Musculoskeletal Optimal Control Tutorial Problems

Nathan Sauder, Anil Rao, and B.J. Fregly
University of Florida

Introduction

This document provides a tutorial for two musculoskeletal optimal control example problems that can be solved using GPOPS-II. Choose one of the two problems to complete during the workshop, and explore the second problem after the workshop at your leisure. Working versions of both example problems are provided for reference. Synthetic data (e.g., marker trajectories, ground reactions, and/or joint torques) for both example problems were generated from a separate optimal control problem described below. All problems utilize a simple two degree-of-freedom (DOF) squatting model with relevant equations developed in Autolev.

The layout of this tutorial document is as follows:

Problem Overviews

1. Data Generation: Full Problem

This problem was designed as a combination of the two example problems to generate the synthetic data necessary for each example problem. A simplified planar "toy" model of a person performing a squatting motion is used (Fig. 1), and the sum of squares of muscle excitations is minimized.

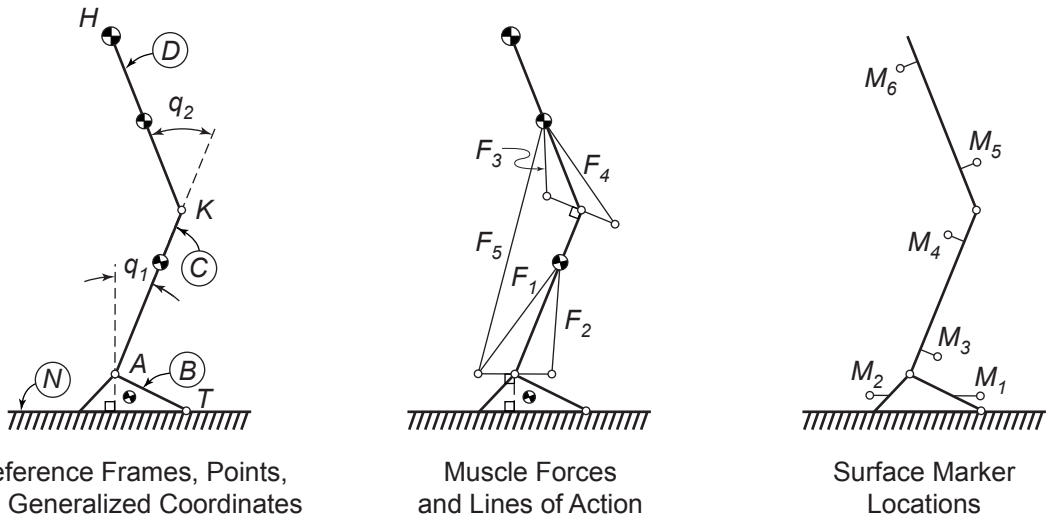

Fig. 1: *Schematic of the original squatting model used for benchmark problem development.*

The model is controlled by five fictitious muscles and contains four reference frames. However, reference frame *B* (representing the foot) is fixed in reference frame *N* (the Newtonian frame) for this problem. Reference frames *C* and *D* represent the shank and thigh, respectively. Points *A* and *K* represent the ankle and knee joint locations, respectively, and generalized coordinates q_1 and q_2 represent the ankle angle and knee angle, respectively. Muscle 1 is a uniarticular soleus-like muscle, muscle 2 a uniarticular tibialis anterior-like muscle, muscle 3 a uniarticular biceps femoris shorthead-like muscle, muscle 4 a uniarticular vastus-like muscle, and muscle 5 a biarticular gastrocnemius-like muscle. The insertions of muscles 1, 2, and 5 are fixed in the foot segment *B*, and those of muscles 3 and 4 are fixed in the shank segment *C*. All muscle origins are at the center of mass of either the shank or the thigh segment. All of the muscles are considered to be rigid tendon Hill-type models with force-length-velocity properties and possessing activation dynamics. The upper body is modeled as a single point mass at the hip *H*.

The optimal control problem is formulated as shown below using two phases, where the first phase takes the model from a static upright posture to a specified deep squatting posture in a specified amount of time, and the second phase takes the model from the specified deep squatting posture back to the original static upright posture in the same amount of time:

States –
[q_1, q_2, u_1, u_2, a_1, a_2, a_3, a_4, a_5] where q_1 and q_2 are generalized coordinates representing the ankle and knee angles, respectively, u_1 and u_2 are their time derivatives and the associated generalized speeds, and $a_{1\text{-}5}$ are the activations of muscles 1-5.

Controls –
[e_1, e_2, e_3, e_4, e_5] where $e_{1\text{-}5}$ represents the excitations of muscle 1-5, which are the inputs to activation dynamics.

Cost Function –
The cost is the integral of the sum of squares of the five muscle excitations.

Constraints –
Dynamic: $da_i / dt = f_i$ (i=1,...,5), $du_j / dt = g_j$ (j=1,2) (explicit dynamics)
Bound: $0 \le a_i \le 1$, $0 \le e_i \le 1$
Initial: $t = 0$, $q_1 = 0$, $q_2 = 0$, $u_1 = 0$, $u_2 = 0$
Phase 1 Terminal: $t = 0.5$, $q_1 = 30°$, $q_2 = 60°$, $u_1 = 0$, $u_2 = 0$
Phase 2 Terminal; $t = 1$, $q_1 = 0$, $q_2 = 0$, $u_1 = 0$, $u_2 = 0$
Event: time and state at end of phase 1 = time and state at beginning of phase 2

The solution to this problem was used to calculate inputs to (or validation quantities for) the two benchmark problems based on an expanded version of the two DOF squatting model where the foot is no longer constrained to remain fixed to the floor (Fig. 2). Elimination of this constraint necessitates the addition of three generalized coordinates q_3 through q_5 defining the position of the hip and the orientation of the thigh segment in reference frame *N*.

