Bioptim, a Python interface for Musculoskeletal Optimal Control in Biomechanics

Benjamin Michaud[a,*,†], François Bailly[a,†], Eve Charbonneau[a], Amedeo Ceglia[a], Léa Sanchez[a] and Mickael Begon[a]

***Abstract*— The abstract**

Keywords – TODO

I. Introduction

Biomechanics researchers rely on numerical simulations of motion to gain understanding on a variety of scientific topics such as the physiological causes of movement disorders and their consequences on health [REF], the estimation of non-measurable physiological quantities (e.g., muscle forces)[REF] and the optimality of human movement [REF]. The musculoskeletal models used in these simulations generally have a large number of degrees of freedom and they are governed by several ordinary differential equations (ODEs) which mainly describe multibody and muscle activation dynamics. The complexity of these systems has led scientists to formulate their simulations as optimal control problems (OCP), relying on efficient non-linear optimization software to find trajectories that fulfill a desired task while enforcing the system dynamics and minimizing a cost (e.g. motion duration, energy expenditure, matching experimental data, etc.). Up to very recently, there was no off-the-shelf software available to the community to quickly formulate and solve such musculoskeletal OCPs. Consequently, researchers had to develop their own solutions, with little or no dissemination to the community, limiting synergies between researchers.

As a result, many approaches coexist to formulate and solve OCPs in the biomechanical literature. The formulation, also called discretization, consists in turning a continuous trajectory optimization problem into a generic discrete non-linear program (NLP) that is solved using a dedicated algorithm. The main family of so-called *direct* transcription methods comes from numerical optimal control. They consist in straightforwardly choosing the state and/or the control as optimization variables at a given number of points along the trajectory and they rely on the integration of the system dynamics between these points.

For instance, the *direct collocation* method has shown its efficiency in some studies investigating human motion [REF, à prendre dans papier MOCO]. It consists in approximating the integration of the system dynamics using polynomials that describe the state and control trajectories. Its main advantages are that it leads to very sparse NLPs, that knowledge about the state trajectory can be used in the initialization, and that it handles unstable systems well [**?**]. Its major disadvantage is that adaptive integration error control implies regridding the whole problem and thus changes the NLP dimensions, discarding its use for such application [REF]. *Direct multiple shooting* is another direct method that was also applied with success in a lot of biomechanics [REF] and robotics [REF] studies. Its advantages are mostly the same as for direct collocation in addition to combine integration error control with fixed NLP dimensions, as it relies on possibly adaptive ODE solvers to integrate the system dynamics. Besides direct methods, other choices can be made, as in [**?**] [+ REF Begon], where the optimization variables are instants at which a switch in the motor strategy occurs, using polynomials function (4th, 5th order) in-between, or in [**?**] [+ REF Huchez, Mombaur, McPhee, Opensim]], where the optimization variables are the coefficients of fourth order polynomial approximations of the states, with linking conditions to enforce the continuity of the controls. These last approaches are less generic than the direct methods as they either require a prior knowledge about the state and control trajectories. Most of the time, when investigating complex biomechanics issues, we do not have this information.

Concerning the non-linear solver, a variety of software exist and have been used to solve transcribed musculoskeletal NLPs. They can use different heuristics: interior point methods

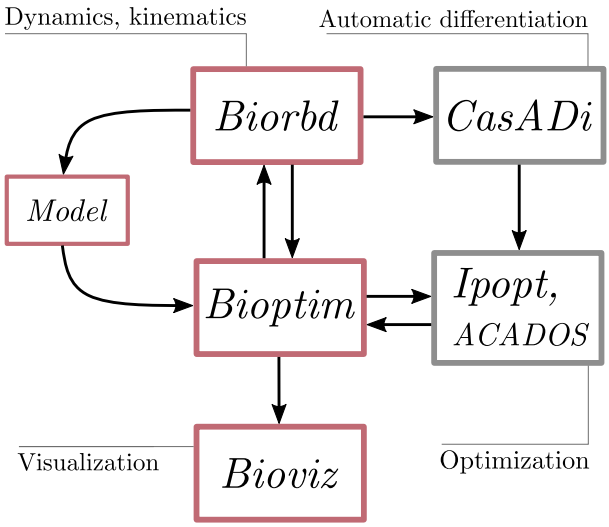

Fig. 1: *Bioptim* dependencies flowchart. The red-boxed software are developed by the S2M team. The *Bioptim* part is further detailed in Fig. 2.

† These authors have contributed equally to this work and share first authorship.
[a] Laboratoire de Simulation et Modélisation du Mouvement, Faculté de Médecine, Université de Montréal, Laval, QC, Canada
* BENJAM@umontreal.ca

(*Ipopt*, [REF]) or sequential quadratic programming (*snopt* [REF], *ACADOS* [REF]), but they are all gradient based. Therefore, derivatives of the NLP cost function and constraints are required to perform optimization. These derivatives can be obtained by finite differences (often implemented but inaccurate thus comprising convergence) or computed exactly using automatic differentiation (requiring to write all dependencies of the software in symbolic variables) [CasADi].

In order to promote the use of musculoskeletal optimal control among biomechanics researcher, we identified a strong need for a dedicated tool, as shown by the recently launched *OpenSim Moco* [REF, Opensim]. The biomechanics community being mainly composed of software users [REF], such a tool should request a flexible user interface written in a widely used high-level and if possible open-source language (e.g. Python) with a low-level core (e.g. C++) for efficiency.

To develop such a software, four interrelated components are essential to us: *i)* a musculoskeletal modeling software, with a visualization module (multibody kinematics and dynamics, muscle dynamics, etc.), *ii)* a method for automatic differentiation, *iii)* a discretization approach, and *iv)* one or several nonlinear programming (NLP) solvers. General-purpose optimal control software (e.g. *GPOPS-II* [REF], *Muscod-II* [REF], *Acado* [REF]) address *ii)* to *iv)* but they need to be interfaced with a musculoskeletal modeling module and they do not provide any built-in biomechanics features (physiological cost functions, kinematic constraints, etc.). In that sense, the aforementioned *OpenSim Moco*, is a welcome initiative that draws its strength from its integration with the widely used *OpenSim*. However, it faces the following limitations: it uses finite differences to avoid the complexity of adapting the *OpenSim* codebase to support automatic differentiation, it uses direct collocation as transcription method, preventing the use of adaptive ODE solvers and it is not as flexible as required by the community, since it requires the user to develop new features, such as new objective functions, in C++.

