```
%---------------------------
% Author: Akira Nagamori
% Last update: 6/8/17
% REFERENCE:
% - Muscle model
% He et al. 1991; Brown et al. 1996; Cheng et al. 2000; Song et al.,
2008a,b
% Brown et al., 1999; Brown & Loeb, 2000
% - Muscle spindle model
% Mileusnic et al. 2006
% - GTO model
% Elias et al. 2014
%---------------------------
function output =
AfferentedMuscleModel(muscle_parameter,delay_parameter,gain_parameter,t
argetTrajectory,feedbackOption)
%---------------------------
% Input: user-defined parameters of the model
% Output: data structure containing simulated data
%---------------------------
% Time
Fs = 10000; % sampling frequency
t = 0:1/Fs:(length(targetTrajectory)-1)/Fs;

%---------------------------
% muscle architectural parameters
alpha = muscle_parameter.pennationAngle; % pennation angle
mass = muscle_parameter.mass; % muscle mass
Lm_initial = muscle_parameter.muscleInitialLength; % muscle initial
length
Lt_initial = muscle_parameter.tendonInitialLength; % tendon initial
length
Lmt = Lm_initial*cos(alpha)+Lt_initial; % intial musculotendon length
L0 = muscle_parameter.optimalLength; % optimal muscle length
L_tendon = muscle_parameter.tendonLength; % tendon slack length
L0T = L_tendon*1.05; % optimal tendon length

% calculate maximal force output of the muscle based on PCSA
density = 1.06; % g/cm^3
PCSA = (mass*1000)/density/muscle_parameter.optimalLength;
%physiological cross-sectional area
Specific_Tension = 31.4;
F0 = PCSA * Specific_Tension; % maximal force
Fmax = muscle_parameter.MVIC; % maximal voluntary force determined
empilically
Fbaseline = muscle_parameter.baseline; % baseline force without muscle
activation

% calculate initial length based on balance between stiffness of muscle
and
% tendon
[Lce,Lse,Lmax] =  InitialLength(muscle_parameter);

%---------------------------
% Parameter Initialization
Vce = 0; % muscle excursion velocity
Ace = 0; % muscle excursion acceleration
```

```
ForceSE = Fse(Lse) * F0; % tendon force
u_a_FB = 0; % tracking contoller output

u_a_Ia = zeros(1,length(targetTrajectory)); % Ia output from spindle
model
Ia_input = zeros(1,length(targetTrajectory)); % Ia input to alpha-
motoneuron
u_a_II = zeros(1,length(targetTrajectory));  % II output from spindle
model
II_input = zeros(1,length(targetTrajectory)); % II input to alpha-
motoneuron
x = zeros(1,length(targetTrajectory)); % temporary vector for Ib firing
rate
u_a_Ib_temp = zeros(1,length(targetTrajectory)); % temporary vector for
Ib firing rate
u_a_Ib = zeros(1,length(targetTrajectory)); % Ib output from GTO model
Ib_input = zeros(1,length(targetTrajectory)); % Ib input to alpha-
motoneuron
ND = zeros(1,length(targetTrajectory)); % neural drive to muscle
ND_long = zeros(1,length(targetTrajectory)); % delayed neural drive to
muscle
noise = zeros(1,length(targetTrajectory)); % signal-dependent noise
noise_filt = zeros(1,length(targetTrajectory)); % signal-dependent
noise filtered at 100 Hz

% vectors to store data
OutputForceTendon = zeros(1,length(targetTrajectory));
OutputForceMuscle = zeros(1,length(targetTrajectory));
OutputLse = zeros(1,length(targetTrajectory));
OutputLce = zeros(1,length(targetTrajectory));
OutputVce = zeros(1,length(targetTrajectory));
OutputAce = zeros(1,length(targetTrajectory));
OutputUeff = zeros(1,length(targetTrajectory));
OutputAf_slow = zeros(1,length(targetTrajectory));
OutputAf_fast = zeros(1,length(targetTrajectory));
MuscleAcceleration = zeros(1,length(targetTrajectory));
MuscleVelocity = zeros(1,length(targetTrajectory));
MuscleLength = zeros(1,length(targetTrajectory));
MuscleLength(1) = Lce*L0/100;

%----------------------------
% filter design parameters for noise
% low-pass filter at 100 Hz
[b100,a100] = butter(4,100/(Fs/2),'low');

%----------------------------
% Activation dynamics parameters
Y_dot = 0;
Y = 0;
Saf_dot = 0;
Saf = 0;

fint_dot = 0;
fint = 0;
feff_dot = 0;
feff = 0;
```

```