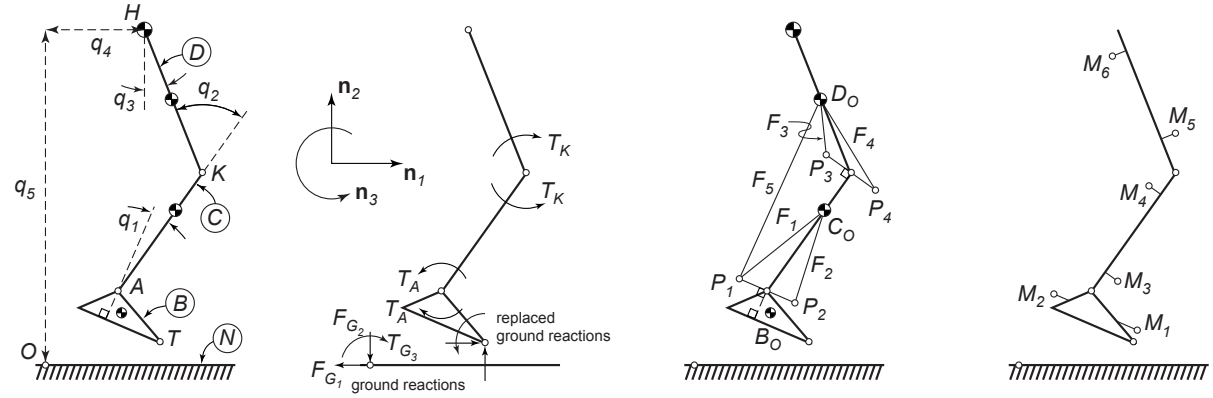

Fig. 2: *Schematic of expanded squatting model used for developing the benchmark problems. Based on the solution from the full problem, this model was used to calculate predicted net joint torques, ground reaction forces and torque, five generalized coordinate trajectories, and surface marker trajectories.*

The following quantities were calculated for benchmark problem use:

- Generalized coordinate trajectories q_1 through q_5 and associated generalized speed trajectories u_1 through u_5, which are the first time derivatives of the corresponding generalized coordinates, as a function of time
- Muscle force trajectories F_1 through F_5 as a function of time
- Moment arm trajectories for all muscles about their spanned joints as a function of time

- Net ankle and knee torque trajectories T_A and T_K as a function of time
- Muscle-tendon length and velocity trajectories for all muscles as a function of time
- Ground reaction force and torque trajectories F_{G1}, F_{G2}, and T_{G3} as a function of time
- *x* and *y* trajectories of six surface markers M_1 through M_6 as a function of time

2. First Benchmark Problem: Dynamic Muscle Force Estimation

This problem was designed as a benchmark for dynamic muscle force estimation where muscle forces predicted using "fast" activation dynamics and compliant tendon models with high stiffness can be compared to a global solution from static optimization using rigid tendon models. Since muscle-tendon lengths and velocities and muscle moment arms are taken from the results of the full problem, skeletal dynamics is eliminated from this problem. The sum of squares of muscle excitations is minimized while requiring that the net torque at the ankle and knee be matched.

Unlike the full problem, the five muscles in this problem possess both activation dynamics and contraction dynamics (i.e., compliant tendon models). Tendons are modelled as linear springs possessing a common stiffness for all muscles.

The optimal control problem is formulated as follows:

States –
[a_1, a_2, a_3, a_4, a_5, $\tilde{\ell}^M_1$, $\tilde{\ell}^M_2$, $\tilde{\ell}^M_3$, $\tilde{\ell}^M_4$, $\tilde{\ell}^M_5$] where $a_{1\text{-}5}$ represent muscle activations and $\tilde{\ell}^M_{1-5}$ represent normalized muscle lengths.

Controls –
[e_1, e_2, e_3, e_4, e_5, $\tilde{v}^M_1$, $\tilde{v}^M_2$, $\tilde{v}^M_3$, $\tilde{v}^M_4$, $\tilde{v}^M_5$, T_{Ares}, T_{Kres} where $e_{1\text{-}5}$ represent muscle excitations, $\tilde{v}^M_{1-5}$ represent normalized muscle velocities, and T_{Ares} and T_{Kres} are residual ankle and knee torques.

Cost function –
The cost is the integral of the weighted sum of squares of muscle excitations and sum of squares of residual actuator torques.

Constraints –
Dynamic: $da_i / dt = f_i$ (j=1,...,5) (explicit dynamics)
Bound: 0 ≤ a ≤ 1, 0 ≤ e ≤ 1, $-1 \le \tilde{v}^M \le 1$
Initial: t = 0
Terminal: t = 1
Path: $f_j(d\tilde{\ell}^M_j / dt) = 0$ (j=1,...,5) (implicit dynamics), difference between predicted and reference joint torques = 0 at all time points

3. Second Benchmark Problem: Dynamically Consistent Inverse Kinematics

This problem was designed as a benchmark for inverse kinematic analysis while maintaining dynamic consistency. Joint torques are controlled to minimize the errors between model and reference marker positions while applying the reference ground reactions.