The objective of the present paper is to introduce *Bioptim*[1], an open-source optimal control software dedicated to musculoskeletal biomechanics. *Bioptim* is based on C++ code for computational efficiency but the user interface is written in Python for flexibility and ease-of-use. The OCP transcription uses direct multiple shooting to preserve the possibility of using arbitrarily accurate ODE solvers for the integration, which is fully parallelized for more efficiency. *Bioptim*'s core is fully written in *CasADi* symbolics to benefit from algorithmic differentiation and to exploit *CasADi*'s interface with several non-linear solvers (*Ipopt*, *SNOPT*). Moreover, *Bioptim* is interfaced with the cutting-edge solver *acados*, a recent NLP solver dedicated to direct multiple shooting, intended for real-time applications. The purpose of *Bioptim* is to allow fast and flexible musculoskeletal OCP formulation and solving by providing a framework with a lot of typical biomechanics problem already implemented and customizable.

The paper is organized as follows: first, the design and implementation of *Bioptim* are described. Next, the versatility and performances of *Bioptim* are shown through a variety of examples available online.

[1] https://github.com/pyomeca/bioptim

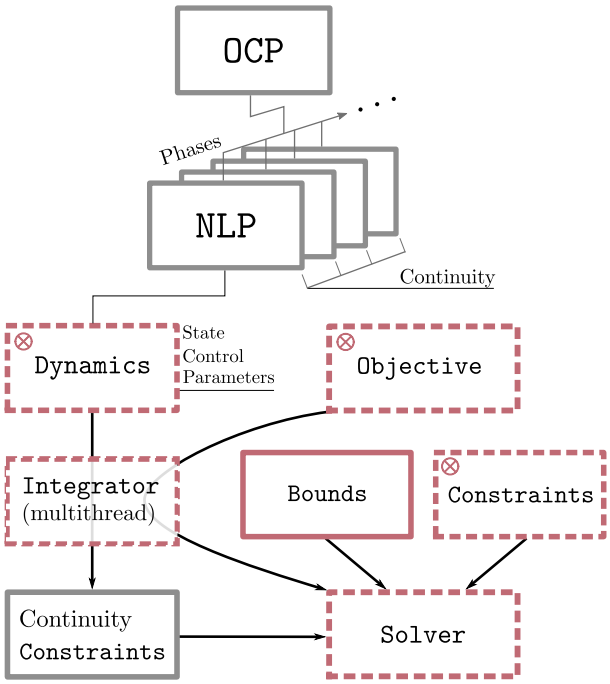

Fig. 2: *Bioptim* design flowchart. The red box correspond to objects that must be filled in by the user. The red-dashed boxes correspond to pre-implemented objects already available to the users. ⊗ stands for easily customizable objects.

II. Implementation and Design

A. Implementation and dependencies

Bioptim is the top layer of a series of libraries (*Biorbd*: dynamics and MSK modeling; *CasADi*: automatic differentiation; *Ipopt*/*ACADOS*: optimization; *Bioviz*: visualization). Within this software suite, *Bioptim*'s main role is to shape the problem to allow its dependencies to communicate efficiently, while providing an intuitive and flexible interface to the user (Fig. 1). Therefore, it was written in Python for its flexibility and its widespread use among researchers. However, all intensive calculations behind the interface are performed in C/C++, keeping *Bioptim* both fast and easy to customize.

B. Design

Bioptim shapes and solves optimal control problems whose two required entries are a model (*.bioMod* file) and an OCP. The model file contains the geometrical characteristics and the segment inertial parameters as well as optional elements, namely, the markers, the actuators of the model (muscles and joint torques possibly with torque/angle/velocity relationships) as well as bounds on joint kinematics and torques. It also allows the user to design or import meshes for visualization purposes. The `OCP` consists in a combination of nonlinear problems (NLPs) that allows for the formulation of multi-staged OCPs. Each `NLP` has the following attributes: *1)* a dynamics type, *2)* an objective function set, *3)* a constraint set, *4)* variables bounds, *5)* a number of shooting points

and the duration of the problem and *6)* initial guesses. Based on these inputs, *Bioptim* properly sets up the multiple shooting transcription of the OCP, with appropriate continuity constraints (between the shooting nodes and the phases) and shapes it up to feed the chosen nonlinear solver (*Ipopt* or *ACADOS*). Next, we develop the different attributes of each `NLP`:

1) Dynamics: The dynamics defines which variables are states ($\mathbf{x}$), controls ($\mathbf{u}$) and parameters ($\mathbf{p}$), the latter being time-independent. Then, it implements the ordinary differential equation governing the state dynamics:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, \mathbf{p}). \tag{1}$$

More than 10 dynamics are already implemented in *Bioptim*, among which the controls can be muscle excitations, muscle activations and/or joint torques, the states can be muscle activations and/or joint kinematics. They can include contact points, external forces, etc. Even if these dynamics types exhaustively span the current usages in biomechanics, a custom dynamics type is also pre-implemented to easily customize problems.

2) Objective function set: In line with the optimal control formalism, there are two main types of objective functions, namely Lagrange and Mayer. Lagrange types are running objectives, integrated over the NLP duration. Mayer types are time-specific objectives. Classically, they correspond to a terminal objective, but to be more versatile, they can be defined at any instant in *Bioptim*.

`Objective functions` can depend on any of the optimization variables, *i.e.* the controls, the states, the parameters and the duration of the problem. A lot of objective function types are already implemented in *Bioptim* ($>$ 20), among which tracking / minimizing, on states / controls / markers / contact forces / problem duration, etc. Should one go missing, a custom objective type is also possible to define.

When declaring the desired list of objective functions for a given NLP, each objective function type is associated with a weight, and the user can choose on which components of the vector variables the objective must apply. If applicable (for tracking objective functions mainly), the user must also specify the numerical target of the objective.

3) Constraint set: Classically, constraints are hard penalties of the optimization problem, i.e., a solution will not be considered optimal, unless all constraints (equality or inequality) are met. The `Constraint` class contains a variety of implemented constraints. Some of them are specific functions, commonly useful in biomechanical problems (e.g. non-slipping contact point, non-linear bounds on torque depending on the state, etc.), the others have their equivalent in the `ObjectiveFunction` class. Should one go missing, a custom constraint type is also possible to define.

4) Bounds: Essentially, the `Bounds` are constraints directly related to the states, the controls and the parameters. They are useful to define model-related constraints such as kinematic, torque or muscle excitation / activation limits.

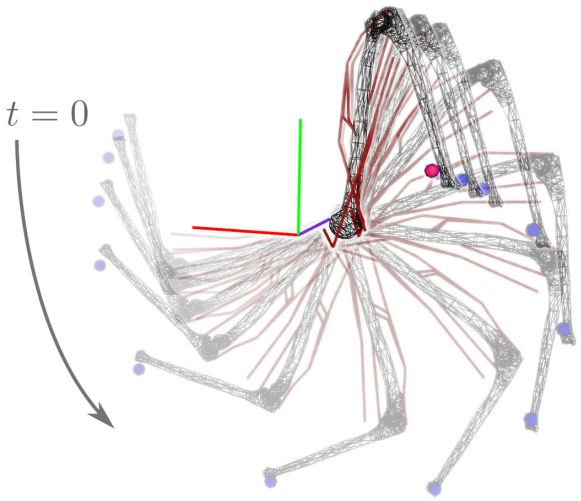

Fig. 3: Snapshots of an optimized activation-driven pointing task with *ACADOS*. The arm starts facing upwards in left hand part of the picture and ends facing downwards in the right hand part. The marker fixed on the ulna head is depicted in blue and the scene-fixed target marker is depicted in red. Red lines show the lines of actions of the muscles.