fint_dot_2 = 0;
fint_2 = 0;
feff_dot_2 = 0;
feff_2 = 0;

Af_slow = 0;
Af_fast = 0;
Ueff = 0;

%----------------------------
% Feedback system parameters
%----------------------------
% Spindle Model (Mileusnic et al. 2006)
p = 2; % power constant relating the fusimotor frequency to activation
R = 0.46; % fascicle length below which force production is zero (L0)
a = 0.3; % nonlinear velocity dependence power constant
K_SR = 10.4649; % sensory region spring constant [FU/L0]
K_PR = 0.15; % polar region spring constant [FU/L0]
M = 0.0002; % intrafusal fiber mass [FU/(L0/s^2)]

LN_SR = 0.0423; % sensory region threshold length (L0)
LN_PR = 0.89; % polar region threshold length (L0)

L0_SR = 0.04; % sensory region rest length (L0)
L0_PR = 0.76; % polar region rest length (L0)

L_secondary = 0.04; % secondary afferent rest length (L0)
X = 0.7; %

% initialing parameters
f_dynamic = 0;
f_static = 0;
T_ddot_bag1 = 0;
T_dot_bag1 = 0;
T_bag1 = 0;
T_ddot_bag2 = 0;
T_dot_bag2 = 0;
T_bag2 = 0;
T_ddot_chain = 0;
T_dot_chain = 0;
T_chain = 0;

%----------------------------
% GTO model (Elias et al. 2014)
G1 = 60; % conversion factor (Hz)
G2 = 4; % conversion factor (N)

% transfer function describing GTO dynamics
s = tf('s');
H = (1.7*s^2+2.58*s+0.4)/(s^2+2.2*s+0.4);
Hd = c2d(H,1/Fs);
[num,den] = tfdata(Hd);
num = cell2mat(num);
den = cell2mat(den);
```

```
%---------------------------
% Gain parameters
%---------------------------
% Tracking controller
K = 0.00035; % gain of tracking controller
Ftarget = targetTrajectory*Fmax; % Convert force trajectory to unit of
newton for tracking controller

% Muscle spindle
gamma_dynamic = gain_parameter.gammaDynamic; % gamma dynamic fusimotor
drive
gamma_static = gain_parameter.gammaStatic; % gamma static fusimotor
drive

Gain_Ia = gain_parameter.Ia; % constant factor to normalize Ia firing
rate into a value between 0 and 1
Gain_II = gain_parameter.II; % constant factor to normalize II firing
rate into a value between 0 and 1

Ia_PC = gain_parameter.Ia_PC; % presynaptic control input for Ia

% GTO
Gain_Ib = gain_parameter.Ib;  % constant factor to normalize Ib firing
rate into a value between 0 and 1

Ib_PC = gain_parameter.Ib_PC; % presynaptic control input for Ib

%---------------------------
% Delay parameters
%---------------------------
% assign delays
delay_efferent = delay_parameter.efferent; % delay along alpha-
motoneuron
delay_Ia = delay_parameter.Ia; % delay along Ia afferent
delay_II = delay_parameter.II; % delay along II afferent
delay_Ib = delay_parameter.Ib; % delay along Ib afferent
delay_tracking = delay_parameter.tracking; % afferent delay for a
tracking controller

% Convert delays in ms to samples
delay_Ia_step = delay_Ia*Fs/1000;
delay_II_step = delay_II*Fs/1000;
delay_Ib_step = delay_Ib*Fs/1000;
delay_c = delay_tracking*Fs/1000;

%---------------------------
% closed-loop simulation of afferent muscle
for i = 1:length(t)

    if feedbackOption == 0 % Feedforward input only
        u_a_FF = Ftarget(i)/Fmax; % feedforward input
        ND_temp = smoothSaturationFunction(u_a_FF); % Neural drive to
muscle
        u_a_Ia(i) = 0; % no Ia feedback
        u_a_II(i) = 0; % no II feedback
        u_a_Ib(i) = 0; % no Ib feedback
```

```
    elseif feedbackOption == 1 % Feedback control (proprioceptive
systems + tracking controller)

        % muscle spindle ouputs
        [OutputPrimary,OutputSecondary] = Spindle(Lce,Vce,Ace);
        u_a_Ia(i) = OutputPrimary; % Ia output
        u_a_II(i) = OutputSecondary; % II output

        % GTO output
        x(i) = G1*log((ForceSE/G2+1)); % Newton to Ib firing rate
        % transfer function implementation
        if i == 1
            u_a_Ib_temp(i) = x(i);
        elseif i == 1
            u_a_Ib_temp(i) = x(i);
        elseif i == 2
            u_a_Ib_temp(i) = (num(1)*x(i))/den(1);
        elseif i == 3
            u_a_Ib_temp(i) = (num(2)*x(i-1) + num(1)*x(i) ...
