1 Introduction

In general, the kinematics of the human body can be completely described by specifying the general-ized coordinates (i.e. joint angles, center of mass position and body orientation in space) over a period of time. Given a completely specified motion, inverse dynamics can be used to explicitly compute the required external forces and joint torques necessary to produce this motion; however, because every joint in the body is actuated by several muscles, the actual distribution of muscle forces required to produce these joint torques is mathematically indeterminate. Clinically, however, it is important to understand the contribution of individual muscles to observed motion. For example, changes in the force generation patterns of specific muscle groups can lead to observable gait abnormalities and severe loss of musculoskeletal function [DAA07].

The muscle force distribution can be experimentally estimated by measuring the electric simulation of individual muscles via EMG (electromyography) data. However, EMG data is extremely noisy and it is often difficult to accurately measure the simulation of distinct muscles. Furthermore, it is difficult to actually measure the muscle forces experimentally. To that end, having a computational model of the neuromuscular system to numerically predict the muscle force distribution is a very useful tool; and poses a problem in which optimization techniques are extremely applicable, albeit still difficult [DAA07].

The mathematical constraints on the muscle forces come from (i) the dynamic equations of motion; and (ii) the neuromuscular contraction-excitation dynamics, which include relationships between muscle neural excitation, actual muscle activation, force-length-velocity properties of the muscle, and force-length properties of the tendon. When the joint motion is specified, the dynamic equations of motion are linear in the muscle forces. However, the constraints induced by the muscle physiology are highly non-linear and moreover, non-convex. While the problem is slightly simplified in the static case, a method that can handle the general dynamic case is certainly preferable. In addition to this, the actual objective function associated with this problem is unknown. Many potential candidates exist, including total muscle activation, total muscle stress, metabolic energy cost, and activation time. Many of these objectives are convex [EMHvdB07]. In line with the simulation program OpenSim, created at Stanford University, the cost function analyzed here will be the activation-squared.
2 Model

The problem examined here concerns just one degree of freedom in body, elbow flexion, and uses convex programming to solve the muscle force distribution problem required to move a 10kg mass through an elbow flexion range of 130°. The key muscles actuating the elbow are the elbow flexors: Biceps (long head and short head) and Brachialis (deep to the Biceps); and the elbow extensors: Triceps (long head, lateral head, and medial head). [See Figure 1.]

![Figure 1: OpenSim Snapshot showing elbow flexors and extensors](image1)

![Figure 2: Representation of musculo-tendon force unit [The03]](image2)

2.1 Objective Function

The goal of the problem is to find the tension sustained by each muscle over the entire range of motion. (Note that these tensions depend on the actual elapsed time, which here, we take as a specified variable.) To solve this distribution problem, we discretize the motion into $K$ timesteps and do a regularized least-squares on three activation-derived cost functions:

$$J_1 = \sum_{i,k} a_{i,t_k}^2 \quad i = 1, ..., 6; k = 1, ..., K$$

$$J_2 = \sum_{i,k} (a_{i,t_{k+1}} - a_{i,t_k})^2 \quad i = 1, ..., 6; k = 1, ..., K - 1$$

$$J_3 = \sum_{i} a_{i,t_1}^2 \quad i = 1, ..., 6$$

It is worth noting that the muscle activation (which directly effects muscle force generation) is actually a time-delayed response to the neural muscle excitation signal, $u$. The two are related by $\frac{da}{dt} = \frac{u-a}{\tau(a,u)}$, where the time constant $\tau(a,u)$ is a non-linear, piecewise function dependent on whether $u \leq a$ or $u > a$.

The muscle excitation, which is a measure of the actual neural signal to the muscle is a more relevant parameter to involve in the objective functions. However, because the relationship between the muscle excitation to the muscle activation is difficult to model, this behaviour is instead captured in the objective functions. In particular, minimizing $J_1$ is correlated to globally minimizing total muscle excitation, minimizing $J_2$ controls against large spikes in excitation levels (since excitation is roughly the derivative of activation), and $J_3$ accounts for the time delay between excitation and activation by saying the initial activation level should be small.
2.2 Constraints

2.2.1 Dynamic Motion Constraints

The elbow is modeled as a revolute joint with elbow flexion defined as a positive rotation about the unit vector $\mathbf{e}_{axis}$. The system ($S$) is taken to be the forearm/hand ($B$) and a 10kg mass ($P$) held at the hand. Forming the rotational equation of motion about a point $E_o$ on the elbow with respect to the elbow revolute axis gives:

$$I\ddot{\theta} = -r_{\perp B_{cm}} m_B g - r_{\perp P} m_P g + \sum m r_{\perp m} F_{MT}$$