5) Shooting points and problem duration: In a direct multiple shooting formulation, the total duration of the problem is divided into smaller intervals whose initial values are called shooting points. In *Bioptim*, the user is asked to define a number of shooting points and a problem duration, per phase. Possibly, the problem duration can be part of the optimization variables, allowing for, e.g., minimal time formulations.

6) Initial guesses: Once the problem is set up, the user can provide an `InitialGuess` for all the optimization variables, at each shooting point. This feature aims at providing prior information to the solver. Several `InterpolationTypes` are implemented in *Bioptim* (constant, linear, spline, each shooting point, etc.), to quickly let the user define the initial guesses. A custom `InterpolationType` is also possible to implement.

III. Examples

In this section, six applications are presented to illustrate the versatility of *Bioptim* and give a practical overview on how to use its main features. The settings and performances (convergence time, single shooting integration error, optimized objective) of each OCP are summarized in Tab. **??**. When possible, problems were solved with both *Ipopt* and *ACADOS*. In the following, bold symbols denote vectors and starred ones (*) denote reference or tracked quantities.

A. Muscle activation driven pointing task

In this first example, the goal was to achieve a muscle activation driven pointing task using a 2-DoF arm model with six muscle elements. In addition to muscle-induced torques, pure joint torques were added to compensate for the model weaknesses. The main term (highest weight) of the objective

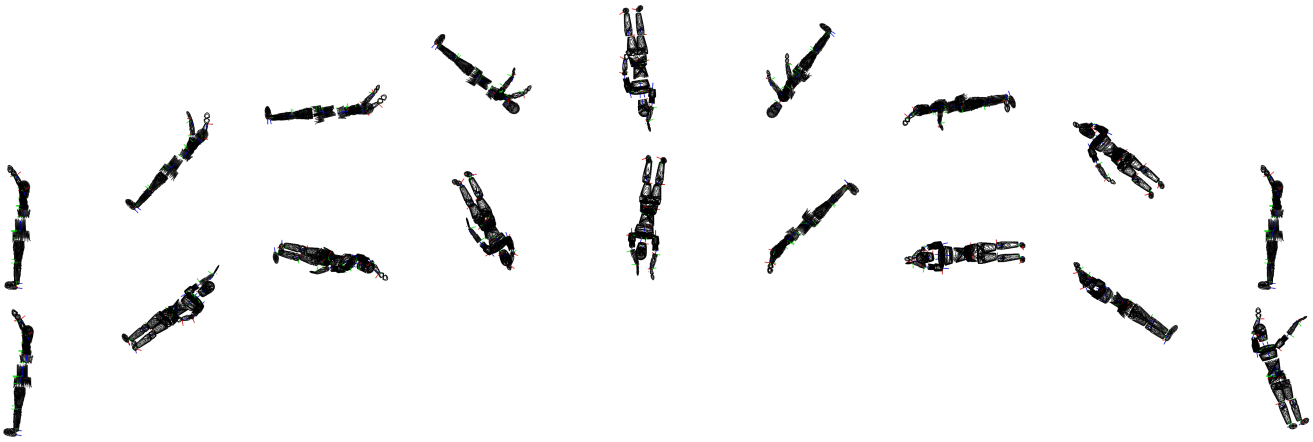

Fig. 4: Snapshots of maximally twisting somersaults driven by shoulder torque actuators and a free base whose rotation is either expressed by Euler angles (top) or by quaternions (bottom).

function (Eq. 2) is a Mayer objective, corresponding to the pointing tasks at the final node, to superimpose two markers, the first one ($\mathbf{m_u}$) fixed in the ulna system of coordinates and the second one ($\mathbf{m_s^*}$) fixed in the scene. The three Lagrange terms were added for control (muscle activation $\mathbf{a}$ and joint torques $\boldsymbol{\tau}$) and state ($\mathbf{x}$) regularization:

$$\mathcal{C} = \omega_1 \underbrace{\|\mathbf{m_u}(T) - \mathbf{m_s^*}\|^2}_{\texttt{TRACK\_MARKERS}} \int_{t=0}^{T} \underbrace{\|\mathbf{a}\|^2}_{\texttt{MIN\_ACTIVATION}} + \underbrace{\|\boldsymbol{\tau}\|^2}_{\texttt{MIN\_TORQUE}} + \underbrace{\|\mathbf{x}\|^2}_{\texttt{MIN\_STATE}} \; dt, \tag{2}$$

where $T = 2\ s$ is the duration of the motion, and $\omega_1 = 1e5$. The problem was discretized using 50 shooting nodes with a 5-steps RK4 integration in-between. The problem was solved using *Ipopt* (with exact Hessian computations) and *ACADOS* (with a Gauss-Newton approximation of the Hessian) resulting in two very close solutions. *ACADOS* was about 50 times faster than *Ipopt* and was better at enforcing the continuity constraints (as shown by the single shooting error in Tab. **??**). *Ipopt* however ended up with a smaller optimized objective (20.8 *vs* 23.2), leading to a more optimal solution than *ACADOS*. Superimposed snapshots of the optimal motion found with *ACADOS* are displayed in Fig. 3. It is worth mentioning that for the purpose of this illustration, no constraint was given on the shoulder range of motion to ensure physiological muscle trajectories.

B. Quaternion base twisting somersault

In this example of acrobatic sports biomechanics, the goal was to maximize the twist rotation (ϕ) of an 8-DoF model in a backward somersault. It illustrates *Bioptim*'s ability to handle quaternionic representations of rotations. The model was composed of a 6-DoF root segment and two 1-DoF torque actuated shoulder joints. Two different numerical descriptions of the root segment rotations were used: Euler angles and quaternions. The objective function was as follows:

$$\mathcal{J} = \int_0^T \underbrace{\omega_1 \dot{\phi}}_{\texttt{MIN\_TWIST}} + \underbrace{\omega_2 \|\boldsymbol{\tau}\|^2}_{\texttt{MIN\_TORQUE}} \; dt, \tag{3}$$

with $\omega_1 = -1$ (resulting in the maximization of the first term) and $\omega_2 = 10^{-6}$, T is the duration of the movement and $\boldsymbol{\tau}$ is the torque control vector. The first term of the objective function (Eq. 3) corresponds to maximizing the change in twist rotation and the second term is for control regularization.

