Static Optimization

Numerical considerations in the use of the polynomial and minmax criterions

Brad T. Humphreys
Cleveland State University
MCE 693 – Human Motion and Control, Fall 2014
humphreysbt@gmail.com

***Abstract*—The polynomial and minmax criteria have been investigated using the the static optimization of a simple 3 Degree of Freedom, 8 muscle lower leg model. Numerical aspects of using these criteria and specifically the effects of scaling have been demonstrated. The effect of muscle discretization on static optimization has also been investigated. The static optimization algorithm in OpenSim has been reviewed with respect to these considerations.**

Keywords—Static Optimization, Polynomial Criterion, Minmax Criterion, OpenSim, Numerical Methods

I. Introduction

As discussed in [1], the human body has 244 kinematic Degrees of Freedom (DoF) and approximately 630 muscles; the human body is redundantly actuated. Due to this redundancy, multiple muscles can provide antagonistic actuation to the same joint. While the required total moment needed to actuate a joint can be calculated (for example from inverse dynamics), the distribution of this moment over more than one muscle is statically indeterminate; this is referred to as the force distribution problem.

A simple and often cited example of this is:

$$M = F_1 d_1 + F_2 d_2 \tag{1}$$

where M is the moment at the anatomical joint, d_1 is moment arm of muscle 1, and d_2 is the moment arm of muscle 2. M is calculated with no required information about the muscle force using inverse dynamics. The muscle moment arms are calculated during inverse dynamics with only muscle geometric information (no muscle force information is required). We are then left with one equation and two unknowns (the muscle forces). The function of static optimization is to determine the muscle forces based on a predefined optimization criterion.

As discussed in [1], it is hypothesized that the central nervous system uses a set of control principles to select muscle activation patterns based on some optimization criteria. The polynomial criterion is one of the most utilized criteria and is the criterion utilized by OpenSim [2].

The polynomial criterion is:

$$G_{poly} = \sum_{i=1}^{\mathrm{m}} \left(\frac{F_i}{PCSA_i}\right)^p \tag{2}$$

where m is the number of muscles, F_i is the force of a muscle, $PCSA_i$ is the muscle's Physiological Cross Section Area, and p is the polynomial exponent. During static optimization, an exponent is chosen and the optimization routine minimizes the value of (2) while satisfying the constraint that forces must generate the desired moments. The exponent p is typically 2 or 3, and the objective can then be stated as the minimization of the sum of squared/cubed muscle stress. For these relatively lower powers of p, the criterion is generally referred to as an "effort" based criterion. At higher values of p, the polynomial criterion is considered to be "fatigue" like [1]. This is due to the high exponent heavily weighting the muscle with the greatest stress; there is then little cost associated with increasing less stressed muscles. The resulting muscle actuation pattern will tend to load level stress across the muscles capable of contributing. It is much less likely that any one muscle will become highly loaded. It has been demonstrated that this criterion along with the incorporation of muscle volume scaling predicts gait.[3]

The minmax criterion for static optimization is:

$$G_{minmax} = max\left(\frac{F_i}{PCSA_i}\right) \tag{3}$$

The optimizer attempts to minimize the max stress in the muscle set. Similar to the polynomial at high values of p, the minmax criterion is a fatigue like criterion. As will be demonstrated, the polynomial criterion approaches the minmax criterion as p goes to infinity. [3]

Details for each of the objective functions and comparison of their results to measured data can be found in [1]. While the main goal in the use of these objective functions is to mimic the muscle activation pattern utilized in the central nervous system, there are considerations that must be made in their ability to be numerically stable and consistent. A simplified model has been utilized to demonstrate the following:

- Polynomial objective functions and scaling
- Comparison of minmax to polynomial
- Muscle discretization
- Review of OpenSim's implementation of static optimization

II. Methods

To support the above goals, static optimization has been performed on a simple three joint, eight muscle leg model. A torque of 50 N-m in the directions shown in Figure 1 is specified at each of the joints: ankle, knee, and hip. Muscle moments arms

are based upon those provided in class (4). The maximum allowable force of each muscle is shown in (5). PCSA is determined by dividing each muscle's max isometric force by the maximum allowable muscle fiber stress (σ_{max}) which is assumed to be 25e4 N/m^2.

$$D = \begin{matrix} (meters) \\ Hip \\ Knee \\ Ankle \end{matrix} \begin{matrix} Ilipos & Glut & Hams & Rect & Vast & Gast & Soleus & Tib \end{matrix} \begin{bmatrix} 0.050 & -0.062 & -0.070 & 0.034 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.034 & 0.050 & 0.042 & -0.020 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.053 & -0.0530 & 0.037 \end{bmatrix} \quad (4)$$

$$F_{max} = \begin{matrix} (N) \\ \end{matrix} \begin{matrix} Ilipos & Glut & Hams & Rect & Vast & Gast & Soleus & Tib \end{matrix} \begin{bmatrix} 1917 & 1967 & 3878 & 1718 & 8531 & 2596 & 3734 & 1233 \end{bmatrix} \quad (5)$$

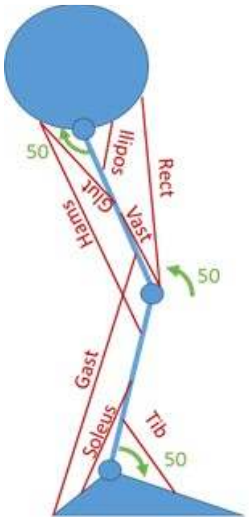

Figure 1. Lower Body Model

All optimizations have been performed in MATLAB using the IPOPT optimizer [5]. Analysis code is provided in the Appendix. Specifics of the software used:

- MATLAB Version: 8.0.0.783 (R2012b)
- IPOPT Version 3.11 with MA57 Linear Solver
- Operating System: Linux 64 Bit Ubuntu 14.04 (3.13.0-39-generic)

IPOPT is utilized with all of its default parameters and only the following explicitly set options:

options.ipopt.hessian_approximation = 'limited-memory';

options.ipopt.to = 1e-7;

III. Results

A. Polynomial Objective Functions and Scaling

Typically the polynomial criterion is utilized with the exponents p=2 or p=3. [1]. If (2) is directly utilized, it suffers from numerical scaling problems at high values of p. For example, if we consider the vastus medialis PCSA of 3.4e-3m^2, at a maximum muscle force of 8531N, the vastus's objective function term will be:

$$G_{poly,vast} = (\frac{F_{max,vast}}{PCSA_{vast}})^p = (\frac{8531}{0.0034})^p = (25e4)^p \quad (6)$$

Noting that because the vastus's F_{max} was utilized for the muscle force in this example, the term becomes equivalent to σ_{max}. From (6), it can be seen that the numerical results become highly sensitive to increasing powers of p. As it is expected that some muscles will provide near zero force and others will provide near their maximum force, the numerical issues at higher values of p should be appreciated.