(1)

where $I$ is the appropriate moment of inertia, $r_{\perp B_{cm}}$ and $r_{\perp P}$ are the moment arms of the gravitational forces on $B$ and $P$ respectively, and $r_{\perp m}$ is the moment arm of the musculotendon force $F_{MT}$ about point $E_o$. [The musculotendon force moment arm is a complicated function of geometry (tendon-bone attachment points, muscle wrapping around bone, non-linear muscle path, etc.) and the elbow flexion angle, but does not depend in any way on the muscle forces; for this problem, the moment arm data was taken directly from the OpenSim simulation program and not explicitly computed.] This continuous dynamic constraint is applied at each discretized time step.

2.2.2 Muscle Physiology Constraints

Figure 2 shows a commonly used muscle-tendon schematic. In the figure, [CE] represents the contraction-excitation dynamics and active force generation of the muscle fibers, the parallel spring represents the muscle compliance, $\alpha$ is the pennation angle of the muscle fibers, and the series spring represents tendon compliance.

The forces that can be generated by a muscle depend non-linearly on several muscle-specific parameters including the muscle fiber length, contraction velocity, tendon slack length and compliance (shown in Figures 3, 4, and 5 respectively).

See [The03] for specific equations representing these curves.

These properties give rise to the constraints described below.

The activation is a dimensionless number always between 0 and 1:

$$0 \leq a_{i,t_k} \leq 1, \ i = 1, ..., 6; k = 1, ..., K$$

(2)
and the actual muscle force is a linear function of activation, but a highly-nonlinear function of muscle length and muscle velocity:

\[ F_{i,t_k}^M = F_{i,t_k}^{M\text{passive}} + a_{i,t_k} F_{i,t_k}^{M\text{active}}, \quad i = 1, ..., 6; k = 1, ..., K \]  

(3)

The total muscle-tendon unit length is purely a function of tendon attachment point and muscle wrapping geometry and does not depend on the applied forces in any way; again, this data was pulled from OpenSim without explicit computation here. The lengths of the muscle fibers and tendon individually, however, are unknown, which imposes the following constraint on the muscle length \( L^M \) and the tendon length \( L^T \) for all muscles and all time steps:

\[ L_{i,t_k}^M + L_{i,t_k}^T = L_{i,t_k}^{MT}, \quad i = 1, ..., 6; k = 1, ..., K \]  

(4)

Finally, because the muscle and tendon are forces in series, we impose the constraint that the two forces must be equal:

\[ F_{i,t_k}^M = F_{i,t_k}^T, \quad i = 1, ..., 6; k = 1, ..., K \]  

(5)

### 3 Problem Statement

Consolidating the objectives, \( J_1, J_2, \) and \( J_3, \) and constraints (1) - (5) from the model, we can pose the following problem statement with variables \( a_{i,t_k}, L_{i,t_k}^M, \) and \( L_{i,t_k}^T, \) for \( i = 1, ..., 6; k = 1, ..., K \)

\[
\text{minimize} \quad J_1 + \lambda J_2 + \mu J_3
\]

\[
\text{subject to} \quad I\ddot{\theta}_{t_k} = -r_{\perp B_{cm}t_k} m_B g - r_{\perp P_{tk}} m_P g + \sum_i r_{\perp i_{tk}} F_{i,t_k}^{MT}, \quad k = 1, ..., K
\]

\[
0 \leq a_{i,t_k} \leq 1, \quad i = 1, ..., 6; k = 1, ..., K
\]

\[
F_{i,t_k}^M = F_{i,t_k}^{M\text{passive}} + a_{i,t_k} F_{i,t_k}^{M\text{active}}, \quad i = 1, ..., 6; k = 1, ..., K
\]

\[
L_{i,t_k}^M + L_{i,t_k}^T = L_{i,t_k}^{MT}, \quad i = 1, ..., 6; k = 1, ..., K
\]

\[
F_{i,t_k} = F_{i,t_k}^T, \quad i = 1, ..., 6; k = 1, ..., K
\]

### 4 Solution Algorithm

The problem is convex with respect to the activation variables and constraints (1) - (3) when the motion (i.e. \( \theta(t) \)) is specified. However, including the muscle and tendon lengths as unknowns along with constraint (5) makes this problem decidedly non-convex and tricky to solve.