The movement lasted for approximately 1 second and was discretized with 80 shooting nodes. The optimal kinematics were different for the two types of models (Fig. 4) because of the presence of local minima. However, both models take profits of a common biomechanical strategy (i.e. tilting the body to bring closer together the twist axis and the angular momentum vector) highlighting the equivalence of the two rotation representations. Euler angles have the advantage to be easily interpretable, but they suffer from the loss of a DoF at the Gimbal lock (leading to numerical instabilities). The use of a quaternion-based representation tackles this numerical stability issue when a joint is free to rotate on a three-dimensional range of motion. Quaternion's integration must be handled with care [**?**]. Indeed, when representing orientations, quaternions must be unitary and thus belong to a constrained manifold (namely, the unit 3-sphere S^3). However, classical numerical integration schemes such as Runge–Kutta methods treat unit quaternions as if they were arbitrarily defined in $\mathbb{R}^4$. To overcome this challenge, *Bioptim* performs a normalization after each Runge–Kutta iteration to project non-unitary quaternions onto S^3.

C. Pendulum on a spring

This spring-mass-pendulum-based example is presented to introduce *Bioptim*'s ability to use external forces. The goal was to hold the position of a $1\,\mathrm{kg}$ mass hanging on a linear spring attached to the ground. A $0.2\,\mathrm{m}$-long pendulum weighting $10\,\mathrm{kg}$ was attached to the mass and free to rotate in one dimension (Fig. 5). In addition to the spring force, the mass was actuated by a vertical external force (e.g., something

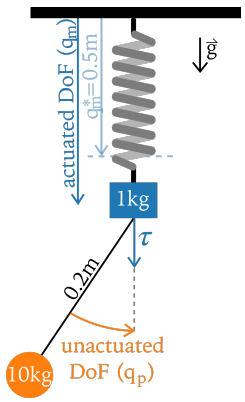

Fig. 5: Spring-mass-pendulum model of Ex. III-C.

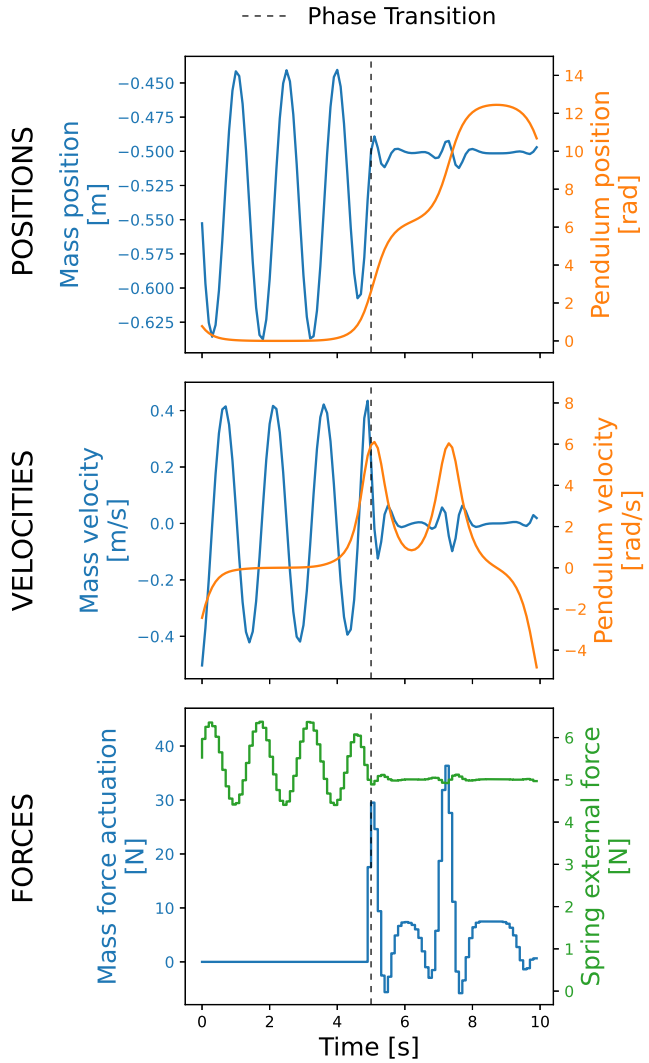

Fig. 6: Two-phases kinematics of the mass-pendulum-spring system. Gray dashed lines show the phase transition, blue lines are related to the mass (position velocity and external force acting on it), red lines are related to the pendulum (position and velocity) and the green line depicts the spring force.

pulling on it) while the pendulum rotation was passive. The system therefore comprised two DoFs, the mass position (q_m) and the pendulum angle (q_p) and one control input, the vertical external force pulling on the mass (τ). The spring force $\mathcal{F}_s$ was:

$$\tau_s = -k * q_m, \tag{4}$$

with k the spring stiffness constant.

The OCP was composed of two phases each lasting for 5 s, with 50 shooting nodes. In the first phase, no objective function was minimized and τ was constrained to be 0, letting the mass oscillating freely. Then, in the second phase, a cost function (Eq. 5) was minimized, to enforce a reference position q_m^* of the mass. This objective function, exclusively composed of Lagrange terms, was formulated as follows:

$$\mathcal{J} = \int_{T/2}^{T} \underbrace{(q_m - q_m^*)^2}_{\texttt{TRACK\_STATE}} + \omega_1 \underbrace{\tau^2\ dt}_{\texttt{MIN\_TORQUE}}\ dt, \tag{5}$$

with $q_m^* = -0.5\,\mathrm{m}$ and $\omega_1 = 10^{-6}$ and T is the duration of the movement. The first term of the objective function (Eq. 5) acts as a position controller for the mass. The second was added for control regularization.

During the first phase, the mass is passively oscillating around its stationary position due to the spring force (Fig. 6). At the beginning of the second phase, when an additional external force acts on the mass, it stabilizes around the targeted position. The standard deviation between the position and the targeted position is $0.04\,\mathrm{m}$. This example highlights the possibility of using optimal control to find activation patterns compensating for external passive forces (e.g., ortheses flexibility, contact surface deformation, interaction between two models, etc.).

D. Multiphase torque driven walking cycle

This example is presented to introduce *Bioptim*'s ability to deal with multiphase locomotion estimation problems including muscle actuations and contact forces. The goal was to estimate muscle activations by tracking markers trajectories and ground reaction forces and moments. The model was a 3D leg with 12 DoFs (6-DoFs pelvis, 3-DoFs hip, 1-DoF knee and 2-DoFs ankle), driven by 17 muscle activations and residual joint torques to compensate for potential muscle actuation weaknesses. The gait cycle was defined from the first heel strike to the end of the swing phase discretized into 90 shooting intervals. To approximate the natural rolling of the foot, the stance was divided into three phases (heel, flatfoot and forefoot contacts) of fixed duration deduced from experimental force platform data and markers position (0.05, 0.36 and 0.16 s). The swing phase lasted 0.38 s. The interaction between the ground and the foot was modeled using a 4-contact points model located at the heel and the forefoot (first, fifth metatarsi and hallux). The optimization problem consisted in minimizing the errors between predicted ($\mathbf{m}$) and reference ($\mathbf{m}^*$) markers trajectories, predicted ($\boldsymbol{\mathcal{F}}$, $\boldsymbol{\mathcal{M}}$) and reference ($\boldsymbol{\mathcal{F}}^*$, $\boldsymbol{\mathcal{M}}^*$) ground reaction forces and moments at all contact points. A regularization term on muscle activations

