} \leq p \leq p_{\max}$$  \hspace{1cm} (6)$$

198

Table 1. Formulation of the optimal control problems

| Example 1 | Example 2 | Example 3 |
|-----------|-----------|-----------|
| Pendulum simulations | 2D predictive simulations of walking | 3D tracking simulations of walking |
| **States** $x(t)$ | Joint positions $\theta$ and velocities $v$ | Muscle activations $\phi$ and tendon forces $\tilde{F}_T$ Joint positions $\theta$ and velocities $v$ |
| **Controls** $u(t)$ | Derivatives of $v$ (accelerations): $u_{dv}$ Joint torques $\Omega_\tau$ | Derivatives of $\phi$: $\dot{\phi}_{dv}$ Derivatives of $F_C$: $\dot{F}_{Cv}$ |
| **Parameters** $p$ | Half gait cycle duration $T_f$ | Contact sphere transversal plane locations $p_{cl}$ |
| **Bounds** | | Arm activations $\phi_{arma}$ |
| $s_q = 3, \bar{s}_q = \bar{s}/s_q$ | $0 \leq \phi \leq 1$ | $0 \leq \phi \leq 1$ |
| $\bar{s}_q = 0.2, \bar{v} = \frac{v}{s_q}$ | $0 \leq T_f \leq T_{fub} = 5$ | $0 \leq T_f \leq T_{fub} = 5$ |
| $0 < v < s_q$ | $-T_f/100 \leq u_{dv} \leq T_f/100$ | $-T_f/100 \leq u_{dv} \leq T_f/100$ |
| $0 < v < s_q$ | $\tau_d = 60 ms, \tau_e = 15 ms$ | $\tau_d = 60 ms, \tau_e = 15 ms$ |
| $s_{\Omega_\tau} = 0.1 \frac{s_{\tau}}{s_q}$ | $1 \leq u_{dv} \leq \Omega_{\tauub} \Omega_{\tauub}$ | $1 \leq u_{dv} \leq \Omega_{\tauub} \Omega_{\tauub}$ |
| $s_{u_{dv}} = \frac{s_{\Omega_\tau}}{s_q}$ | $q_{lvub} = q_{lv} \leq q_{lvub}$ | $q_{lvub} = q_{lv} \leq q_{lvub}$ |
| $u_{dv} = \frac{u}{s_q}$ | $u_{dv} = \frac{u_{dv}}{s_q}$ | $u_{dv} = \frac{u_{dv}}{s_q}$ |
| $s_T = 100, q_T = 100/v_T$ | $s_{\Omega_\tau} = 100, s_{\Omega_\tau} = 100$ | $s_{\Omega_\tau} = 100, s_{\Omega_\tau} = 100$ |
| Objective function $L = \frac{w_1 \|u\|_2^2 + L_p}{L_p}$ | Objective function $L = \frac{w_1 \|u\|_2^2 + \bar{w}_1 \|u\|_2^2}{L_p}$ | Objective function $L = \frac{w_1 \|u\|_2^2 + \bar{w}_1 \|\dot{u}\|_2^2 + \bar{w}_1 \|\ddot{u}\|_2^2}{L_p}$ |
| Dynamic constraints $(dv/dt)/s_q = v/s_q$ | Dynamic constraints $(dv/dt)/s_q = v/s_q$ | Dynamic constraints $(dv/dt)/s_q = v/s_q$ |
Path constraints

\[ T = f_s(q,v,u_d) \]
\[ \dot{q}(0) = q(1) = \dot{q}(0) = \dot{q}(1) \]
\[ 0 \leq s_d u_d + a ft_d \]
\[ s_d u_d + a ft_d \leq 1 ft_d \]
\[ f_c(a, F) = 0 \]
\[ T = f_s(q,v,u) \]
\[ T_{pelvis} = 0 \]
\[ T_{\text{ll},trunk} = \sum_{m=1}^{M} MA_m F_m \]
\[ T_{\text{arms}}, q_{\text{pelvis},\text{tor}}(t_f) - q_{\text{pelvis},\text{tor}}(0) \]
\[ \tau_f = 1.33 \]

Controls: we introduced controls that are time derivatives of states (implicit formulations).

Bounds: \( lb \) and \( ub \) are for lower and upper bounds, respectively. (Example 2) we manually (man) set the bounds of the joint states and controls. (Example 3) we used measured data from the tracked motion (\( \dot{q}, \dot{v}, \ddot{u}_d \)) to set the bounds of the joint states and controls; \( \text{min} \) and \( \text{max} \) are maximum and minimum values during the motion, and \( \tau \) is range of motion. We used values from the literature [3] to set the bounds of the contact model parameters (\( \dot{p}_c \) and \( \ddot{p}_c \)).