One way to solve this problem is to make a guess at the muscle and tendon lengths over the trajectory, solve the problem using only activation variables and constraints (1) - (3), check the violation of constraint (5), refine the guesses of the muscle and tendon lengths, and iterate. The convergence of this method is sensitive to the initial guess of muscle and tendon lengths, as well as to the method of guess refinement.
Here, we first solve different, but related optimization problem:

\[
\text{minimize} \quad \sum_{i,k} F_{i,t,k}^2 + \eta \sum_{i,k} (F_{i,t,k+1} - F_{i,t,k})^2
\]

subject to

\[
I \ddot{\theta}_k = -r_{\perp Bcm} m_B g - r_{\perp P} m_P g + \sum_i r_{\perp M} F_{MT}^k, \quad k = 1, ..., K
\]

\[
0 \leq F_{i,t,k} \leq F_{\text{max},i}, \quad i = 1, ..., 6
\]

with unknowns \( F_{i,k}, i = 1, ..., 6 \) and \( k = 1, ..., K \).

From the forces obtained here, because there is a one-to-one relationship between tendon force and tendon length, we can solve for the tendon lengths \( L^T \) by assuming \( F^T = F \) (calculated here) and solve for the muscle lengths by taking \( L^M = L^{MT} - L^T \). These lengths can then be used as guesses for the modified optimization problem consisting of the original objective \( J_1 + \lambda J_2 + \mu J_3 \) and the original constraints (1) - (3).

After each iteration, we will have a set of muscle activations, as well as our guesses of muscle and tendon length. Using these parameters, we can calculate and check the difference between the muscle and tendon forces. In this statement, we check that \( \frac{1}{6K} \sum_{i,k} (F_{i,t,k}^T - F_{i,t,k}^M)^2 \leq \varepsilon \), for some specified tolerance \( \varepsilon \). If the error is too large, we guess that the actual muscle-tendon force is near the average of the current calculated muscle and tendon forces, resolve for the muscle and tendon lengths, and repeat.

5 Results

The first test run used an essentially static case (the elapsed time for the elbow to go from fully extended to 130° was 10 seconds). Here, we had convergence after 7 iterations. (See figures 6 and 7.)

![Figure 6: Muscle force (solid) and Tendon force (dashed) after one averaging iteration](image)

![Figure 7: Muscle force (solid) and Tendon force (dashed) after seven averaging iterations](image)
The activation plots, shown in Figures 8 and 9 are nearly identical to the results computed in the simulation software, OpenSim, with the added smoothing in the activation curves in this model.

![Figure 8: Activation curves from cvx model](image)

![Figure 9: Activation curves from OpenSim](image)

We see similar validation in figures 10 and 11 when running a dynamic second (flexion within 1 second).

![Figure 10: Activation curves from cvx model](image)

![Figure 11: Activation curves from OpenSim](image)

The total run time for any of these simulations did not take longer than approximately 45 seconds, which is comparable to the computation time in OpenSim.

6 Conclusion

Convex optimization techniques are a useful tool in analyzing the muscle force distribution problem. Even though the distribution problem itself is not convex as posed, certain approximations and iterative solving are useful in forming a tractable, convex problem statement. This method is still very computationally expensive, especially when dealing with larger systems, but is still a valuable approach.
References


7 Code

7.1 MUSCLE object

Each muscle was constructed as an object with muscle-specific properties. Parameters common to most muscles were pre-set, but can still be changed.

%%% Muscle parameters
classdef MUSCLE
    properties
        % STANDARD MUSCLE/TENDON FORCE PARAMETERS
        kP = 4 % Passive force shape factor
        gamma = 0.5 % Active force shape factor
        epsM = 0.6 % Passive muscle strain due to max isometric force
        epsT = 0.033 % Tendon strain due to max isometric force
        FTtoe = 0.33 % Transition from non-linear to linear behavior (normalized)
        ktoe = 3 % Tendon force-strain shape factor
        Vmax = 10 % Maximum contraction velocity, in opt. fiber-lengths per second
        kV1 = 1.0894 % Active force-velocity shape factor 1
        kV2 = 0.25 % Active force-velocity shape factor 2
        tau_act = 0.01 % Activation time constant
        tau_deact = 0.04 % Deactivation time constant

        % MUSCLE-SPECIFIC PARAMETERS
        FmaxISO % Max isometric force
        optFiberLength % Optimum fiber length (max isometric force)
        tendSlackLength % Tendon slack length
        alpha % Pennation angle at optimum fiber length
    end
end