The expanded model used for this problem (Fig. 2) contains the same segments as the full problem except that the foot (body *B*) is no longer fixed to the ground. This model possesses

five DOFs. Generalized coordinates q_1 and q_2 still define the ankle and knee angles, respectively. In addition, generalized coordinate q_3 defines the thigh segment angle with respect to a vertical axis fixed in reference frame *N*, and generalized coordinates q_4 and q_5 define the horizontal and vertical position, respectively, of point *H* relative to point *O* fixed in *N*. The ground reaction forces and torque are calculated from the output of the full problem. The ground reactions are replaced to the toes point *T* similar to how experimentally measured ground reactions would be handled. Muscles are not used for this problem because joint torques are controlled directly. Thus, only skeletal dynamics are present in this problem.

The optimal control problem is formulated as follows:

States –
$[q_1, q_2, q_3, q_4, q_5, u_1, u_2, u_3, u_4, u_5]$ where $q_{1\text{-}5}$ are generalized coordinates and $u_{1\text{-}5}$ are the corresponding generalized speeds defined as the first time derivatives of $q_{1\text{-}5}$.

Controls –
$[T_A, T_K]$ where T_A is the ankle torque and T_K is the knee torque.

Cost function –
The cost is the sum of squares of marker errors.

Constraints –
Initial: $t = 0$
Terminal: $t = 1$

Note that for both benchmark poblems, completed fully functional versions of all Matlab files are also provided. These files are indicated by the `_completed` addition to the file name.

Problem Instructions

Choose one of the two benchmark problems below to complete during the workshop. If you are interested, you can explore the second benchmark problem on your own after the workshop.

1. First Benchmark Problem: Dynamic Muscle Force Estimation

a. Main File

Open the file `muscleForce_main.m` in the Matlab editor. This is a script that will setup and run the optimization problem.

Scaling:
First, we will set up scale factors for the variables in our problem using a canonical transformation of units. This approach creates a new system of units with the goal of making the variables and their derivatives all of a similar magnitude. Scaling is extremely important in optimizations as it can have a huge influence on problem convergence.

Create a structure of scale factors for time, mass, length, velocity, angular velocity, acceleration, angular acceleration, force, and torque. Only the scale factors for time, mass, and length will be manually assigned. The other scale factors are calculated so as to keep the units consistent.

For example, the scale factor for velocity is equal to the length scale divided by the time scale. Set the time scale factor to 30 (i.e. 1 second = 30 new time units) and leave those of mass and length at 1.

It is often useful to be able to scale the cost value separately from the rest of the variables. Create a cost scale factor equal to 10 divided by the time scale factor.

```
scale.time = 30;
scale.mass = 1;
scale.length = 1;
scale.vel = scale.length/scale.time;
scale.accel = scale.vel/scale.time;
scale.force = scale.mass*scale.accel;
scale.torque = scale.force*scale.length;
scale.angVel = 1/scale.time;
scale.angAccel = scale.angVel/scale.time;
scale.inertia = scale.torque/scale.angAccel;

scale.cost = 1e1/scale.time;
```

Load Data:
Load the file `muscleForce_data.mat` (make sure it is on your Matlab path or in your current folder). This file contains an `auxdata` structure with various constant parameters as well as splines for time varying reference data such as muscle moment arms. It may be useful to take a minute to look through everything contained in the file.

Add fields to auxdata for activation and deactivation time constants and for tendon stiffness. Start with an activation time constant of 0.01, deactivation time constant of 0.04, and tendon stiffness of 10^9. Also add the `scale` structure you created to `auxdata` as a new field.

```
load('muscleForce_data.mat');
auxdata.scale = scale;

auxdata.tact = 0.01;
auxdata.tdeact = 0.04;
auxdata.kT = 1e9;
```

Bounds:
GPOPS must be given appropriate bounds for all variables. Define row arrays for maximum and minimum values of the activations states ($0 \le a \le 1$), the normalized muscle length states ($0 \le \tilde{\ell}^M \le 3$), muscle excitations ($0 \le e \le 1$), normalized muscle velocities ($-1 \le \tilde{v}^M \le 1$), and residual actuators (-200 ≤ `resAct` ≤ 200). Also constrain the path (explained below in the continuous function section) to have seven columns of zeros for both minimum and maximum values. This path constraint will require joint torque errors and muscle and tendon force differences to be zero.

```
a_min = zeros(1,5);
a_max = ones(1,5);
lMtilda_min = zeros(1,5);
lMtilda_max = 3*ones(1,5);
```

```
e_min = zeros(1,5);
e_max = ones(1,5);
vMtilda_min = -ones(1,5);
vMtilda_max = ones(1,5);
resAct_min = -200*ones(1,2);
resAct_max = 200*ones(1,2);

path_min = zeros(1,7);
path_max = zeros(1,7);
```

Define variables for the initial time (`t0 = 0`) and final time (`tf = 1`). Also define row arrays for the initial and final activation and normalized muscle lengths from the variables `a0_ref`, `af_ref`, `lMtilda0_ref`, and `lMtildaf_ref` in the `auxdata` structure. In general, the initial and final activations and muscle lengths would be free rather than fixed, however it is difficult for the optimization to determine initial and final states in this formulation. Therefore we will be fixing the initial and final state for this tutorial. After completing this problem with the initial and final state fixed, it may be instructive to go back and allow the bounds on the initial and final state to be free (i.e., equal to the bounds of the state). If this same problem was being solved using experimental data rather than artificially generated data, extra time would be included before and after the region of interest to correct for this issue.

```
t0 = 0;
tf = 1;
a0_min = auxdata.a0_ref;
a0_max = auxdata.a0_ref;
af_min = auxdata.af_ref;
af_max = auxdata.af_ref;
lMtilda0_min = auxdata.lMtilda0_ref;
lMtilda0_max = auxdata.lMtilda0_ref;
lMtildaf_min = auxdata.lMtildaf_ref;
lMtildaf_max = auxdata.lMtildaf_ref;
```