Scaling: (Example 1) we used time scaling for the joint states and controls. (Example 2 and 3) we scaled joint states and controls, and tendon forces such that their lower and upper bounds were between -1 and 1. The other muscle, trunk, and arm states and controls were not scaled. (Example 3) we scaled the contact model parameters such that their lower and upper bounds were -0.5 and 0.5, respectively.

Objective function: to avoid singular arcs, situations for which controls are not uniquely defined by the optimality conditions [22], we appended a penalty function \( L_p \) (with low weight) with the remaining controls to the cost function \( L \). (Example 2) we scaled the objective function by the distance travelled \( d \). (Example 3) we tracked experimental joint positions (\( q \)), ground reaction forces (\( \text{GRF} \)) and torques (\( \text{GRt} \)), and joint torques of the lower limbs, trunk, and arms (\( T_{\text{ll},\text{trunk,arms}} \)).

Dynamic constraints: we scaled the dynamic constraints using the same scale factors as for the states [22]. We used implicit formulations except for the trunk (example 2) and arms (example 3). Path constraints: \( f_s(\cdot) \) computes net joint torques \( T \) according to the skeleton dynamics, \( f_c(\cdot) \) describes muscle contraction dynamics according to a Hill type muscle model [9], \( MA_m \) is moment arm of muscle \( m \), \( \vec{x}(\cdot) \) contains all states except the pelvis forward position \( q_{\text{pelvis,tor}} \) (symmetry constraint), 1.33 m s\(^{-1} \) is the prescribed gait speed, and \( T_p \) are passive joint torques [4].

In the first example, we perturbed the balance of 9 inverted pendulums, with between 2 and 10 degrees of freedom (DOFs), by applying a backward translation to their base of support. The optimal control problem identified the joint torques necessary to restore the pendulums’ 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 DOFs, 18 muscles actuating the lower limbs, one ideal torque actuator at the trunk, and 2 contact spheres per foot [15]). We identified muscle excitations and walking cycle duration that minimized a weighted sum of muscle fatigue (i.e., muscle activations at the third power [5]) 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).

In the third example, we performed tracking simulations of walking with a three-dimensional (3D) musculoskeletal model (29 DOFs, 92 muscles actuating the lower limbs and trunk, 8 ideal torque actuators at the arms, and 6 contact spheres per foot [3,15,23]) 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 [15,17]. 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 [24]. 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 [8,25] and a Hill-type muscle model [9,26], respectively. We defined muscle-tendon lengths, velocities, and moment arms as a function of joint positions and velocities using polynomial functions [27]. 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 and velocities.

We transcribed each optimal control problem into a nonlinear programming problem (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 $1 \times 10^{-6}$ 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 3 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.

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 [28], which CasADi provides, and an approximated Hessian. Third, we compared different linear solvers, namely mumps with the collection of solvers from HSL [29], 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.

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 with 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, 1 and 2 cases, respectively, out of 9 (3 derivative scenarios and 3 initial guesses) were excluded from the comparison as they converged to different solutions.
Fig 3. Comparison of CPU time (top) and number of iterations (bottom) between FD, 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 DOFs; Pred and Track indicate predictive and tracking simulations, respectively. The results were obtained using mumps and an approximated Hessian.

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 2). 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 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, 1 case out of 18 (6 solvers and 3 initial guesses) and 4 cases out of 12 (4 solvers and 3 initial guesses), respectively, were excluded from the comparison as they converged to different solutions.
Table 2. Comparison of CPU time, number of iterations, and CPU time per iteration between linear solvers.

| Solver | Pendulum-based simulations | 2D predictive simulations | 3D tracking simulations |
|--------|-----------------------------|---------------------------|-------------------------|
|        | CPU time | Iteration Number | CPU time | Iteration Number | CPU time | Iteration Number | CPU time | Iteration Number |
| *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 | 0.6 ± 0.3 | 0.8 ± 0.3 | 0.7 ± 0.1 |
| *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 ± 0.1 | 0.6 ± 0.2 | 0.8 ± 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 | 1.7 ± 0.0 | 1.3 ± 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). The ratios are averaged over results from different initial guesses. Ratios larger than one indicate faster convergence, fewer iterations, or more 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.

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 (9 pendulums, 6 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, 4 cases out of 6 (2 solvers and 3 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.
Fig 4. Comparison of CPU 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 DOFs; Pred and Track indicate predictive and tracking simulations, respectively. The results were obtained using all solvers and AD-Recorder.

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 DOFs (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).

Fig 5. Results from optimal control simulations 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 EMG data are 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.