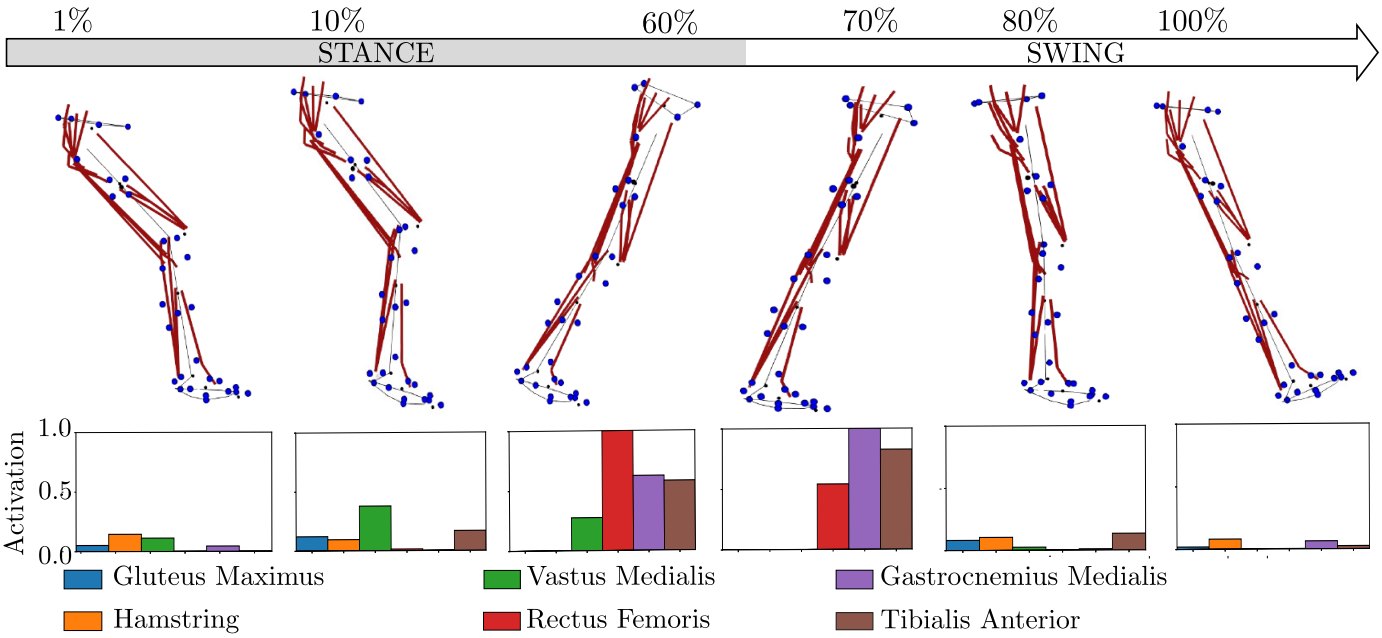

Fig. 7: Snapshots of a walking gait cycle driven by muscles activation with histogram of muscle activations below. The red lines represent muscles lines of action and the blue points depict the tracked markers. The activation of the Gluteus Maximus is the mean of its three parts and the Hamstring is the mean activation of the Semimembranous, Semitendinous and Biceps Femoris.

($\mathbf{a}$) was also added (least-activations) as well as a penalization term on the residual torques ($\boldsymbol{\tau}$):

$$\mathcal{J} = \int_{t=0}^{T} \underbrace{\omega_1(\|\boldsymbol{m} - \boldsymbol{m}^*\|^2)}_{\texttt{TRACK\_MARKERS}} + \underbrace{\omega_2(\|\boldsymbol{\mathcal{F}} - \boldsymbol{\mathcal{F}}^*\|^2)}_{\texttt{TRACK\_FORCES}} + \underbrace{\omega_3(\|\boldsymbol{\mathcal{M}} - \boldsymbol{\mathcal{M}}^*\|^2)}_{\texttt{TRACK\_MOMENTS}} + \underbrace{\omega_4\|\mathbf{a}\|^2}_{\texttt{MIN\_ACTIVATION}} + \underbrace{\|\boldsymbol{\tau}\|^2}_{\texttt{MIN\_TORQUE}} \; dt, \tag{6}$$

where ω_1=1e5, ω_2=0.1, ω_3=0.1, ω_4=10 are weighting factors and T is the the duration of the current phase. Non-slipping (`NON_SLIPPING`) and unilateral contact force (`CONTACT_FORCE`) constraints were added to prevent the foot from slipping and pulling from the ground. In between phases, the use of the `IMPACT` state transition allowed to represent the gain or loss of contact(s) in the dynamics (e.g., swing phase to heel strike [**?**])

Tracking experimental data allowed to reproduce leg motion during the walking cycle (Fig. **??**). The root mean square tracking error on markers trajectories was 27 mm (mean errors on pelvic and foot markers were 7.5 mm and 14.7 mm, respectively). Concerning ground reaction forces tracking, the root mean square error was 4.85 N. During the stance phase, Gluteal muscles and Vastus Medialis were mainly activated during the loading response (10 %) and hamstrings during initial contact (1 %) (Fig. **??**). These results were similar to the characteristic average activity patterns of the lower limb muscles during locomotion described in [**?**]. The transition from stance to swing (60 % - 70 %) was highly actuated by the Rectus Femoris and leg muscles (Gastocnemius Medialis and Tibialis Anterior).

E. Moving Horizon Estimation of Shoulder Elevation

This example is presented to introduce *Bioptim*'s ability to provide real-time estimation of biomechanical variables. The goal was to perform a real-time estimation of dynamically consistent joint kinematics and muscle forces, using a moving horizon estimation (MHE) approach (i.e. an optimization approach that uses a series of measurements observed over time). A shoulder elevation motion was performed with a 4-DoFs ($\mathbf{q}$) arm actuated by 19 Hill-type muscle elements. The control inputs of the model were the muscle activations ($\mathbf{a}$). The MHE implementation consists in splitting the OCP into a succession of smaller one for processing fixed-size subsets of the tracking data moving forward in time. Each time one subproblem is solved, a new measurement is added, the oldest one is discarded and a new subproblem is defined. Due to their similarities, the solution of the previous OCP is a good initial guess to the new one. The dynamical consistency of the final solution is enforced by continuity constraints on the initial state. Each objective function (Eq. **??**) was written as the sum of three terms: tracking reference joint angles ($\mathbf{q}^*$), states and muscle activations regularizations (i.e., least-square criteria):

$$\mathcal{J} = \int_{t}^{t+t_{mhe}} \underbrace{\omega_1(\|\boldsymbol{q} - \boldsymbol{q}^*\|^2)}_{\texttt{TRACK\_STATE}} + \underbrace{\omega_2\|\boldsymbol{q}\|^2}_{\texttt{MIN\_STATE}} + \underbrace{\omega_3\|\boldsymbol{a}\|^2}_{\texttt{MIN\_ACTIVATION}} \; dt, \tag{7}$$

where $\omega_1 = 10^5$, $\omega_2 = 10^2$, $\omega_3 = 10^4$ and t_{mhe} is duration of each sub-problem.