Now we will create a structure called `bounds` with a field called `phase`. This is a single phase problem, but GPOPS still requires a field for phases. `Phase` will have fields for `state`, `control`, `integral`, `path`, `initialtime`, `finaltime`, `initialstate`, and `finalstate`, and each of these will have fields called `lower` and `upper`. Assemble appropriate row arrays into the upper and lower bound structures. When assembling the variables into their corresponding structures, multiply each one by the correct scale factor. For example, the activation state is already a normalized quantity and therefore does not get further scaled, but the initial and final time must be multiplied by the time scale factor. The residual actuator bounds should also be scaled (by the torque scale factor).

```
bounds.phase.initialtime.lower = t0*scale.time;
bounds.phase.initialtime.upper = t0*scale.time;
bounds.phase.finaltime.lower = tf*scale.time;
bounds.phase.finaltime.upper = tf*scale.time;
bounds.phase.initialstate.lower = [a0_min,lMtilda_min];
bounds.phase.initialstate.upper = [a0_max,lMtilda_max];
bounds.phase.state.lower = [a_min,lMtilda_min];
bounds.phase.state.upper = [a_max,lMtilda_max];
```

```
bounds.phase.finalstate.lower = [af_min,lMtildaf_min];
bounds.phase.finalstate.upper = [af_max,lMtildaf_max];
bounds.phase.control.lower = [e_min,vMtilda_min,resAct_min*scale.torque];
bounds.phase.control.upper = [e_max,vMtilda_max,resAct_max*scale.torque];
bounds.phase.integral.lower = 0;
bounds.phase.integral.upper = 1e5;
bounds.phase.path.lower = path_min;
bounds.phase.path.upper = path_max;
```

Initial Guess:
An initial guess may also be provided to GPOPS. We will be providing a simple guess with only initial and final values for the time, state, and controls.

- Time guess: initial and final time multiplied by the time scale factor
- Activations guess: minimum activations
- Normalized muscle length guess: ones
- Control guess: zeros
- Integral guess: 0.1 multiplied by the cost scale factor

Assemble these quantities into a structure called `guess.phase`. Remember that each field in `guess.phase` except for `integral` (i.e. `time`, `state`, `control`) should be a 2 x n array where n is number of variables for that field. The guess provided for the integral (which will be the cost value for us) is always 1 x n.

```
guess.phase.time    = [t0*scale.time; tf*scale.time];
guess.phase.state   = [a_min,ones(1,5); a_min,ones(1,5)];
guess.phase.control = zeros(2,12);
guess.phase.integral = 0.1*scale.cost;
```

Problem Structure:
Now we will setup the actual problem structure to be given to GPOPS. Look over it and note the options used and how they are assembled into the `setup` structure.

```
setup.name = 'MuscleOptimization_rigidTendon';
setup.functions.continuous = @muscleForce_continuous;
setup.functions.endpoint = @muscleForce_endpoint;
setup.auxdata = auxdata;
setup.bounds = bounds;
setup.guess = guess;

setup.nlp.solver = 'ipopt';
setup.nlp.ipoptoptions.tolerance = 1e-7;
setup.derivatives.supplier = 'sparseCD';
setup.derivatives.derivativelevel = 'second';
setup.derivatives.dependencies = 'sparse';

meshphase.colpoints = 4*ones(1,20);
meshphase.fraction = 1/20*ones(1,20);
setup.mesh.phase = meshphase;
setup.mesh.method = 'hp-PattersonRao';
setup.mesh.tolerance = 1e-5;
setup.mesh.maxiterations = 10;
```

```
setup.mesh.colpointsmin = 4;
setup.mesh.colpointsmax = 10;

setup.method = 'RPM-Integration';
```

Run GPOPS:
Call the `gpops2` function with the `setup` structure as the input, and create a variable for the output. Save the output to your computer so that it can be reloaded later.

```
tic
output = gpops2(setup);
toc

save('muscleForce_output','output');
```

Post Analysis:
A post hoc analysis function has been provided to graph the output from GPOPs and a reference solution. Run the function `muscleForce_postAnalysis` with the inputs of `auxdata` and the output from GPOPS2. If any of the variable names in the `auxdata` structure are different from those presented in this document, then this function will need to be modified to reflect the new variable names.

```
muscleForce_postAnalysis(auxdata,output);
```

This is the end of the main file.

b. Continuous Function

Open the file `muscleForce_continuous.m` in the Matlab editor. This is the continuous function which will be called by GPOPS.

Unpack Auxdata:
First we will "unpack" the `auxdata` structure so that we can access the variables contained more easily. Unpack `scale`, `spline`, `tact`, `tdeact`, `muscParams`, `FaCoefs`, `FvCoefs`, `FpCoefs`, and `kT` (tendon stiffness).

```
auxdata = input.auxdata;
scale = auxdata.scale;
spline = auxdata.spline;

tact = auxdata.tact;
tdeact = auxdata.tdeact;
muscParams = auxdata.muscParams;
FaCoefs = auxdata.FaCoefs;
FvCoefs = auxdata.FvCoefs;
FpCoefs = auxdata.FpCoefs;
kT = auxdata.kT;
```