The polynomial form in (2) is typically modified by noting that:

$$PCSA_i = \frac{F_{max,i}}{\sigma_{max}} \quad (7)$$

$$G_{poly} = \sum_{i=1}^{8} (\frac{F_i}{F_{i,max}} \sigma_{max})^p \quad (8)$$

For an optimization problem, the maximum muscle stress can then be factored out as it is a general parameter for muscle fibers and assumed to not be specific to a particular muscle. The objective function then becomes:

$$G_{poly} = \sum_{i=1}^{8} (\frac{F_i}{F_{i,max}})^p \quad (9)$$

This objective function was implemented as follows in IPOPT:

Objective Function:

$$obj = \sum_{i=1}^{8} \left(\frac{F_i}{F_{i,max}}\right)^p \quad \textbf{(10)}$$

Gradient of the Objective:

$$grad = \frac{p\left(\frac{F_i}{F_{i,max}}\right)^p}{F_i} \quad \textbf{(11)}$$

Constraint Function:

$$cnstr = D * F \quad \textbf{(12)}$$

Jacobian of the Constraint:

$$jac = D \quad \textbf{(13)}$$

IPOPT is configured so that the constraint must equal the moment vector M (an equality constraint) and the values of F_i are limited to the bounds of 0 to $F_{max,i}$ (specific to each muscle).

Polynomial objective functions can become numerically problematic if not scaled correctly. This can be seen in Figure 2 where there is no obvious trend in muscle force and the exponent. IPOPT reports a failure to converge ("Restoration Phase Failed") at p>3. For all exponents, the solution does come close to generating the desired moments.

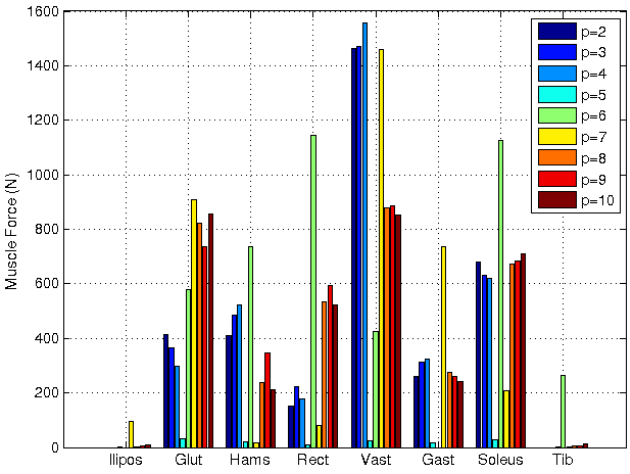

Figure 2. Muscle Force: Polynomial Criterion and Unscaled Objective Function (10)

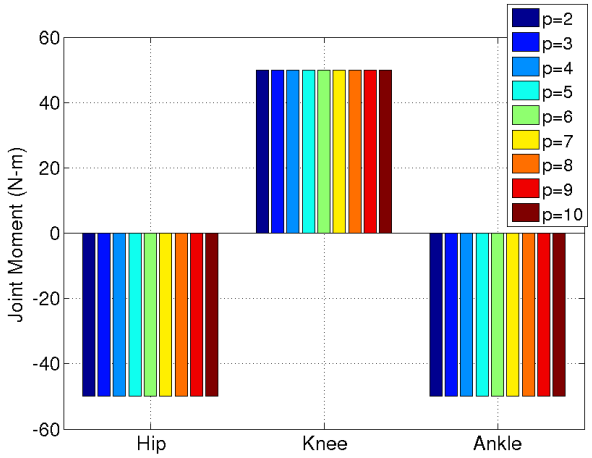

Figure 3. Joint Torques: Polynomial Criterion and Unscaled Objective Function (10)

The lack of a trend and IPOPT's warning were determined to be due to poor scaling in the objective function. It should be appreciated that at very low activation, for example 0.01, and a high power such as p=10, the term for the muscle would be 1e-20. In reviewing the IPOPT documentation, the following was found:

1) "You should try to make the "typical" values of the non-zero first partial derivatives of the objective and constraint functions to be on the order of, say, 0.01 to 100."[6]
2) "By default, Ipopt performs some very simple scaling of the problem functions, by looking at the gradients of each function evaluated at the user-provided starting point. If any entry for a given function is larger than 100, then this function is scaled down to make the largest entry in the gradient 100" [6]
3) The default scaling method is "gradient-based". If the maximum gradient is above 100 (this value is setable using the **nlp_scaling_max_gradient parameter)**, then gradient based scaling will be performed. Scaling parameters are calculated to scale the maximum gradient back to this value. The lower bound for the scaling factors computed by gradient-based scaling method is 1e-8 (setable by using the **nlp_scaling_min_value parameter).** [7]

The IPOPT scaling parameters were not adjusted (but it is planned to investigate them further). Instead a manual scaling factor of 10 was added to the objective and gradient functions:

$$G_{poly} = \sum_{i=1}^{8} \left(10 \frac{F_i}{F_{i,max}}\right)^p \tag{14}$$

The addition of the 10x coefficient produced good results as seen in Figure 4 and Figure 5.