In this example, reference data of an 8 s series of four arm elevations were generated at 100 Hz, by computer simulation. A centered Gaussian noise (mean = 0, std = 0.005 $q^*(t)$) was added to q^*, to simulate experimental-like joints angle

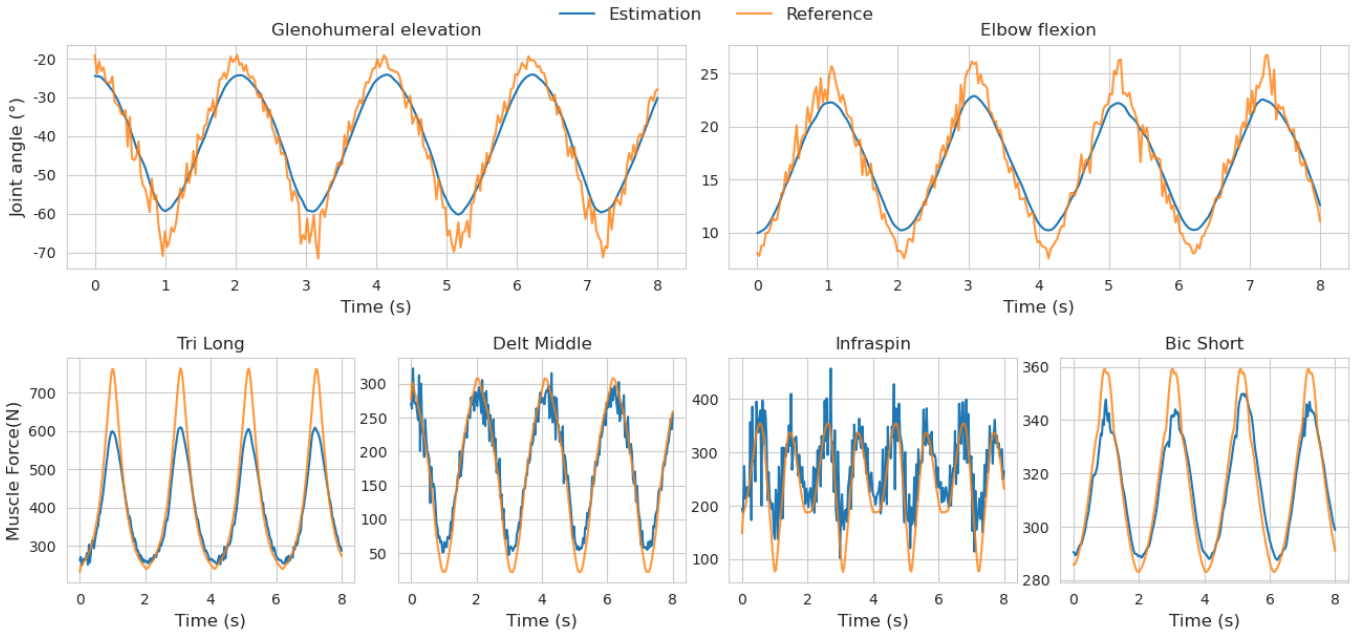

Fig. 8: Time series of real-time (1) estimated joint angles (blue) and noisy reference joint angles (orange) of a cyclic motion (up). (2) estimated muscle forces (blue) and ground truth muscle forces (orange) of a cyclic motion (bottom) (Ex. **??**). Only four muscles with significative action (peaks force $>$ 15 N), on the two selected DoFs, are shown. Muscle abbreviations stand for (from left to right): Triceps Long head, Deltoid Middle, Infraspinatus, Biceps Brachial Short head. (Ex. **??**).

measurements. Using a windows size of 7 nodes (i.e., 210 ms), the estimator ran at about 33 Hz (one in three reference data frame was sent to the estimator to simulate experimental-like conditions), i.e., two and half times faster than standard biofeedback (13 Hz, [**?**]). The MHE was able to forecast the movement kinematics with a root mean square error of 1.3 ± 0.5 ° while provided a realistic estimation of muscle forces close to the ground truth with a root mean square error of 10.6 ± 13.6 N (Fig. **??**).

F. Multiphase vertical jumper

This example was designed to introduce *Bioptim*'s ability to reduce the number of degrees-of-freedom (DoF) of a model via the `mappings`, to account for nonlinear boundaries on the controls, and to solve complex multiphase programs including impacts and free time phases.

We used a full-body model consisting of 3 DoFs at the pelvis (forward and upward translations, tranverse rotation), 1 DoF at each upper limb (shoulder flexion), and 3 DoFs at each lower limb (hip, knee and ankle flexion) for a total of 11 DoFs. For this optimization, the `BidirectionalMapping` was used to symmetrize the left-hand side with the right-hand side, effectively creating a 7-DoFs pseudo-2D model. Since this is a full-body model, the root segment (i.e., the pelvis) was left uncontrolled, reducing the number of control variables to 4, namely the shoulder, hip, knee and ankle flexions.

A total of five phases were used to describe the dynamics of the jump, flight and landing. The first two were push-off phases consisting in one phase with two ground contacts (heel and toe) and a second one with one contact (toe). A contact is described as a point where forces are applied to cancel its acceleration. The third phase was purely aerial, described by a free-fall dynamics. The last two were landing phases, a first one with one contact (toe) and a second one with two contacts (heel and toe). The transitions between phases with addition of contact points were approximated with the build-in inelastic impact `PhaseTransition.IMPACT`, which computes the velocity of the kinematic chain after an impact.