Unscale States and Controls:
We have given scaled versions of the states and controls to GPOPS, but it will be easier to work with these variables inside the continuous function if we unscale them first. We will later rescale the output of the continuous function so that GPOPS is working with scaled variables. Unpack and unscale (by dividing by the correct scale factor) time, activation, normalized muscle length, excitation, normalized muscle velocity, and residual actuators.

```
t = input.phase.time/scale.time;
a = input.phase.state(:,1:5);
lMtilda = input.phase.state(:,6:10);
e = input.phase.control(:,1:5);
vMtilda = input.phase.control(:,6:10);
resAct = input.phase.control(:,11:12)/scale.torque;
```

Sample Splines:
Now sample the splines for the muscle moment arms, the reference joint torques, and the muscle tendon lengths using the `ppval` command with the unscaled time vector.

```
R1A = ppval(spline.R1A,t);
R2A = ppval(spline.R2A,t);
R5A = ppval(spline.R5A,t);
R3K = ppval(spline.R3K,t);
R4K = ppval(spline.R4K,t);
R5K = ppval(spline.R5K,t);

TA_ref = ppval(spline.TA_ref,t);
TK_ref = ppval(spline.TK_ref,t);

lMT = ppval(spline.lMT_ref,t)';
```

Activation Dynamics:
Get the derivatives of the muscle activations by running the provided `activationDynamics` function. The inputs for this function are the excitations, activations, activation time constants, and deactivation time constants and the output is an array of the derivatives of muscle activations.

```
ad = activationDynamics(e,a,tact,tdeact);
```

Contraction Dynamics:
For this formulation of the muscle optimization problem the contraction dynamics are actually in the form of a constraint rather than simply being integrated by GPOPS. This is accomplished by constraining the projection of muscle force in the direction of the tendon to be equal to a calculated tendon force. The result of this constraint is that the muscle lengths must be correct in order to get equal tendon and muscle forces. Use the `forceEquilibrium` function provided to get the force error and the tendon force. The inputs to `forceEquilibrium` are activations, normalized muscle length, normalized muscle velocity, muscle tendon length, muscle parameters, FaCoefs, FvCoefs, FpCoefs, and tendon stiffness.

```
[Ferror,FT] =
```

```
forceEquilibrium(a,lMtilda,vMtilda,lMT,muscParams,FaCoefs,FvCoefs, ...
FpCoefs,kT);
```

Calculate Torques:
Each joint torque is calculated by summing the muscle moment arms multiplied by the corresponding tendon forces. For the moment arms provided, the number indicates the muscle it pertains to, and the letter *A* or *K* indicates whether it is for the ankle or knee moment. For the ankle, muscle 2 should contribute a negative moment, and for the knee muscles 3 and 5 should contribute negative moments. Since all moment arms provided are positive, these moment arms must be multiplied by -1 in the torque summation.

```
TA = resAct(:,1) + FT(:,1).*R1A - FT(:,2).*R2A + FT(:,5).*R5A;
TK = resAct(:,2) + -FT(:,3).*R3K + FT(:,4).*R4K - FT(:,5).*R5K;

torqueError = [TA - TA_ref,TK - TK_ref];
```

Now calculate the torque error by subtracting the torques you just calculated from the reference torques. Assemble these errors into an array with the ankle torque error in column one, and the knee torque error in column two.

Output:
The output structure for the continuous function must, in this case, include the dynamics, the integrand, and the path constraints. The `dynamics` field contains an array of the derivatives of the states at each time point, the `integrand` contains an array of the values of the curves to be integrated at each time point, and the `path` contains an array of the values of the path constraints at each time point. We must rescale all of these quantities such that GPOPS is working with the scaled values of them.

First, construct the array for the dynamics. The derivative of the normalized muscle length is equal to the normalized muscle velocity multiplied by the maximum muscle velocity divided by the optimal muscle fiber length ($d\tilde{\ell}^M / dt = \tilde{v}^M \cdot v^M_{\max} / \ell^M_o$). To calculate this expression easily for all muscles, we will create matrices for maximum muscle velocity and optimal fiber length, each with size n by 5, where n is the number of time points. The optimal fiber length is the second row in `auxdata.muscParams`, and the maximum muscle velocity is the fifth row. Create variables for the matrix of duplicate optimal fiber length and maximum muscle velocity and calculate the muscle length derivative.

```
npts = length(t);
lMo = repmat(muscParams(2,:),npts,1);
vMmax = repmat(muscParams(5,:),npts,1);
lMtildad = vMtilda.*vMmax./lMo;
```

The derivative of the muscle activations has already been calculated by the `activationDynamics` function. In order to be properly scaled, both the activation and normalized muscle length derivatives must be divided by the time scale factor.

The integrand will be the sum of the excitations squared (for each time point) plus 0.1 times the sum of the residual actuators squared (also for each time point). Thus the integrand array

should be of size one by number of time points. Multiply the whole integrand array by the cost scale factor.

The path field will be an array of the joint torque error and the muscle force error. Both will need to be multiplied by the appropriate scale factor.

```
output.dynamics = [ad/scale.time,lMtildad/scale.time];
output.integrand = (sum(e.^2,2) + 1e-1*sum((resAct).^2,2))*scale.cost;
output.path = [torqueError*scale.torque,Ferror*scale.force];
```

This is the end of the continuous function.

c. Endpoint Function

Open the file `muscleForce_endpoint.m` in the Matlab editor. This is the endpoint function that will be called by GPOPS.

The endpoint function for this problem is very simple. All we need to do for the endpoint function is set the field `output.objective` equal to the field `input.phase.integral`.

```
output.objective = input.phase.integral;
```

Now that the whole problem is constructed, run the main file. After the problem is working, try changing the bounds on the initial and final activations and muscle lengths such that initial and final state have the same bounds as the rest of the state. What do you notice occurring near the beginning and end?