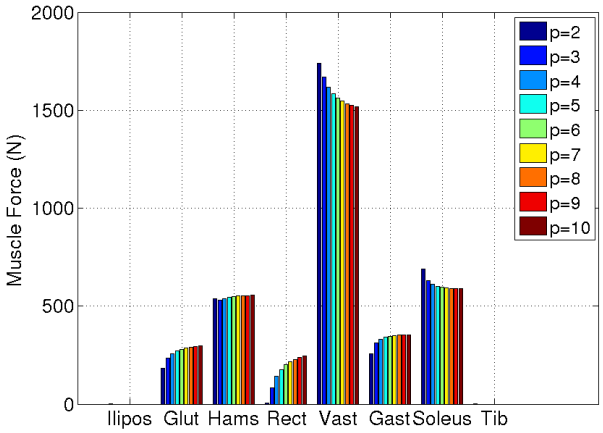

Figure 4. Muscle Force: Polynomial Criterion and Unscaled Objective Function (14)

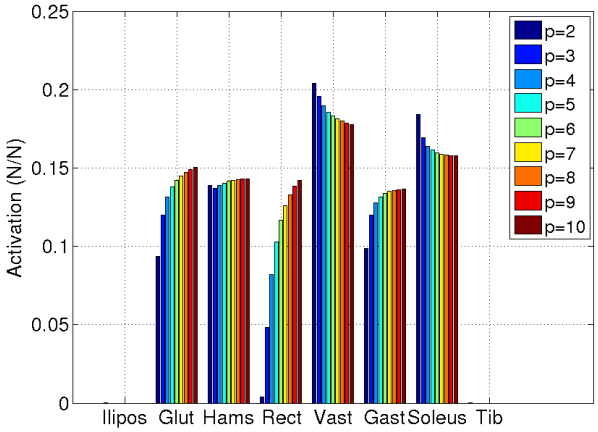

Figure 5. Muscle Force: Polynomial Criterion and Unscaled Objective Function (14)

A max-norm formulation (not to be confused with a minmax optimization) was also investigated. This adds an outer 1/p root to scale the objective:

$$G_p = \left(\sum_{i=1}^{8} \left(\frac{F_i}{F_{i,max}}\right)^p\right)^{1/p} \tag{15}$$

$$\frac{dG_p}{dF_i} = \frac{\left(\frac{F_i}{F_{i,max}}\right)^p}{F_i\left(\sum\left(\frac{F_i}{F_{i,max}}\right)^p\right)^{\frac{p-1}{p}}} \quad (16)$$

The max-norm formulate converged without the need of a scaling coefficient (Figure 6). In addition to the typical p=2 and p=3 exponents, 5 log-spaced exponents from 5 to 50 were also analyzed. It should be mentioned that for all of the criteria that converged in this study, global convergence was verified by additionally running static optimization with 100 trials of random initial guesses of the muscle force. This indicates strong robustness (at least for the model utilized in this study). Equation (16) is not unitless though and the number of muscle terms can affect the scaling; dividing by the number of muscles in the summation term and adding an outer multiplier of the mean F_{max} may provide additional robustness but has not been attempted here.

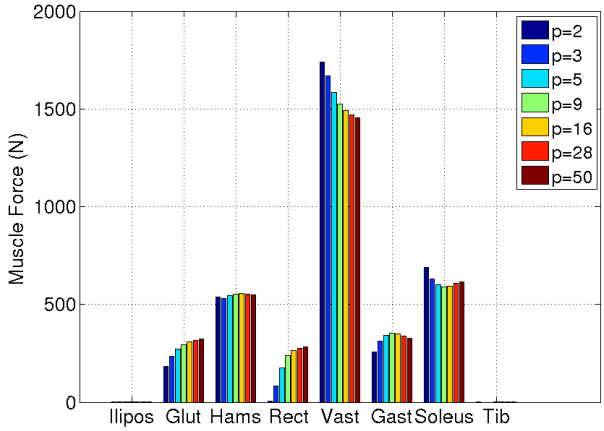

Figure 6. Muscle Force: Polynomial Criterion and Max-Norm Objective Function (15)

B. Comparison of minmax to polynomial

The minmax criterion, minimizing the maximum activation of all the muscles was implemented using the "bound formulation". The objective function is a single parameter: s. This parameter is added add to the vector of muscle values to be solved for by the optimizer (referred to as a slack variable). In addition to the moment constraints, an additional set of constraints linking s and each of the muscle forces are added. The moment constraints are equity constraints (to the value of the moments) and the additional slack variable constraints are inequity constraints that activation minus s must be less than 0.

Objective Function:

$$obj = s \quad \textbf{(17)}$$

Gradient of the Objective Function:

$$grad = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1] \quad \textbf{(18)}$$

Constraint Function:

$$cnstr = [D * F_i, \frac{F_i}{F_{i,max}} - s] \quad \textbf{(19)}$$

Jacobian of the Constraint Function:

$$jac = [D\ 0\ \frac{1}{F_{i,max}}\ -1] \quad \textbf{(20)}$$

Bounds on parameters

$0<Fi<Fimax$ (Fi must be between 0 and Fmax) (21)

$0<s<\infty$ (22)

Constraints

$$D * F_i = M_i \quad \textbf{(23)}$$

$$-\infty < \frac{F_i}{F_{i,max}} - s < 0 \quad \textbf{(24)}$$

The static optimization results are shown in Figure 7 and Figure 8. To demonstrate that the polynomial criterion approaches the minmax criterion, in addition to the typical p=2 and p=3 exponents, 5 log-spaced exponents from 5 to 50 were again analyzed (Figure 8). The relationship between p and activation is not monatomic for soleus. This is indicative, at least for this model, that relatively high polynomial orders are needed if the intent is to use the polynomial criterion to perform a minmax fatigue type of optimization. As previously discussed, it can be seen that the activation in the glut, vast, rectus, and soleus become load leveled.

Due to the formulation of the optimization problem, the minmax criterion does not suffer from scaling problems and becomes a linear optimization problem.