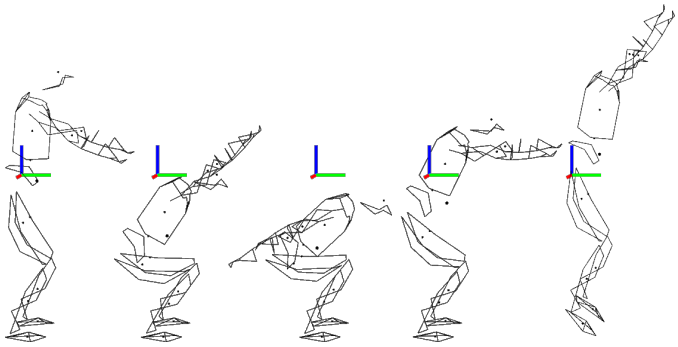

Fig. 9: Snapshots of the push-off phase of a vertical jump (Ex. **??**). The avatar reproduces a human-like jump movement. The first four positions represent the first phase of the optimization (i.e., heel and toe in contact with the floor) and the fifth position depicts the end of the second phase (i.e., only the heel in contact with the floor)

The objective function with the most important weight was

a Mayer objective computed at the end of the push-off phase consisting in maximizing the jump height (h)—represented by the position of the CoM—from the projectile equations. The remaining objective functions were added either to help the solver to converge (regularization) or to converge toward a more human-like solution.

$$\mathcal{J} = \underbrace{\omega_h h}_{\texttt{MIN\_PREDICTED\_COM\_HEIGHT}} + \underbrace{\omega_x \|\mathbf{x}_+(T_5) - \mathbf{x}_+^*\|^2}_{\texttt{TRACK\_STATE}} + \sum_{i=1}^{5} \underbrace{\omega_t (T_i - T_{i-1})}_{\texttt{MIN\_TIME}} + \sum_{i=2}^{4} \int_{t=T_i}^{T_{i+1}} \underbrace{\omega_{sd} \left\| \frac{d\dot{\mathbf{q}}}{dt} \right\|^2}_{\texttt{MIN\_STATE\_DERIVATIVE}} dt \quad (8)$$

where $T_0 = 0$, T_i with $i \in [1, 2, 3, 4, 5]$ are the final times of the ith phase respectively; ω_h is the weight of the maximization of the CoM height and is defined negative (-100) to effectively maximize this term objective function; ω_t, ω_{sd} and ω_x are weights equal to 0.1, 0.1, and 1, respectively; $\dot{\mathbf{q}}$ is the generalized velocities part of the state vector $\mathbf{x}$; and $\mathbf{x}_+$ is the state vector excluding the translations of the root segment. The $\mathbf{x}_+^*$ corresponds to a reference static position of the avatar with its knee slightly flexed and its arms horizontally raised.

Several constraints are necessary to describe a realistic jump. Joint angles were bounded to human-like limits. The first node of the first phase was enforced to be equal to $\mathbf{x}^*$ (i.e., including the translations of the root segment to be at the origin). Joint velocities were arbitrarily bounded to $[-10\pi; 10\pi]$ rad/s. Joint torques were bounded with nonlinear torque/angle/velocity relashionships measured on a high level athlete using an isokinetic dynamometer (Fig. **??**). Non slipping (`NON_SLIPPING`) and unilateral (`CONTACT_FORCE`) contact force constraints were added to prevent the contact points from slipping and pulling on the ground. Finally, some constraints were added to help convergence. First, during the push-off and landing phases, the heels had to remain over the floor. Then, the center of mass velocity had to point upward when leaving the floor and at the same instant, the arms had to be frontward.

To speed-up the solving of the problem with *ipopt*, the problem was first approximately solved using the BFGS hessian approximation option for 200 iterations maximum. Then, starting from this first solution, the problem was re-optimized, with exact-hessian computations for up to 1000 iterations. The optimized solution was obtained in 148 iterations of the exact-hessian optimization for a total optimization time of $1780\,\mathrm{s}$ ($\approx 30\,\mathrm{min}$), resulting in a $1.28\,\mathrm{m}$ jump height. The optimized time for the phases 1 to 5 were $0.70, 0.05, 0.99, 0.36, 0.21$ seconds, respectively. The solution reproduced a human proximo-distal strategy (Fig **??**), i.e., activating large segments first (for instance the torso) and sequentially adding more distal segments, consequently ending up with the ankles.

IV. Discussion

The purpose of *bioptim* is to solve a variety of biomechanical OCPs with minimal user effort and high

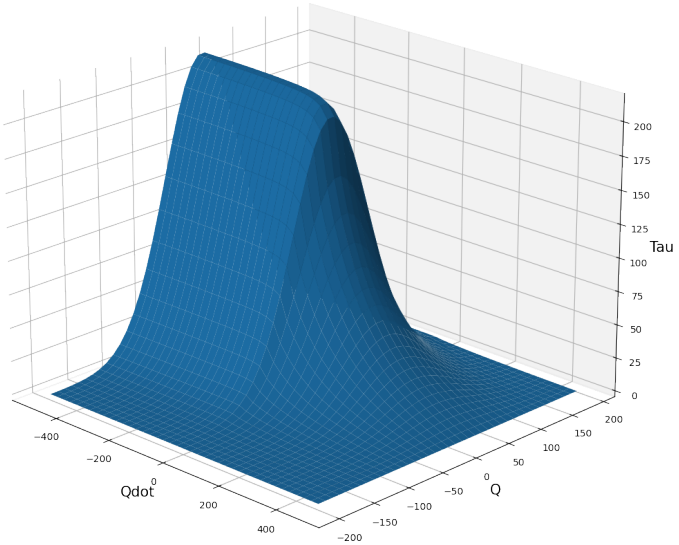

Fig. 10: Surface representing the nonlinear constraint from the torque/position/velocity relashionship of the hip flexion

performances in terms of computational time. The main features illustrated by the six provided examples are (Tab. **??**):

- the possibility to use torque- or muscle-driven models (and their combinations);
- a variety of ready-to-use cost functions, constraints and dynamics (with and without contacts)...
- ... easily customizable in Python when required by the user;
- the possibility to solve advanced OCPs (possibly multiphase) in a few seconds or minutes, that previously took us hours;
- the interface with two different NLP solvers

In the following, several aspects of *bioptim* are discussed.

A. Direct multiple shooting-based

While the debate remains about the performances of direct collocations versus direct multiple shooting (DMS) [**?**], the development of *Bioptim* was oriented toward the latter, because: *i)* it allows to select effortlessly an arbitrary accuracy for the integration (e.g., order and numbers of RK steps); *ii)* it allows to use DMS-based fast NLP solvers such as *ACADOS*. Concerning the integration, either internally or via *ACADOS*, several schemes are implemented in *bioptim* (RK4, RK8, IRK). While IRK showed better convergence in our experience with hard problems in *ACADOS*, RK4 showed to be a good speed/robustness tradeoff in most of the cases. In contrast to what is claimed in [REF], DMS is not a limitation to the performances (cost value and time to convergence), since, in our experience, the performances of *Bioptim* often outperform state-of-the-art results.

B. Automatic differentiation

One of the reasons explaining the performances of *Bioptim* is the rewriting of the core software, *RBDL* and *Biorbd* implementing the dynamics, into *CasADi* symbolics to automatically provide the exact Jacobians and Hessians of

TABLE I: Overview of computational results for the different OCPs cases and links to detailed implementations. The single shooting state trajectory is obtained by forwardly integrating the initial state with the optimized control inputs during 1 second. The single shooting error is computed as the mean RMSE between the optimized state vector and the single shooting one at 1 second.