Change the scaling factors to the problem and observe the result. Does the problem still converge if the time scale factor is changed to one? What about if the mass scale factor is changed to 100? Prior to running the problem with new scaling, change the `maxiterations` option in the `setup.mesh` structure to be zero rather than ten so that you don't have to wait for the mesh to iterate ten time on a problem that may not ever converge.

Change the tendon stiffness value to 10^6. Does the solution change much?

2. Second Benchmark Problem: Dynamically Consistent Inverse Kinematics

a. Main File

Open the file `IK_main.m` in the Matlab editor. This is a script that will setup and run the optimization problem.

Scaling:
First, we will set up scale factors for the variables in our problem using a canonical transformation of units. This approach creates a new system of units with the goal of making the variables and their derivatives all of a similar magnitude. Scaling is extremely important in optimizations as it can have a huge influence on problem convergence.

Create a structure of scale factors for time, mass, length, velocity, angular velocity, acceleration, angular acceleration, force, and torque. Only the scale factors for time, mass, and length will be manually assigned. The other scale factors are calculated so as to keep the units consistent. For example, the scale factor for velocity is equal to the length scale divided by the time scale. Set the time scale factor to 30 (i.e. 1 second = 30 new time units) and leave those of mass and length at 1.

It is often useful to be able to scale the cost function separately from the rest of the variables. Create a cost scale factor equal to 100 divided by the time scale factor.

```
scale.time = 30;
scale.mass = 1;
scale.length = 1;
scale.vel = scale.length/scale.time;
scale.accel = scale.vel/scale.time;
scale.force = scale.mass*scale.accel;
scale.torque = scale.force*scale.length;
scale.angVel = 1/scale.time;
scale.angAccel = scale.angVel/scale.time;
scale.inertia = scale.torque/scale.angAccel;

scale.cost = 1e2/scale.time;
```

Load Data:
Load the file `IK_data.mat` (make sure that it is in your Matlab path or current working folder). This file contains an `auxdata` structure with various constant parameters as well as splines for time varying reference data such as ground reaction loads. It may be useful to take a minute to look through everything contained in the file. After loading the file, add the `scale` structure you created to the `auxdata` structure as a new field.

```
load('IK_data.mat');
auxdata.scale = scale;
```

Bounds:
GPOPS must be given appropriate bounds for all variables. Define variables for maximum and minimum values of the generalized coordinates ($-\pi/3 \le q_1 \le \pi/3$, $-\pi/12 \le q_2 \le \pi/2$, $-\pi/2 \le q_3 \le \pi/2$, $-0.3 \le q_4 \le 0.3$, $0.75 \le q_5 \le 1$), the derivatives of the generalized coordinates ($-3\pi \le u_1 \le 3\pi$, $-3\pi \le u_2 \le 3\pi$, $-3\pi \le u_3 \le 3\pi$, $-0.4 \le u_4 \le 0.4$, $-1.5 \le u_5 \le 1.5$), and torque actuators ($-500 \le TA \le 500$, $-500 \le TK \le 500$), and the cost (0 ≤ cost ≤ 10). Also define the initial and final time (`t0 = 0, tf = 1`).

```
t0 = 0;
tf = 1;

q1_min = -pi/3;
q1_max = pi/3;
q2_min = -pi/12;
q2_max = pi/2;
q3_min = -pi/2;
q3_max = pi/2;
```

```
q4_min = -0.3;
q4_max = 0.3;
q5_min = 0.75;
q5_max = 1;

u1_min = -3*pi;
u1_max = 3*pi;
u2_min = -3*pi;
u2_max = 3*pi;
u3_min = -3*pi;
u3_max = 3*pi;
u4_min = -0.4;
u4_max = 0.4;
u5_min = -1.5;
u5_max = 1.5;

TA_min = -500;
TA_max = 500;
TK_min = -500;
TK_max = 500;

cost_min = 0;
cost_max = 10;
```

Now we will create a structure called `bounds` with a field called `phase`. This is a single phase problem, but GPOPS still requires a field for phases. `Phase` will have fields for `state`, `control`, `integral`, `path`, `initialtime`, `finaltime`, `initialstate`, and `finalstate`, and each of these will have fields called `lower` and `upper`. Assemble the min and max values defined above into row arrays in the upper and lower bound structures. The bounds for the initial and final state should be the same as the bounds for the state. When assembling the variables into their corresponding structures, multiply each one by the correct scale factor. For example, the length coordinates (q_4 and q_5) must be multiplied by the length scale factor. The initial and final time must also be scaled (by the time scale factor), the velocities and angular velocities must be scaled by their respective scale factors, control torques must be scaled by the torque scale factor, and the cost should be scaled by the cost scale factor.

```
bounds.phase.initialtime.lower = t0*scale.time;
bounds.phase.initialtime.upper = t0*scale.time;
bounds.phase.finaltime.lower = tf*scale.time;
bounds.phase.finaltime.upper = tf*scale.time;
bounds.phase.initialstate.lower =
[q1_min,q2_min,q3_min,q4_min*scale.length,q5_min*scale.length, ...
    u1_min*scale.angVel,u2_min*scale.angVel,u3_min*scale.angVel, ...
    u4_min*scale.vel,u5_min*scale.vel];
bounds.phase.initialstate.upper =
[q1_max,q2_max,q3_max,q4_max*scale.length,q5_max*scale.length, ...
    u1_max*scale.angVel,u2_max*scale.angVel,u3_max*scale.angVel, ...
    u4_max*scale.vel,u5_max*scale.vel];
bounds.phase.finalstate.lower =
[q1_min,q2_min,q3_min,q4_min*scale.length,q5_min*scale.length, ...
    u1_min*scale.angVel,u2_min*scale.angVel,u3_min*scale.angVel, ...
    u4_min*scale.vel,u5_min*scale.vel];
bounds.phase.finalstate.upper =
```

```
[q1_max,q2_max,q3_max,q4_max*scale.length,q5_max*scale.length, ...
    u1_max*scale.angVel,u2_max*scale.angVel,u3_max*scale.angVel, ...
    u4_max*scale.vel,u5_max*scale.vel];
bounds.phase.state.lower =
[q1_min,q2_min,q3_min,q4_min*scale.length,q5_min*scale.length, ...
    u1_min*scale.angVel,u2_min*scale.angVel,u3_min*scale.angVel, ...
    u4_min*scale.vel,u5_min*scale.vel];
bounds.phase.state.upper =
[q1_max,q2_max,q3_max,q4_max*scale.length,q5_max*scale.length, ...
    u1_max*scale.angVel,u2_max*scale.angVel,u3_max*scale.angVel, ...
    u4_max*scale.vel,u5_max*scale.vel];
bounds.phase.control.lower = [TA_min*scale.torque,TK_min*scale.torque];
bounds.phase.control.upper = [TA_max*scale.torque,TK_max*scale.torque];
bounds.phase.integral.lower = cost_min;
bounds.phase.integral.upper = cost_max*scale.cost;
```