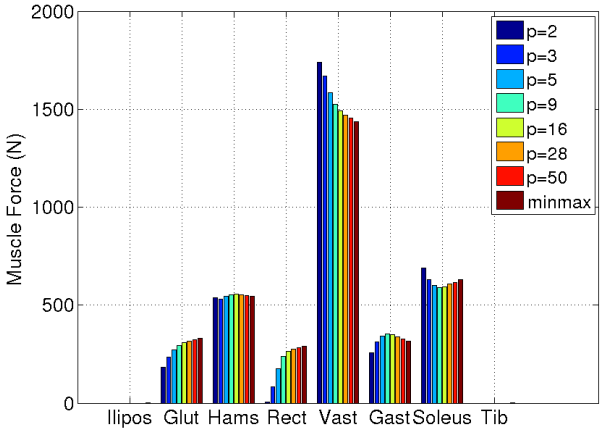

Figure 7. Muscle Force: Minmax Criterion

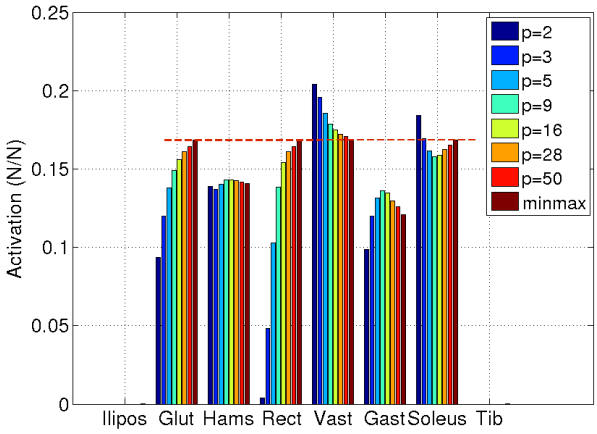

Figure 8. Muscle Activation: Minmax Criterion

C. *Muscle Discretization*

It may not be intuitively obvious that the polynomial objective functions may suffer from a mathematical inconsistency: muscle discretization. If a muscle is split into two parallel muscles (discretized) having the same moment arm, it would seem that it would be appropriate to scale the muscles by dividing their max isometric forces in half. The effect of this incorrect assumption can easily be seen if we replace a single vastus muscle with a max isometric force of $F_{max,vast,init}$ with two muscles with max forces of $F_{max,vast,init}$ /2. As shown in the static optimization results of Table 1, the sum of the forces of the 2 half vasti muscles does not equal the force of the single vastus muscle. Accordingly the results for the other muscles are also effected. Holmberg[8] derived a factor for scaling the muscle when using the polynomial criterion:

$$C_{poly,scale} = 2^{\frac{1-p}{p}} \tag{25}$$

Table 1. Muscle Discretization Effects (Force in N)

	Polynomial (p=2)			*Minmax*	
	Basline 1 Vast Muscle	2 Vast Muscle	2 Vast Muscle	*Baseline* 1 Vast Muscle	2 Vast Muscle
C_{scale}	1	0.5	0.707	1	0.5
Vast Fmax	8,531	4,266	6,031	8,531	4,266
Ilipos	0	0	0	0	0
Glut	184	236	184	331	331
Hams	539	519	539	546	546
Rect	7	59	7	289	289
Vast 1	**1,740**	**820**	**870**	**1,437**	**719**
Vast 2		**820**	**870**		**719**
Gast	256	210	256	314	314
Soleus	688	733	688	629	629
Tib	0	0	0	0	0

While this example is for the simple case of splitting a muscle with the same moment arm (same insertion points), it should be appreciated the additional level of complexity if the muscle insertion points are also changed.

As can be seen in Table 1, the minmax criterion does not require the use of the scaling factor; dividing the muscle max isometric force in half produces vasti muscles forces that are half of the single vastus baseline and the other 7 muscles provide the same force.

D. *Review of OpenSim Static Optimization Algorithm*

OpenSim is capable of utilizing several optimization engines, and ships with IPOPT. The following source code from OpenSim3.2 was reviewed:

- Optimizer.cpp
- InteriorPointOptimizer.cpp
- StaticOptimization.cpp
- StaticOptimizationTarget.cpp

OpenSim static optimization uses the polynomial criterion:

$$G_{poly} = \sum_{i=1}^{m} (A_i)^p \tag{26}$$

Where A is the muscle activation.

The objective function is the minimization of the sum of the muscle (or actuator) activation and the polynomial power, p, is selectable by the user. The user is also able to choose whether the muscle force-length-velocity effects should be included. Including this relationship insures that the results of the static optimization are realizable when forward dynamics is performed. In the previous discussion on the polynomial and minmax criterion, note that the muscle maximum achievable force, $F_{max,I}$ is the maximum isometric force of the muscle. The maximum achievable force though decreases when the muscle is not at optimum length or at high shortening velocities. This ability in OpenSim ensures that static optimization results will not overstate the capability of muscles. The ability to not use the force-length-velocity relationship allows for easier troubleshooting of a static optimization problem that will not converge. In this case, all of the muscles F_{max} values can be increased (by the same proportion) and the problem solved to determine which muscle complexes are being over actuated.

A MATLAB implementation of the OpenSim algorithm can be found in the appendix. As seen in Figure 9, the optimization fails to converge above p=9. When a 10x coefficient was added to the activation, the optimization converged through p=50 (Figure 10).

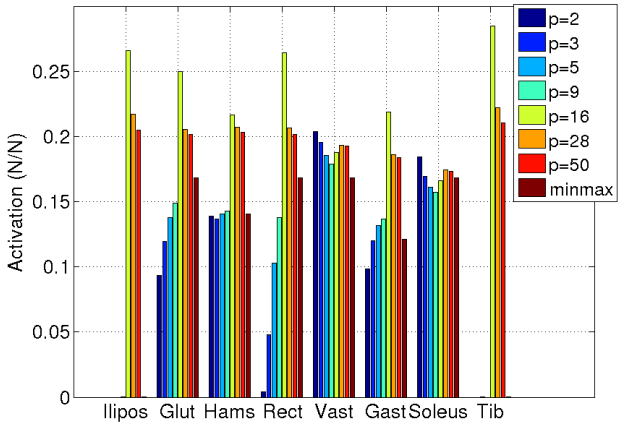

Figure 9. Muscle Activation: Polynomial Criterion and Unscaled Activation Objective Function (26)