		III-A Activation-driven pointing		Ex# 2		Ex# 3	
Setup	# states $\boldsymbol{x}(t)$	4		–	–	–	–
	# control $\boldsymbol{u}(t)$	6		–	–	–	–
	# shooting nodes	50		–	–	–	–
	OCP duration (s)	2		–	–	–	–
		Ipopt	*ACADOS*	*Ipopt*	*ACADOS*	*Ipopt*	*ACADOS*
Solve	# NLP iterations	47	19	–	–	–	–
	Optimized cost	20.8	23.2	–	–	–	–
	Time to convergence (s)	22.3	0.45	–	–	–	–
	Single shooting error	$< 10^{-7}$	$< 10^{-13}$	–	–	–	–

the resulting NLP. The gain in accuracy for the calculation of derivatives leads to shorter convergence times (due to much less iterations) and to optimal solutions reached with lower tolerances. This last aspect must be emphasized for complex motions (fast, highly dynamics ones), because, for instance when using *Ipopt*, an optimal solution obtained with a convergence criterion of 10^{-2} is very unlikely to be dynamically sound; i.e., it would diverge when forwardly integrating the controls in a single-shooting manner. A lower tolerance (10^{-6}. 10^{-8}), which is only reachable with exact derivatives—for most of OCPs in biomechanics—, is expected to lead to better forward dynamics results.

C. Python based, but fast!

Bioptim was thought as an interface, and was therefore written in Python to allow the user to easily combine existing cost functions or constraints and self-implemented ones, to switch from one solver to another, etc. We believe this feature to be of importance given that the biomechanics community is mainly composed of software users rather than developers. Therefore, providing a custom interface in Python rather than in C++ [MOCO], was a driving objective of our work to facilitate a rapid appropriation by the community. Since flexibility and ease-of-use should not compromise the performances, the integration is multi-threaded and all the inside computations are expressed as C++ *CasADi* graphs, interfaced with C++ NLP solvers. These graphs can either be built in `casadi.MX()` or `casadi.SX()`. The latter requires more RAM for building the problem but is faster to solve. While both may be used with *Ipopt*, *ACADOS* is only compatible with `casadi.SX()`. By leveraging the speed of `casadi.SX()` graphs, we were able to estimate muscle forces in real time using *ACADOS* on a standard laptop (Ex. **??**). For a more in-depth analysis of the real-time estimation capabilities of *Bioptim*, see [**?**].
Alongside with the 3D visualizer *Bioviz* that animates the solution, *Bioptim* proposes a series of online-generated figures, inspired by the real-time graphics from *Muscod-II* REF, to visualize the optimized variables at each iteration of the solver. This is made with minimal computational cost thanks to the multiprocessing Python toolbox. Our implementation leverages the *Python pickle* library for easily saving and loading OCPs for, e.g., post-processing analysis. Finally, every layer (integration, optimization, visualization) of *Bioptim* is optimized to be flexible and fast.

D. Fast vs robust NLP solvers

Fast solvers, such as *ACADOS*, offer the opportunity to use multi-start approaches on complex problems, to circumvent the obstacle of local minima [**?**], [**?**]. It also allows to get meaningful initial solutions from simpler problems, for guiding the resolution of the harder problems. On the other hand, robust solvers, such as *Ipopt*, are convenient when the user lacks information about the sought solutions and thus cannot guide the solver through a good initial guess. For biomechanics applications, the complementary characteristics of the interfaced solvers is a really useful tool. Moreover, *Bioptim*'s full compatibility with *CasADi* provides the opportunity to use any solver already interfaced with it, including third-party software such as *SNOPT* REF, *WORHP* REFand *KNITRO* REF(not tested yet).

E. Multiphase

Biomechanics studies often face changing dynamics or objective functions due to the loss or gain of contacts or time-varying biomechanical tasks. When tracking such a motion or trying to predict it, these changes translate into multiphase OCP. This is one of the reported drawbacks of *OpenSim Moco* REF, which does not provide this feature yet. *Bioptim*, however, is able to handle multiphase OCPs, although they can currently only be solved with *Ipopt* (see Exs. III-D and **??**).

F. From constraints to objectives: easy problem relaxation

As stated in Sec. II.B, there exists a correspondence between most of the pre-implemented `Constraints` and `Objective functions`. This is intended to allow for easy relaxation when the problem is reluctant to converge. For instance, when a biomechanical task requires the final configuration of the model to be enforced (reaching, cyclic motions, sports, etc.), one should first use a `Constraint` (e.g., `TRACK_STATE`). If the convergence is challenging, just turning this constraint into its namesake Mayer `Objective function`, with an heavy weight, should help the solver.

G. Limitations

Bioptim is already a mature solution for solving biomechanical OCP. However some limitations should be raised. First, it is based on *Biorbd* which is not as advanced as *OpenSim* or *AnyBody* (AnyBody Technology) in terms of biomechanical features and audience. Nevertheless, *Biorbd* is actively maintained, fast and *CasADi*-compatible for automatic differentiation. The variety of proposed examples highlighted simple to advanced models. Even if defining a new model was made straightforward thanks to the `.bioMod` file format, *biorbd* does not include a GUI for building models. Some Opensim models can be translated into `.bioMod` [LINK] but *Biorbd* does not yet support multiple wrapping objects, non-orthogonal DoFs between bodies, compliant contact force models (e.g. [SmoothSphereHalfSpaceForce [52]]) or muscle-tendon equilibrium. As seen in REF, wrapping objects are rare due to computational cost and required optimization when a line of action is in contact with more than one object, which compromises automatic differentiation. Via points [REF] and pre-processed moment arms REF(to be expressed as polynomial functions of crossed DoFs) are often preferred.

H. Future directions

Bioptim v1.0 was released in January 2021, with all the features presented in this communication. Some improvements are expected in a near future. First, a graphical model builder is planned in *Biorbd*, to easily generate *.bioMod* files. Also, models of muscular fatigue are to be included in *Bioptim*, to predict adapted motor strategies for long or demanding motions. The formulation of moving horizon schemes (MHE, Nonlinear Model Predictive Control) will be pre-implemented, with efficient warm-starting heuristics, to facilitate their use. The implementation of muscle-tendon equilibrium is planned for fast movements or those with large ranges of motions. It will require an additional optimization step to achieve the equilibrium as done in CEINMS [REF] or the addition of muscle lengths as state variables, with explicit constraints of XXXX like in [REF]. Moreover, an effort will be made to extend the compatibility of *ACADOS* with all the features of *Bioptim* (multiphase, nonlinear constraints, etc.). Finally, we plan to add an inverse optimal control module to *Bioptim*.

ACKNOWLEDGMENT

This study and the *Biorbd* library development was partly funded by a scholarship of the Vanier program (BM), the Canada First Research Excellence Fund via the TransMedTech Institute (FB) and the NSERC Discovey Programme (MB). *Bioptim* acts as a catalyst in our group and several students contributed to this library. Thank you to Paul Wegiel, Théophile Gousselot, Ariane Dang, André Venne and Kilpéric Nouvellet.