Initial Guess:
An initial guess may also be provided to GPOPS. We will be providing a simple guess with only initial and final values for the `time`, `state`, and `controls`.

- Time guess: initial and final time multiplied by the time scale factor
- States guess: static, upright pose ($q_{1\text{-}3}$ = 0, q_4 = 0.2, q_5 = 0.935, $u_{1\text{-}5}$ = 0) for both initial and final points
- Control guess: zeros
- Integral guess: 0.001

Assemble these quantities into a structure called `guess.phase`. Remember that each field in `guess.phase` except for `integral` (i.e. `time`, `state`, `control`) should be a 2 x n array where n is number of variables for that field. The guess provided for the `integral` (which will be the cost value for us) is always 1 x n.

```
guess.phase.time    = [t0*scale.time; tf*scale.time];
guess.phase.state   =
[zeros(1,3),0.2*scale.length,0.935*scale.length,zeros(1,5);
zeros(1,3),0.2*scale.length,0.935*scale.length,zeros(1,5)];
guess.phase.control = zeros(2,2);
guess.phase.integral = 0.001*scale.cost;
```

Problem Structure:
Now we will setup the actual problem structure to be given to GPOPS. Look over it and note the options used and how they are assembled into the setup structure.

```
setup.name = 'Inverse_Kinematics';
setup.functions.continuous = @IK_continuous;
setup.functions.endpoint = @IK_endpoint;
setup.auxdata = auxdata;
setup.bounds = bounds;
setup.guess = guess;

setup.nlp.solver = 'ipopt';
setup.nlp.ipoptoptions.tolerance = 1e-7;
setup.derivatives.supplier = 'sparseCD';
```

```
setup.derivatives.derivativelevel = 'second';
setup.derivatives.dependencies = 'sparse';

setup.mesh.phase.colpoints = 4*ones(1,20);
setup.mesh.phase.fraction = 1/20*ones(1,20);
setup.mesh.method = 'hp-LiuRao';
setup.mesh.tolerance = 1e-4;
setup.mesh.maxiterations = 10;
setup.mesh.colpointsmin = 4;
setup.mesh.colpointsmax = 10;

setup.method = 'RPM-Integration';
```

Run GPOPS:
Call the `gpops2` function with the `setup` structure as the input, and create a variable for the output. Save the output to your computer so that it can be reloaded later.

```
tic
output = gpops2(setup);
toc

save('IK_output','output');
```

Post Analysis:
A post hoc analysis function has been provided to graph the output from GPOPs and a reference solution. Run the function `IK_postAnalysis` with the inputs of `auxdata` and the output from GPOPS2. If any of the variable names in the `auxdata` structure are different from those presented in this document, then this function will need to be modified to reflect the new variable names.

```
IK_postAnalysis(auxdata,output);
```

This is the end of the main file.

b. Continuous Function

Open the file `IK_continuous.m` in the Matlab editor. This is the continuous function which will be called by GPOPS.

Unpack Auxdata:
First we will “unpack” the scale and spline fields from the `auxdata` structure so that we can access the variables contained more easily.

```
scale = input.auxdata.scale;
spline = input.auxdata.spline;
```

Unscale Time, States, and Controls:
We have given scaled versions of the states and controls to GPOPS, but it will be easier to work with these variables inside the continuous function if we unscale them first. We will later rescale

the output of the continuous function so that GPOPS is working with scaled variables. Unpack and unscale (by dividing by the correct scale factor) `time`, q_{1-5}, u_{1-5}, `TA`, and `TK`.

```
t = input.phase.time/scale.time;
q1 = input.phase.state(:,1);
q2 = input.phase.state(:,2);
q3 = input.phase.state(:,3);
q4 = input.phase.state(:,4)/scale.length;
q5 = input.phase.state(:,5)/scale.length;
u1 = input.phase.state(:,6)/scale.angVel;
u2 = input.phase.state(:,7)/scale.angVel;
u3 = input.phase.state(:,8)/scale.angVel;
u4 = input.phase.state(:,9)/scale.vel;
u5 = input.phase.state(:,10)/scale.vel;

TA = input.phase.control(:,1)/scale.torque;
TK = input.phase.control(:,2)/scale.torque;
```