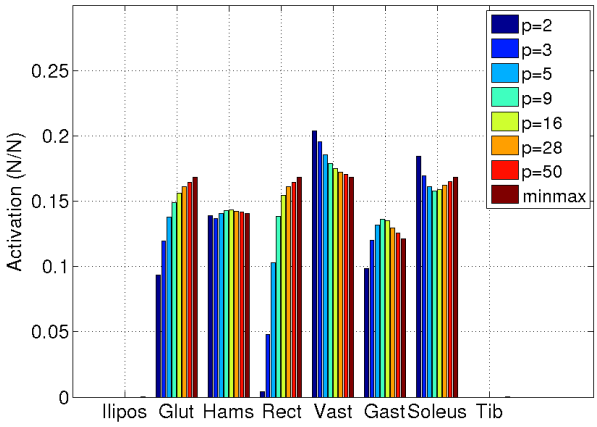

Figure 10. Muscle Activation: Polynomial Criterion and Scaled Activation Objective Function

It has been my experience that OpenSim static optimization is prone to fail to converge when p is greater than 3. There is no prescaling term in OpenSim; activation are directly used. Given that IPOPT seems to not perform scaling for terms that are less than 10e-8, a lightly actuated muscle (0.01) will not be considered in scaling at about a p=4. Moments have been used in the constraint functions developed to support this paper (moments are readily available as an output of Forward Dynamics). OpenSim uses generalized coordinate acceleration in place of moments in static optimization constraints. From a mathematical stand point, accelerations are more prone to scaling issues; relatively small mass bodies such as appendages can exhibit high accelerations at low forces. I believe it would be productive to develop a static optimization analysis (or augment the current OpenSim static optimization analysis tool) to:

1) Add a manual scaling factor to the objective and constraint (advanced user option)
2) Investigate the use of generalized moments and forces in place of accelerations
3) Add minmax criterion to support fatigue like optimization (due to both its numerical stability and for better modeling of fatigue based activities). During this project, the minmax criteria never failed to converge.
4) OpenSim starts with an initial guess of activation for all muscles of 0. As discussed above, gradient scaling is performed by IPOPT utilizing this guess. IPOPT though will need to increase the activations to a non-zero terms to establish the gradient and scaling. The process that IPOPT uses should be further investigated along with the benefits of being able to specify using a default muscle activation other than 0.

IV. Discussion

The implementation of the polynomial and minmax criterion has been investigated and demonstrated using a simple model. The polynomial criterion is useful for effort like optimization when the exponent is small. It can be used for fatigue like optimization with higher exponents but becomes problematic due to scaling considerations. The minmax criterion is numerically much better suited to performing fatigue like optimization and by its nature is better scaled.

The effect of muscle discretization on the polynomial and minmax criteria were also demonstrated. While the polynomial criterion can be corrected using Holmberg's factor, the minmax criterion can be more intuitively used. As the polynomial and minmax criteria represent relatively different optimization paradigms, effort versus fatigue, both have a purpose. Consideration must be given to the type of static optimization that will be used when building or modifying a muscle-skeletal model's properties.

Efforts here have been placed on investigating the scaling of objective functions. Scaling of constraint functions needs to be considered also.

In reviewing the OpenSim static optimization analysis, it may be possible to better scale problems and allow for better convergence. Several possible modifications have been presented. Better scaling not only will provide a better chance to reach convergence, but also can increase the speed to reach convergence for problems that are currently successful.

I came across this quote at the completion of this project, while researching scaling methodologies and believe it reflects the theme of my findings:

"*Scaling is everything. Poor scaling can make a good algorithm bad. Scaling changes the convergence rate, termination tests, and numerical conditioning.*" - John T. Betts, Optimization Author and Guru [9]

References

[1] B. Prilutsky, V. Zatsiorsky, "Optmization-based models of Muscle Coordination," Exercise Sports Science Review, January 2002.

[2] Delp SL, Anderson FC, Arnold AS, Loan P, Habib A, John CT, Guendelman E, Thelen DG. OpenSim: Open-source Software to Create and Analyze Dynamic Simulations of Movement. IEEE Transactions on Biomedical Engineering. (2007)

[3] M. Ackermann, A. van den Bogert, "Optimality principles for model-based prediction of human gait," Journal of Biomechanics 2010 1055-1060

[4] J. Rasmussen, M. Damsgaard, M. Voight, "Muscle recruitment by the min/max criterion – a comparitive numerical study," Journal of Biomechanics 2001 409-415

[5] A. Wächter. An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, January 2002.

[6] Computational Infrastructure for Operations Research, IPOPT On-line Docuemntation: https://projects.coin-or.org/Ipopt/wiki/HintsAndTricks

[7] Computational Infrastructure for Operations Research, Introduction to IPOPT: A tutorial for downloading, installing, and using IPOPT. Revision 2500, August 16, 2014.

[8] L. Holmberg, A. Klarbring, "Muscle decomposition and recruitemnt criteria influence muscle force estimates," Multibody System Dynamics September 2011.