                - den(2)*u_a_Ib_temp(i-1))/den(1);
        elseif i > 3
            u_a_Ib_temp(i) = (num(3)*x(i-2) + num(2)*x(i-1) +
num(1)*x(i) ...
                - den(3)*u_a_Ib_temp(i-2) - den(2)*u_a_Ib_temp(i-
1))/den(1);
        end
        u_a_Ib(i) = u_a_Ib_temp(i); % Ib output
        if u_a_Ib(i) < 0
            u_a_Ib(i) = 0;
        end


        if i <= 0.2*Fs

            ND_temp = 0; %input to the muscle

        elseif i > 0.2*Fs

            u_a_FB = K*(Ftarget(i)-(OutputForceTendon(i-delay_c)-
Fbaseline))/Fmax + u_a_FB; % feedback input from tracking controller
            Ia_input(i) = smoothSaturationFunction(u_a_Ia(i)/Gain_Ia +
Ia_PC); % Ia input to alpha-motoneuron
            II_input(i) = smoothSaturationFunction(u_a_II(i)/Gain_II);
% II input to alpha-motoneuron
            Ib_input(i) = smoothSaturationFunction(u_a_Ib(i)/Gain_Ib +
Ib_PC); % Ib input to alpha-motoneuron
            % integrate all the inputs
            input_x = Ia_input(i-delay_Ia_step)  ...
                + II_input(i-delay_II_step) ...
                + u_a_FB ...
                - Ib_input(i-delay_Ib_step);
            ND_temp = smoothSaturationFunction(input_x);
        end

    end
```

```
    % generate signal-dependent noise low-pass filtered at 100 Hz
    if i > 5
        noise(i) = 2*(rand(1)-0.5)*(sqrt(0.3*ND_temp)*sqrt(3));
        noise_filt(i) = (b100(5)*noise(i-4) + b100(4)*noise(i-3) +
b100(3)*noise(i-2) + b100(2)*noise(i-1) + b100(1)*noise(i) ...
            - a100(5)*noise_filt(i-4) - a100(4)*noise_filt(i-3) -
a100(3)*noise_filt(i-2) - a100(2)*noise_filt(i-1))/a100(1);
    else
        noise(i) =0;
        noise_filt(i) = noise(i);
    end

    % combine noise and neural drive
    ND(i) = ND_temp + noise_filt(i);

    if ND(i) < 0
        ND(i) = 0;
    elseif ND(i) > 1
        ND(i) = 1;
    end

    % add delay along efferent pathway
    if i > delay_efferent*Fs/1000
        ND_long(i) = ND(i-delay_efferent*10);
    else
        ND_long(i) = 0;
    end

    % activation filter (Song et al. 2008)
    if ND_long(i) >= Ueff
        TU = 0.03;
    elseif ND_long(i) < Ueff
        TU = 0.15;
    end

    Ueff_dot = (ND_long(i) - Ueff)/TU;
    Ueff = Ueff_dot*1/Fs + Ueff; % effective neural drive

    % force from contractile element
    ForceTotal = forceContractile(Lce,Vce,Lmax,Ueff);
    ForceTotal = ForceTotal*F0;

    % force from series elastic element
    ForceSE = Fse(Lse) * F0;

    % calculate muscle excursion acceleration based on the difference
    % between muscle force and tendon force
    MuscleAcceleration(i+1) = (ForceSE*cos(alpha) -
ForceTotal*(cos(alpha)).^2)/(mass) ...
        + (MuscleVelocity(i)).^2*tan(alpha).^2/(MuscleLength(i));
    % integrate acceleration to get velocity
    MuscleVelocity(i+1) = (MuscleAcceleration(i+1)+ ...
        MuscleAcceleration(i))/2*1/Fs+MuscleVelocity(i);
    % integrate velocity to get length
    MuscleLength(i+1) = (MuscleVelocity(i+1)+ ...