Sample Splines:
Now sample the splines for the ground reaction loads and all of the reference marker positions using the `ppval` command with the unscaled time vector.

```
M_ref.xB1 = ppval(spline.markers.xB1,t);
M_ref.yB1 = ppval(spline.markers.yB1,t);
M_ref.xB2 = ppval(spline.markers.xB2,t);
M_ref.yB2 = ppval(spline.markers.yB2,t);
M_ref.xC1 = ppval(spline.markers.xC1,t);
M_ref.yC1 = ppval(spline.markers.yC1,t);
M_ref.xC2 = ppval(spline.markers.xC2,t);
M_ref.yC2 = ppval(spline.markers.yC2,t);
M_ref.xD1 = ppval(spline.markers.xD1,t);
M_ref.yD1 = ppval(spline.markers.yD1,t);
M_ref.xD2 = ppval(spline.markers.xD2,t);
M_ref.yD2 = ppval(spline.markers.yD2,t);
```

Skeletal Dynamics and Marker Positions:
Get the accelerations of the generalized coordinates by running the provided `skeletalDynamics` function. The inputs for this function are `auxdata`, time, the generalized coordinates and their velocities, and the joint torques, and the outputs of the function are the generalized coordinate accelerations and a structure containing the marker positions.

```
[u1d,u2d,u3d,u4d,u5d,markers] =
skeletalDynamics(input.auxdata,t,q1,q2,q3,q4,q5,u1,u2,u3,u4,u5,TA,TK);
```

Unpack Marker Positions:
Unpack the individual marker positions from the `markers` structure output by `skeletalDynamics`. The marker coordinates are named using the same convention as they were for the reference coordinate positions.

```
xB1 = markers.xB1;
```

```
yB1 = markers.yB1;
xB2 = markers.xB2;
yB2 = markers.yB2;
xC1 = markers.xC1;
yC1 = markers.yC1;
xC2 = markers.xC2;
yC2 = markers.yC2;
xD1 = markers.xD1;
yD1 = markers.yD1;
xD2 = markers.xD2;
yD2 = markers.yD2;
```

Output:
The output structure for the continuous function must, for this problem, include the dynamics and the integrand. The `dynamics` field contains an array of the derivatives of the states at each time point and `integrand` contains an array of the values of the curve to be integrated at each time point. We must rescale all of these quantities such that GPOPS is working with the scaled versions.

First, construct an array for the `dynamics`. The derivatives of the generalized coordinates (the first five states) are the generalized velocities (the second five states). The derivatives of the velocities are the accelerations we previously calculated. In order to be properly scaled, all of the derivatives should be multiplied by their scale factors. For example, the derivative of q_1 (u_1) should be multiplied by the angular velocity scale, whereas the derivative of u_4 should be multiplied by the acceleration scale factor.

The `integrand` will be the sum of the marker errors squared (for each time point) multiplied by the length scale factor squared times the cost scale factor. Thus the integrand array should be of size one by number of time points.

```
cost = ((xB1-M_ref.xB1).^2 + (yB1-M_ref.yB1).^2 + (xB2-M_ref.xB2).^2 + ...
    (yB2-M_ref.yB2).^2 + (xC1-M_ref.xC1).^2 + (yC1-M_ref.yC1).^2 + ...
    (xC2-M_ref.xC2).^2 + (yC2-M_ref.yC2).^2 + (xD1-M_ref.xD1).^2 + ...
    (yD1-M_ref.yD1).^2 + (xD2-M_ref.xD2).^2 + (yD2-M_ref.yD2).^2);

output.integrand = cost*scale.length^2*scale.cost;
output.dynamics =
[u1*scale.angVel,u2*scale.angVel,u3*scale.angVel,u4*scale.vel, ...
    u5*scale.vel,u1d*scale.angAccel,u2d*scale.angAccel, ...
    u3d*scale.angAccel,u4d*scale.accel,u5d*scale.accel];
```

This is the end of the continuous function.

c. Endpoint Function

Open the file `IK_endpoint.m` in the Matlab editor. This is the endpoint function that will be called by GPOPS.

The endpoint function for this problem is very simple. All we need to do for the endpoint function is set the field `output.objective` equal to the field `input.phase.integral`.

```
output.objective = input.phase.integral;
```

Now that the whole problem is constructed, run the main file. You should notice that, while the marker position and generalized coordinate trajectories look fine, the torques and accelerations have some regions where the solution is not very good. This occurs because the marker positions are not very sensitive to high frequency oscillations or spikes in joint torques. A very fast oscillation of the controlled joint torques about the true joint torques will only cause very small marker errors. This occurs because the torques are used to calculate generalized coordinate accelerations, which are then integrated twice to get positions. This double integral damps out high frequencies. One method to solve this problem is to track marker accelerations as well as positions. Experimental marker trajectories, however, will be noisy, and therefore calculating a second derivative numerically will yield very extremely noisy accelerations. A Kalman smoothing approach [1] can be used to eliminate the problem of noisy accelerations.

After the problem is working, try changing the time scale factor from 30 to 1. Does the problem still converge? What about if it is set to 60? What if the mass scale factor is changed to 100? Before running the problem with new scaling, change the `maxiterations` option in `setup.mesh` from 10 to 0 so that you aren't spending a lot of time waiting for ten mesh iterations for a problem that may never converge.

Now change the cost scale factor from 100 to 1. What happens to the solution?

References

[1] De Groote, F., et al., 2008, "Kalman smoothing improves the estimation of joint kinematics and kinetics in marker-based human gait analysis." J Biomech, 41:3390-3398.