[9] J. Betts, "Practical Methos for Optimal Control and Estimation Using Non-linear Programming," Second Edition, soceiety for Industrial and Applied Mathematics. Pg 45.

```
% makeSummaryPlots - This script runs each of the different criterions
%   and creates plots.  Chane the switch value below to run different
%   criterion.  Change the q values (pre-scaler) and p (exponents)
%   as needed.


clpear, close all, clc

%Muscle Moment Arms (m)
D=[0.050  -0.0620 -0.0720     0.0340      0       0       0       0; ...
    0       0       -0.0340     0.0500      0.0420  -0.0200 0       0; ...
    0       0       0           0           0       -0.0530 -0.0530 0.037];
%Muscle Max Isometric Force (N)
fMax=[1917  1967    3878       1718        8531    2596    3734    1233];

%Target Joint Moments (N-m)
mTarget=[-50;50;-50];

%Initial Guess for Muscle Forces
fo=zeros(1,8);



switch 0   %Change this value to run different static optmizations

    case 0   % Perform SO with Stress Directly
        q=1;  %The pre-scaling coeffecient
        for i=1:9
            [f(:,i),m(:,i),s,info]=z2_polyOptStressScaling(D,fMax, ...
                mTarget,fo,i+1,q);
        end
        legText={'p=2','p=3','p=4','p=5','p=6','p=7','p=8','p=9','p=10'};

    case 1  % Perform SO with polynomial criterion
        q=10;  %The pre-scaling coeffecient
        for i=1:9
            [f(:,i),m(:,i),s,info]=z2_polyOpt(D,fMax,mTarget,fo,i+1,q);
        end
        legText={'p=2','p=3','p=4','p=5','p=6','p=7','p=8','p=9','p=10'};

    case 2  %Perform SO with higher p and with minmax
        q=10; %The pre-scaling coeffecient

        p=[2,3,5,9,16,28,50];
        for i=1:length(p)
            [f(:,i),m(:,i),s,info]=z2_polyOpt(D,fMax,mTarget,fo,p(i),q);
        end
        [fMinmax,mMinmax,s,info]=minMaxOpt(D,fMax,mTarget,fo);
        f=[f,fMinmax];
        m=[m,mMinmax];
```

```
        legText={'p=2','p=3','p=5','p=9','p=16','p=28','p=50','minmax'};


    case 3  %Perform SO with activation directly (OpenSim like)
        q=10;
        p=[2,3,5,9,16,28,50];
        for i=1:length(p)
            [f(:,i),m(:,i),s,info]=z2_polyOptActivation(D,fMax,mTarget, ...
                fo,p(i),q)
        end
        [fMinmax,mMinmax,s,info]=minMaxOpt(D,fMax,mTarget,fo);
        f=[f,fMinmax];
        m=[m,mMinmax];
        legText={'p=2','p=3','p=5','p=9','p=16','p=28','p=50','minmax'};

    case 4  %Perfom SO with maxnorm poly
        q=1;
        p=[2,3,5,9,16,28,50];
        for i=1:length(p)
            [f(:,i),m(:,i),s,info]=z2_polyOptWithOuterExponent(D,fMax, ...
                mTarget,fo,p(i),q)
        end
        legText={'p=2','p=3','p=5','p=9','p=16','p=28','p=50'};
end

% Plot Muscle Force
figure
bar(f,'grouped')
set(gca,'fontsize',22)
a={'Ilipos', 'Glut',  'Hams',    'Rect',   'Vast',  'Gast', 'Soleus','Tib'};
set(gca,'xTickLabel',a)
ylabel('Muscle Force (N)')
set(gcf,'Position',[671 300 929 664])
l=legend(legText)
set(l,'fontsize',20)

%Plot Muscle Activation
for i=1:size(f,2)
    act(:,i)=f(:,i)./fMax';
end
figure
bar(act,'grouped')
set(gca,'ylim',[0 0.3]);
set(gca,'fontsize',22)
set(gca,'xTickLabel',a)
ylabel('Activation (N/N)')
set(gcf,'Position',[671 300 929 664])
l=legend(legText)
set(l,'fontsize',20)

%Plot Resulting Moments
```

```
figure
bar(m,'grouped')
set(gca,'fontsize',22)
a={'Hip','Knee','Ankle'};
set(gca,'xTickLabel',a)
ylabel('Joint Moment (N-m)')
l=legend(legText)
set(l,'fontsize',20)
set(gcf,'Position',[671 300 929 664])
```

```
function  [fOpt,mResult,s,infoOut]=z2_polyOptStressScaling(D,fMax,mTarget,fo,p,q)


%z2_polyOptStressScaling - Perform Static Optimization with Stress Units
%
%
%[fOpt,mResult,s,infoOut]=z2_polyOptStressScaling(D,fMax,mTarget,fo,p,q)
%
%       Inputs:
%           D - Matrix (Joints by Muscle) of moments arms
%           fMax - Max Isometric Force for Each  (row vector)
%           mTarget - Moments at Each Joint (column vector)
%           fo - Initial guess for each force (row vector)
%           p - polynomial power to use
%           q - prescaler value
%
%       Outputs:
%           fOpt - Result Forces
%           mResults - result moments (should match mTarget)
%           s - value of objective function
%           infoOut - info returned from IPOPT



global c


c.D=D;
c.fMax=fMax;
c.pCsa=c.fMax/25e4;
c.n=p;
c.q=q;


% Set the bounds and constraints
options.lb = zeros(1,length(fo));   % Lower bound on the variables.
options.ub = c.fMax;   % Upper bound on the variables.
options.cl = mTarget;    % Lower bounds on the constraint functions.
options.cu = mTarget;    % Upper bounds on the constraint functions.

% Set the IPOPT options
%options.ipopt.jac_c_constant        = 'yes';
options.ipopt.hessian_approximation = 'limited-memory';
%options.ipopt.mu_strategy           = 'adaptive';
options.ipopt.tol                   = 1e-7;

% The callback functions.
funcs.objective         = @objFunc;
funcs.constraints       = @constrFunc;
funcs.gradient          = @gradObjFunc;
funcs.jacobian          = @constraintJac;
```

```
funcs.jacobianstructure = @() sparse(c.D);

%Run IPOPT
[fOpt infoOut] = ipopt(fo,funcs,options);
mResult=constrFunc(fOpt);
s=objFunc(fOpt);

%-------------------------------------------------------
function s=objFunc(f)
%The objective to minimize
%   Input  f: muscle forces as provided by IPOPT

global c
s=sum((f./c.pCsa*c.q).^c.n);

%-------------------------------------------------------
function ds_df=gradObjFunc(f)
%The gradient of the objective
%   Input  f: muscle forces as provided by IPOPT
%   Output ds_df: change in objective with change in force

global c
a=(c.q./c.pCsa).^c.n;
b=c.n.*(f.^(c.n-1));
ds_df=a.*b;

%-------------------------------------------------------
function m=constrFunc(f)
%The constraints
%   Input  f: muscle forces as provided by IPOPT
%   Output m: moments at the joints