```

```
        MuscleVelocity(i))/2*1/Fs+MuscleLength(i);

    % normalize each variable to optimal muscle length or tendon length
    Ace = MuscleAcceleration(i+1)/(L0/100);
    Vce = MuscleVelocity(i+1)/(L0/100);
    Lce = MuscleLength(i+1)/(L0/100);
    Lse = (Lmt - Lce*L0*cos(alpha))/L0T;

    % store data
    OutputForceMuscle(i) = ForceTotal; % muscle force
    OutputForceTendon(i) = ForceSE; % tendon force
    OutputLse(i) = Lse; % normalized tendon length
    OutputLce(i) = Lce; % normalized muscle length
    OutputVce(i) = Vce; % normalized muscle excursion velocity
    OutputAce(i) = Ace; % normalized muscle excursion acceleration
    OutputUeff(i) = Ueff; % effective neural drive to muscle
    OutputAf_slow(i) = Af_slow; % activation-frequency relationship for
slow-twitch fiber
    OutputAf_fast(i) = Af_fast; % activation-frequency relationship for
fast-twitch fiber

end

%---------------------------
% plot output
figure()
plot(t,OutputForceTendon)
hold on
plot(t,Ftarget,'r')
legend('Output Force','Target Force')
xlabel('Time(sec)','Fontsize',14)
ylabel('Force(N)','Fontsize',14)
%---------------------------
% save data as output in structure format
output.Force = OutputForceMuscle; % muscle force
output.ForceTendon = OutputForceTendon; % tendon force
output.Target = Ftarget; % target force trajectory
output.Lce = OutputLce; % muscle length
output.Vce = OutputVce; % muscle velocity
output.Ace = OutputAce; % muscle acceleration
output.Lse = OutputLse; % tendon length
output.ND = ND; % nueral drive
output.ND_long = ND_long; % delayed neural drive
output.noise = noise; % signal dependent noise
output.noise_filt = noise_filt; % noise low-filtered at 100Hz
output.U = OutputUeff; % effective neural drive
output.Ia = u_a_Ia; % Ia afferent output
output.II = u_a_II; % II afferent output
output.Ib = u_a_Ib; % Ib afferent output
output.Af_slow = OutputAf_slow; % activation-frequency relationship for
slow-twitch fiber
output.Af_fast = OutputAf_fast; % activation-frequency relationship for
fast-twitch fiber

% below are functions used in the model
%---------------------------
% muscle spindle model (Mileusnic et al. 2006)
```

```
    function AP_bag1 = bag1_model(L,L_dot,L_ddot)
        %---------------------------
        % afferent potential for bag 1 fiber
        % input: muscle length, velocity and acceleration
        % output: afferent potential for primary ending
        %---------------------------
        tau_bag1 = 0.149; % low-pass filter time constant (s)
        freq_bag1 = 60; % constant relating the fusimotor frequency to
activation

        beta0 = 0.0605; % coef. of damping due to dynamic fusimotor
input [FU/(L0/s)]
        beta1 = 0.2592; % coef. of damping due to static fusimotor
input [FU/(L0/s)]
        Gamma1 = 0.0289; % coef. of force generation due to dynamic
fusimotor input [FU]

        G = 20000;  % Term relating the sensory region's stretch to
afferent firing

        if L_dot >= 0 % lengthening
            C = 1; % Coef. of asymmetry in F-V curve
        else % shortening
            C = 0.42;
        end

        % convert fusimotor frequency (gamma_dynamic)
        % fusimotor activation level (f_dynamic)
        df_dynamic = (gamma_dynamic^p/(gamma_dynamic^p+freq_bag1^p)-
f_dynamic)/tau_bag1;
        f_dynamic = 1/Fs*df_dynamic + f_dynamic;

        beta = beta0 + beta1 * f_dynamic; % polar region's damping term
        Gamma = Gamma1 * f_dynamic; % active force generator term

        % tension in the sensory and polar regions
        T_ddot_bag1 = K_SR/M * (C * beta * sign(L_dot-
T_dot_bag1/K_SR)*((abs(L_dot-T_dot_bag1/K_SR))^a)*(L-L0_SR-T_bag1/K_SR-
R)+K_PR*(L-L0_SR-T_bag1/K_SR-L0_PR)+M*L_ddot+Gamma-T_bag1);
        T_dot_bag1 = T_ddot_bag1*1/Fs + T_dot_bag1;
        T_bag1 = T_dot_bag1*1/Fs + T_bag1;

        % convert tension to afferent potentials for primary endings
        AP_bag1 = G*(T_bag1/K_SR-(LN_SR-L0_SR));
    end

    function [AP_primary_bag2,AP_secondary_bag2] =
bag2_model(L,L_dot,L_ddot)
        %---------------------------
        % afferent potential for bag 2 fiber
        % input: muscle length, velocity and acceleration