7.2 Muscle & Tendon Force-Length-Velocity Relationships

7.2.1 Muscle active force generation

function activeForce = fActive( anyMUSCLE, L )
%% Calculates the un-normalized active force generated
%% by anyMUSCLE at the given muscle fiber length, L

gamma = anyMUSCLE.gamma;
Lm = L./(anyMUSCLE.optFiberLength);
FmaxISO = anyMUSCLE.FmaxISO;

activeForce = FmaxISO*( exp( -(Lm-1).^2/gamma ) );
7.2.2 Muscle passive force generation

function passiveForce = fPassive( anyMUSCLE, L )
% Calculates the un-normalized passive force generated
% by anyMUSCLE at the given muscle fiber length, L

kP = anyMUSCLE.kP;
Lm = L./(anyMUSCLE.optFiberLength);
epsM = anyMUSCLE.epsM;
FmaxISO = anyMUSCLE.FmaxISO;

passiveForce = FmaxISO.*(exp(kP.*(Lm - 1)./epsM) - 1 )./( exp(kP) - 1 );

7.2.3 Muscle force-velocity factor

function scaleFactor = FVrel( anyMUSCLE, V )
% Calculates the force-velocity effect on max active force
% for anyMUSCLE at the given fiber shortening velocity, V

Vmax = anyMUSCLE.Vmax;
Vm = V./(anyMUSCLE.optFiberLength);
kV1 = anyMUSCLE.kV1;
kV2 = anyMUSCLE.kV2;

if Vm <= 0
    scaleFactor = 1.8 + 0.5./(Vm-1./sqrt(kV1.*kV2));
elseif Vm >= Vmax
    scaleFactor = 0;
else
    scaleFactor = 1 - kV1.*(1-exp(-kV2.*Vm));
end

7.2.4 Tendon force-length relation

function tendonForce = fTendon( anyMUSCLE, LT )
% Calculates the un-normalized tendon force generated
% by anyMUSCLE at the given tendon length, LT

epsT = anyMUSCLE.epsT;
epsTtoe = 0.609*epsT;
ktoe = anyMUSCLE.ktoe;
klin = 1.712/epsT;

FmaxISO = anyMUSCLE.FmaxISO;
FTtoe = anyMUSCLE.FTtoe;
\% tendon strain
eps = (LT-anyMUSCLE.tendSlackLength)./anyMUSCLE.tendSlackLength;

if eps <= epsTtoe
    tendonForce = (FTtoe./(exp(ktoe) - 1)).*(exp(ktoe.*eps./epsTtoe) - 1);
else
    tendonForce = klin.*(eps - epsTtoe) + FTtoe;
end

tendonForce = tendonForce.*FmaxISO;

7.3 Main Code

clear all; close all; clc;

numActuators = 6;
numDiscr = 100;

%% Initialize muscles
muscles = {'BIClong', 'BICshort', 'BRA', 'TRIlat', 'TRIlong', 'TRImed'};

BIClong = MUSCLE;
    BIClong.FmaxISO = 624.3;
    BIClong.optFiberLength = 0.1157;
    BIClong.tendSlackLength = 0.2723;
    BIClong.alpha = 0;

BICshort = MUSCLE;
    BICshort.FmaxISO = 435.56;
    BICshort.optFiberLength = 0.1321;
    BICshort.tendSlackLength = 0.1923;
    BICshort.alpha = 0;

BRA = MUSCLE;
    BRA.FmaxISO = 987.26;
    BRA.optFiberLength = 0.0858;
    BRA.tendSlackLength = 0.0535;
    BRA.alpha = 0;

TRIlat = MUSCLE;
    TRIlat.FmaxISO = 624.3;
    TRIlat.optFiberLength = 0.1138;
    TRIlat.tendSlackLength = 0.098;
    TRIlat.alpha = 0.15707963;
TRIlong = MUSCLE;
TRIlong.FmaxISO = 798.52;
TRIlong.optFiberLength = 0.134;
TRIlong.tendSlackLength = 0.143;
TRIlong.alpha = 0.20943951;

TRImed = MUSCLE;
TRImed.FmaxISO = 624.3;
TRImed.optFiberLength = 0.1138;
TRImed.tendSlackLength = 0.0908;
TRImed.alpha = 0.15707963;

allMUSCLES = [BIClong, BICshort, BRA, TRIlat, TRIlong, TRImed];

%% Get muscle-tendon length, moment arm data (geometry)
load MuscleTendonLengthData.txt
load MomentArmData.txt

LenMT = MuscleTendonLengthData(:,2:7);
MA = MomentArmData(:,2:7);

%% Get specified motion
ElapsedTime = 1;
generateThetaB  \% generates 100x1 vector of qB, qBp, qBpp