global c
m=c.D*f';  %Calculate the moments

%-------------------------------------------------------
function dm_df=constraintJac(f)
%The jacobian (sparse gradient) of the constraints
%   Input  f: muscle forces as provided by IPOPT
%   Output dm_df: change in moments with change in forces

global c
dm_df=sparse(c.D);  %dm_df is just the moment arm
```

```
function  [fOpt,m,s,infoOut]=z2_polyOpt(D,fMax,mTarget,fo,p,q)

%z2_polyOpt - Perform Static Optimization with Force Units and Polynomial
%   Criterion
%
%
%[fOpt,m,s,infoOut]=z2_polyOpt(D,fMax,mTarget,fo,p,q)
%
%       Inputs:
%           D - Matrix (Joints by Muscle) of moments arms
%           fMax - Max Isometric Force for Each  (row vector)
%           mTarget - Moments at Each Joint (column vector)
%           fo - Initial guess for each force (row vector)
%           p - polynomial power to use
%           q - prescaler value
%
%       Outputs:
%           fOpt - Result Forces
%           mResults - result moments (should match mTarget)
%           s - value of objective function
%           infoOut - info returned from IPOPT

if nargin<6
    q=1;
end

global c
c.D=D;
c.fMax=fMax;
c.pCsa=c.fMax/25e4;
c.q=q;
c.n=p;

% Set the bounds and constraints
options.lb = zeros(1,length(fo));   % Lower bound on the variables.
options.ub = c.fMax;   % Upper bound on the variables.
options.cl = mTarget;    % Lower bounds on the constraint functions.
options.cu = mTarget;    % Upper bounds on the constraint functions.

% Set the IPOPT options
%options.ipopt.jac_c_constant        = 'yes';
options.ipopt.hessian_approximation =  'limited-memory';
%options.ipopt.mu_strategy           = 'adaptive';
options.ipopt.tol                   = 1e-7;

% The callback functions.
funcs.objective         = @objFunc;
funcs.constraints       = @constrFunc;
funcs.gradient          = @gradObjFunc;
funcs.jacobian          = @constraintJac;
funcs.jacobianstructure = @() sparse(c.D);
```

```
[f infoOut] = ipopt(fo,funcs,options);  % Run IPOPT
fOpt=f';
m=constrFunc(f);
s=objFunc(f);



%-----------------------------------------------------
function s=objFunc(f)
%The objective to minimize
%   Input  f: muscle forces as provided by IPOPT

global c

s=sum((c.q.*f./c.fMax).^c.n);


%-----------------------------------------------------
function ds_df=gradObjFunc(f)
%The gradient of the objective
%   Input  f: muscle forces as provided by IPOPT
%   Output ds_df: change in stress with change in force

global c
a=c.q.*f./c.fMax;
ds_df=c.n.*(a.^c.n)./f;

%-----------------------------------------------------
function m=constrFunc(f)
%The constraints
%   Input  f: muscle forces as provided by IPOPT
%   Output m: moments at the joints

global c
m=c.D*f';  %Calculate the moments

%-----------------------------------------------------
function dm_df=constraintJac(f)
%The jacobian (sparse gradient) of the constraints
%   Input  f: muscle forces as provided by IPOPT
%   Output dm_df: change in moments with change in forces

global c
dm_df=sparse(c.D);  %dm_df is just the moment arm
```

```
function  [fOpt,mResult,s,infoOut]=z2_polyOptWithOuterExponent(D,fMax,mTarget,fo,p,q)

%z2_polyOptWithOuterExponent - Perform Static Optimization with Force Units
%   and Polynomial Criterion with Maxnorm outer exponent
%
%[fOpt,mResult,s,infoOut]=z2_polyOptWithOuterExponent(D,fMax,mTarget,fo,p,q)
%
%       Inputs:
%           D - Matrix (Joints by Muscle) of moments arms
%           fMax - Max Isometric Force for Each  (row vector)
%           mTarget - Moments at Each Joint (column vector)
%           fo - Initial guess for each force (row vector)
%           p - polynomial power to use
%           q - prescaler value
%
%       Outputs:
%           fOpt - Result Forces
%           mResults - result moments (should match mTarget)
%           s - value of objective function
%           infoOut - info returned from IPOPT

global c
c.D=D;
c.fMax=fMax;
c.pCsa=c.fMax/25e4;
c.q=q;
c.n=p;



% Set the bounds and constraints
options.lb = zeros(1,length(fo));   % Lower bound on the variables.
options.ub = c.fMax;   % Upper bound on the variables.
options.cl = mTarget;    % Lower bounds on the constraint functions.
options.cu = mTarget;    % Upper bounds on the constraint functions.

% Set the IPOPT options
%options.ipopt.jac_c_constant        = 'yes';
options.ipopt.hessian_approximation =  'limited-memory';
%options.ipopt.mu_strategy           = 'adaptive';
options.ipopt.tol                   = 1e-7;

% The callback functions.
funcs.objective         = @objFunc;
funcs.constraints       = @constrFunc;
funcs.gradient          = @gradObjFunc;
funcs.jacobian          = @constraintJac;
funcs.jacobianstructure = @() sparse(c.D);

%Run IPOPT
[fOpt infoOut] = ipopt(fo,funcs,options);
```

```
mResult=constrFunc(fOpt);
s=objFunc(fOpt);

%-------------------------------------------------------
function s=objFunc(f)
%The objective to minimize
%   Input  f: muscle forces as provided by IPOPT

global c
s=(sum((f./c.fMax*c.q).^c.n))^(1/c.n);


%-------------------------------------------------------
function ds_df=gradObjFunc(f)
%The gradient of the objective
%   Input  f: muscle forces as provided by IPOPT

global c
n1=(sum((f./c.fMax*c.q).^c.n)) ^ (1/c.n-1);
n2=c.q * ((f.*c.q./c.fMax) .^ (c.n-1) ) ./ c.fMax;
ds_df=n1.*n2;

%-------------------------------------------------------
function m=constrFunc(f)
%The constraints
%   Input  f: muscle forces as provided by IPOPT
%   Output m: moments at the joints

global c
m=c.D*f';  %Calculate the moments

%-------------------------------------------------------
function dm_df=constraintJac(f)
%The jacobian (sparse gradient) of the constraints
%   Input  f: muscle forces as provided by IPOPT
%   Output dm_df: change in moments with change in forces

global c
dm_df=sparse(c.D);  %dm_df is just the moment arm