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 (EMG) data, even though EMG 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 optimal control simulations. 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 the problem size; FD thus requires more objective function evaluations, whereas AD still only requires one reverse sweep.
The choice of the objective function influences CPU time. As an illustration, we added a term representing the metabolic energy rate [30] to the objective function of the 2D predictive simulations. Minimizing metabolic energy rate is common in predictive studies of walking [4,6,24]. 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 [24]. 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 [20], 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). In future work, we aim to support computer programs containing branches with Recorder.

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. Our study hence 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 [21]. 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 [31]. 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 [8].

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 [5,24]. 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 (MTP) 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 MTP [32]. Third, the lack of knee flexion combined with the simple trunk model (i.e., one DOF controlled by one ideal torque actuator) might explain the high vertical ground reaction forces at impact [5]. 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.

Conclusions

In this study, we enabled the use of AD when performing OpenSim-based optimal control simulations. We showed that using AD drastically improved the computationally 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 optimal control simulations 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.

Acknowledgments

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

References

1. 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.

2. 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.

3. 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.
4. Anderson FC, Pandy MG. Dynamic optimization of human walking. J Biomech Eng. 2001;123(5):381–90.

5. Ackermann M, van den Bogert AJ. Optimality principles for model-based prediction of human gait. J Biomech. 2010;43(6):1055–60.

6. Song S, Geyer H. A neural circuitry that emphasizes spinal feedback generates diverse behaviours of human locomotion. J Physiol. 2015;593(16):3493–511.

7. 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.

8. 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.

9. 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.

10. Lee L-F, Umberger BR. Generating optimal control simulations of musculoskeletal movement using OpenSim and MATLAB. PeerJ. 2016;4:e1638.

11. van den Bogert AJ, Blana D, Heinrich D. Implicit methods for efficient musculoskeletal simulation and optimal control. Procedia IUTAM. 2011;2:297–316.

12. Wächter A, Biegler LT. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math Progr. 2006;106(1):25–57.

13. 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.

14. Andersson JAE. A General-Purpose Software Framework for Dynamic Optimization. KU Leuven; 2013.

15. 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 T Biomed Eng.
16. 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.

17. Sherman MA, Seth A, Delp SL. Simbody: multibody dynamics for biomedical research. Procedia IUTAM. 2011;2:241–61.

18. Gebremedhin AH, Manne F, Pothen A. What color is your Jacobian? Graph coloring for computing derivatives. SIAM Rev. 2005;47(4):629–705.

19. 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.

20. 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.

21. 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.

22. 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.

23. Hamner SR, Seth A, Delp SL. Muscle contributions to propulsion and support during running. J Biomech. 2010;43(14):2709–16.

24. 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.

25. Raasch CC, Zajac FE, Ma B, Levine WS. Muscle coordination of maximum-speed pedaling. J Biomech. 1997;30(96):595–602.

26. Zajac FE. Muscle and tendon: properties, models, scaling, and application to biomechanics and...
motor control. Crit Rev Biomed Eng. 1989;17(4):359–411.

27. 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.

28. 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.

29. HSL. A collection of Fortran codes for large scale scientific computation. http://www.hsl.rl.ac.uk/.

30. Bhargava LJ, Pandy MG, Anderson FC. A phenomenological model for estimating metabolic energy consumption in muscle contraction. J Biomech. 2004;37(1):81–8.

31. 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.

32. Hall C, Nester CJ. Sagittal plane compensations for artificially induced limitation of the first metatarsophalangeal joint: a preliminary study. J Am Pod Med Assoc. 2004;94(3):269–74.

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.

S1 Movie. Pendulum-based simulations of perturbed balance. The pendulums have between 2 and 10 degrees of freedom. The playback speed is 0.2 times real-time.

S2 Movie. 2D muscle-driven predictive simulation of walking (1.33 m s\(^{-1}\)). The playback speed is 0.2 times real-time.
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.
Fig 3
Fig 5
Forward mode: \( \dot{y} = \frac{\partial f}{\partial x} \dot{x} \)

\[
\begin{align*}
\bar{x}_2 &= -\bar{y} \sin x_2 - \bar{y} x_1 \\
\bar{x}_2 &= w_0 = -\bar{w}_2 \sin w_0 + \bar{w}_3 w_1 \\
w_0 &= x_2 \\
w_0 &= x_2 \\
w_2 &= w_4 \\
w_2 &= -\bar{w}_0 \sin w_0 \\
w_2 &= \cos w_0 \\
w_4 &= \bar{w}_2 - \bar{w}_3 \\
w_4 &= \bar{y} \\
w_4 &= \bar{y} \\
\bar{y} &= f(x_1, x_2) \\
\bar{y} &= \bar{x}_2 \sin x_2 \\
\bar{y} &= -x_1 x_2 - x_1 x_2
\end{align*}
\]

Reverse mode: \( \bar{x} = \frac{\partial f^T}{\partial x} \bar{y} \)