%% Make initial guesses

\% Guess muscle length, tendon length over trajectory
\% by first optimizing w.r.t. least-squares on tendon forces
\% Then solve for the tendon lengths that achieve this

tic
cvx_begin
variables FT(numDiscr, numActuators)

diffMat = full(spdiags([ones(numDiscr,1), -ones(numDiscr,1)], 0:1, numDiscr-1,numDiscr));
lambda = [1,1];

cost(1) = sum((FT(:,1)./allMUSCLES(1).FmaxISO).^2);
for i=2:numActuators
    cost(1)= cost(1) + sum((FT(:,i)./allMUSCLES(i).FmaxISO).^2);
end

```matlab
end
```
cost(2) = 1./(numDiscr.*numActuators)*sum((diffMat*FT(:,1)).^2);
for i=2:numActuators
    cost(2) = cost(2) + 1./(numDiscr.*numActuators)*sum((diffMat*FT(:,i)).^2);
end

objective = sum(lambda.*cost);

minimize( objective );

subject to
    for i=1:numDiscr
        totalMuscleTorque = sum(MA(i,:).*FT(i,:));
        2.695183*cos(qB(i)) + 33.34295*sin(qB(i)) + 1.054046*qBpp(i) == totalMuscleTorque;
    end

0 <= FT;
    for i=1:numActuators
        FT(:,i) <= allMUSCLES(i).FmaxISO;
    end

cvx_end

% Calculate tendon lengths
for i=1:numDiscr
    for j = 1:numActuators
        LT(i,j) = getTendonLengthFromForce(allMUSCLES(j), FT(i,j));
    end
end

LM = LenMT - LT;

% Now iterate to try and find lengths
% s.t. tendon force = muscle force (roughly)
err = 1e9;
tolerance = 1e-1;
numIter = 1;
while err > tolerance
    % Calculate muscle shortening velocity over trajectory
    Vshort = zeros(numDiscr, numActuators);
    for i=1:numActuators
        Vshort(:,i) = -([0; diff(LM(:,i))./diff(qB)]).*qBp;
    end

% Calculate muscle pennation angles over trajectory
pennAngles = zeros(numDiscr, numActuators);
for i = 1:numDiscr
    for j = 1:numActuators
        pennAngles(i,j) = getAlpha(allMUSCLES(j), LM(i,j));
    end
end

% Calculate tendon force along trajectory
tendonForce = zeros(numDiscr, numActuators);
for i = 1:numDiscr
    for j = 1:numActuators
        tendonForce(i,j) = fTendon(allMUSCLES(j), LT(i,j));
    end
end

clear a cost objective muscleTension totalMuscleTorque;
cvx_begin
variables a(numDiscr, numActuators)

diffMat = full(spdiags([ones(numDiscr,1), -ones(numDiscr,1)], 0:1, numDiscr-1,numDiscr));
lambda = [1,10,20];

cost(1) = sum(sum(a.^2));
cost(2) = sum((diffMat*a(:,1)).^2 + (diffMat*a(:,2)).^2 + (diffMat*a(:,3)).^2 + (diffMat*a(:,4)).^2); % Additional term for the fourth dimension

cost(3) = sum(a(1,:).^2);

objective = sum(lambda.*cost);

minimize( objective );

subject to
    for i=1:numDiscr
        for j=1:numActuators
            muscleTension(i,j) = (fPassive(allMUSCLES(j), LM(i,j)) + a(i,j).*fActive(allMUSCLES(j), LM(i,j)));
        end
    end

    totalMuscleTorque = sum(MA(i,:).*muscleTension(i,:));

    2.695183*cos(qB(i)) + 33.34295*sin(qB(i)) + 1.054046*qBpp(i) == totalMuscleTorque;

    0 <= a; a <= 1;
    0 <= muscleTension;

cvx_end
diffA = zeros(numDiscr, numActuators);
for i=1:numActuators
    diffA(:,i) = ([0; diff(a(:,i))./diff(qB)]).*qBp;
end
u = getExcitationFromActiv(allMUSCLES, a, diffA);

tendForce = ... % Some force calculation...
errMat = (muscleTension-tendonForce).^2;
err = 1./(numDiscr.*numActuators).*sqrt((sum(sum(errMat))));
minErrCol = min(errMat);
for i = 1:numDiscr
    for j=1:numActuators
        desiredTendonForce = 0.5*(muscleTension(i,j) + tendonForce(i,j));
        LT(i,j) = getTendonLengthFromForce(allMUSCLES(j), desiredTendonForce);
    end
end
LM = LenMT - LT;
numIter = numIter + 1;

toc

%% plot

plotCode