```

```
function [fOpt,mResult,s,infoOut]=minMaxOpt(D,fMax,mTarget,fo)

%minMaxOpt - Perform Static Optimization with Minmax
%
%
%[fOpt,mResult,s,infoOut]=minMaxOpt(D,fMax,mTarget,fo)
%
%       Inputs:
%           D - Matrix (Joints by Muscle) of moments arms
%           fMax - Max Isometric Force for Each  (row vector)
%           mTarget - Moments at Each Joint (column vector)
%           fo - Initial guess for each force (row vector)
%
%       Outputs:
%           fOpt - Result Forces
%           mResults - result moments (should match mTarget)
%           s - value of objective function
%           infoOut - info returned from IPOPT

global c

c.D=D;
c.fMax=fMax;

fo=[fo 0];  %Append the slack variable

% Set the bounds and constraints
options.lb = zeros(1,size(fo,2));   % Lower bound on the variables.
options.ub = [c.fMax inf];   % Upper bound on the variables.

nMuscles=length(fo)-1;
infV(1:nMuscles)=-Inf;
z(1:nMuscles)=0;
options.cl = [mTarget;infV'];   % Lower bounds on the constraint functions.
options.cu = [mTarget;z'];    % Upper bounds on the constraint functions.

% Set the IPOPT options
%options.ipopt.jac_c_constant        = 'yes';
options.ipopt.hessian_approximation =  'limited-memory';
%options.ipopt.mu_strategy           = 'adaptive';
options.ipopt.tol                   = 1e-7;

% The callback functions.
funcs.objective         = @objFunc;
funcs.constraints       = @constrFunc;
funcs.gradient          = @gradObjFunc;
funcs.jacobian          = @constraintJac;
sparseMat=constraintJac(fo);
funcs.jacobianstructure = @() sparseMat;
```

```
[fOpt infoOut] = ipopt(fo,funcs,options);

mResult=fOpt(1:end-1)*c.D';
mResult=mResult';
s=constrFunc(fOpt);

fOpt=fOpt(1:end-1)';
%-----------------------------------------------------
function s=objFunc(f)
%The objective to minimize
%   Input  f: muscle forces as provided by IPOPT

s=f(end);


%-----------------------------------------------------
function ds_df=gradObjFunc(f)
%The gradient of the objective
%   Input  f: muscle forces as provided by IPOPT

ds_df=zeros(size(f,1),size(f,2));
ds_df(end)=1;

%-----------------------------------------------------
function m=constrFunc(f)
%The constraints
%   Input  f: muscle forces as provided by IPOPT
%   Output m: moments at the joints and muscle stress (column vector)

global c

s=f(end);  %Max activation slack variable
f=f(1:end-1);  % Forces activation

mAct=f./c.fMax-s;

mMoments=c.D*f';
m=[mMoments;mAct'];

%-----------------------------------------------------
function dm_df=constraintJac(f)
%The jacobian (sparse gradient) of the constraints
%   Input  f: muscle forces as provided by IPOPT
%   Output dm_df: change in moments with change in forces

global c
n=length(f);  % Number of variables



% Stress equations
```

```
dm_dfAct=diag(1./c.fMax);
sVect=-ones(n-1,1);
dm_dfAct=[dm_dfAct sVect];


% Moment equations
nMoments=size(c.D,1);
dm_dfMoments=[c.D zeros(nMoments,1)];

dm_df=sparse([dm_dfMoments;dm_dfAct]);
```

```
function  [fOpt,m,s,infoOut]=z2_polyOptActivation(D,fMax,mTarget,fo,p,q)

%z2_polyOptActivation - Perform Static Optimization with Activation Units
%   and Polynomial Criterion similar to OpenSim
%
%[fOpt,m,s,infoOut]=z2_polyOptActivation(D,fMax,mTarget,fo,p,q)
%
%       Inputs:
%           D - Matrix (Joints by Muscle) of moments arms
%           fMax - Max Isometric Force for Each  (row vector)
%           mTarget - Moments at Each Joint (column vector)
%           fo - Initial guess for each force (row vector)
%           p - polynomial power to use
%           q - prescaler value
%
%       Outputs:
%           fOpt - Result Forces
%           mResults - result moments (should match mTarget)
%           s - value of objective function
%           infoOut - info returned from IPOPT

if nargin<6
    q=1;
end

global c
c.D=D;
c.fMax=fMax;
c.pCsa=c.fMax/25e4;
c.q=q;
c.n=p;

% Set the bounds and constraints
options.lb = zeros(1,length(fo));   % Lower bound on the variables.
options.ub = ones(1,length(fo));    % Upper bound on the variables.
options.cl = mTarget;    % Lower bounds on the constraint functions.
options.cu = mTarget;    % Upper bounds on the constraint functions.

% Set the IPOPT options
%options.ipopt.jac_c_constant        = 'yes';
options.ipopt.hessian_approximation =  'limited-memory';
%options.ipopt.mu_strategy           = 'adaptive';
options.ipopt.tol                   = 1e-7;

% The callback functions.
funcs.objective         = @objFunc;
funcs.constraints       = @constrFunc;
funcs.gradient          = @gradObjFunc;
funcs.jacobian          = @constraintJac;
funcs.jacobianstructure = @() sparse(c.D);
```

```
ao=fo./c.fMax;  %Convert forces to activation

[a infoOut] = ipopt(ao,funcs,options);  %Run IPOPT

f=a.*c.fMax;  %Convert activation to force
fOpt=f'  ;
m=constrFunc(a);
s=objFunc(a);

%-----------------------------------------------------
function s=objFunc(a)
%The objective to minimize
%   Input  f: muscle forces as provided by IPOPT

global c
s=sum((c.q.*a).^c.n);


%-----------------------------------------------------
function ds_df=gradObjFunc(a)
%The gradient of the objective
%   Input  f: muscle forces as provided by IPOPT

global c
ds_df=  c.n.*  ( (c.q.*a).^(c.n-1) );


%-----------------------------------------------------
function m=constrFunc(a)
%The constraints
%   Input  f: muscle forces as provided by IPOPT
%   Output m: moments at the joints

global c
f=a.*c.fMax;
m=c.D*f';  %Calculate the moments

%-----------------------------------------------------
function dm_df=constraintJac(a)
%The jacobian (sparse gradient) of the constraints
%   Input  f: muscle forces as provided by IPOPT
%   Output dm_df: change in moments with change in forces

global c

for i=1:size(c.D,1)
    fD(i,:)=c.D(i,:).*c.fMax;
end
dm_df=sparse(fD);  %dm_df is just the moment arm
```