        % output: afferent potential for primary and secondary endings
        %---------------------------
        tau_bag2 = 0.205; % low-pass filter time constant (s)
        freq_bag2 = 60; % constant relating the fusimotor frequency to
activation
```

```
        beta0 = 0.0822; % coef. of damping due to dynamic fusimotor
input [FU/(L0/s)]
        beta2 = -0.046; % coef. of damping due to static fusimotor
input [FU/(L0/s)]
        Gamma2 = 0.0636; % coef. of force generation due to dynamic
fusimotor input [FU]

        G = 10000; % Term relating the sensory region's stretch to
afferent firing

        if L_dot >= 0
            C = 1;  % Coef. of asymmetry in F-V curve
        else
            C = 0.42;
        end

        % convert fusimotor frequency (gamma_static)
        % fusimotor activation level (f_static)
        df_static = (gamma_static^p/(gamma_static^p+freq_bag2^p)-
f_static)/tau_bag2;
        f_static = 1/Fs*df_static + f_static;

        beta = beta0 + beta2 * f_static; % polar region's damping term
        Gamma = Gamma2 * f_static;% active force generator term

        % tension in the sensory and polar regions
        T_ddot_bag2 = K_SR/M * (C * beta * sign(L_dot-
T_dot_bag2/K_SR)*((abs(L_dot-T_dot_bag2/K_SR))^a)*(L-L0_SR-T_bag2/K_SR-
R)+K_PR*(L-L0_SR-T_bag2/K_SR-L0_PR)+M*L_ddot+Gamma-T_bag2);
        T_dot_bag2 = T_ddot_bag2*1/Fs + T_dot_bag2;
        T_bag2 = T_dot_bag2*1/Fs + T_bag2;

        % convert tension to afferent potentials for primary and
secondary
        % endings
        AP_primary_bag2 = G*(T_bag2/K_SR-(LN_SR-L0_SR));
        AP_secondary_bag2 = G*(X*L_secondary/L0_SR*(T_bag2/K_SR-(LN_SR-
L0_SR))+(1-X)*L_secondary/L0_PR*(L-T_bag2/K_SR-L0_SR-LN_PR));

    end

    function [AP_primary_chain,AP_secondary_chain] =
chain_model(L,L_dot,L_ddot)
        %---------------------------
        % afferent potential for chain fiber
        % input: muscle length, velocity and acceleration
        % output: afferent potential for primary and secondary endings
        %---------------------------
        freq_chain = 90;  % constant relating the fusimotor frequency
to activation

        beta0 = 0.0822; % coef. of damping due to dynamic fusimotor
input [FU/(L0/s)]
        beta2_chain = - 0.069; % coef. of damping due to static
fusimotor input [FU/(L0/s)]
```

```
        Gamma2_chain = 0.0954; % coef. of force generation due to
dynamic fusimotor input [FU]

        G = 10000; % Term relating the sensory region's stretch to
afferent firing

        if L_dot >= 0
            C = 1; % Coef. of asymmetry in F-V curve
        else
            C = 0.42;
        end

        % convert fusimotor frequency (gamma_static)
        % fusimotor activation level (f_static_chain)
        f_static_chain = gamma_static^p/(gamma_static^p+freq_chain^p);

        beta = beta0 + beta2_chain * f_static_chain; % polar region's
damping term
        Gamma = Gamma2_chain * f_static; % active force generator term

        % tension in the sensory and polar regions
        T_ddot_chain = K_SR/M * (C * beta * sign(L_dot-
T_dot_chain/K_SR)*((abs(L_dot-T_dot_chain/K_SR))^a)*(L-L0_SR-
T_chain/K_SR-R)+K_PR*(L-L0_SR-T_chain/K_SR-L0_PR)+M*L_ddot+Gamma-
T_chain);
        T_dot_chain = T_ddot_chain*1/Fs + T_dot_chain;
        T_chain = T_dot_chain*1/Fs + T_chain;

        % convert tension to afferent potentials for primary and
secondary
        % endings
        AP_primary_chain = G*(T_chain/K_SR-(LN_SR-L0_SR));
        AP_secondary_chain = G*(X*L_secondary/L0_SR*(T_chain/K_SR-
(LN_SR-L0_SR))+(1-X)*L_secondary/L0_PR*(L-T_chain/K_SR-L0_SR-LN_PR));

    end

    function [OutputPrimary,OutputSecondary] = Spindle(Lce,Vce,Ace)
        %---------------------------
        % afferent firing model
        % input: muscle length, velocity and acceleration
        % output: Ia and II afferent firing rate
        % (OutputPrimary,OutputSecondary)
        %---------------------------
        S = 0.156; % amount of partial occlusion
        AP_bag1 = bag1_model(Lce,Vce,Ace);
        [AP_primary_bag2,AP_secondary_bag2] = bag2_model(Lce,Vce,Ace);
        [AP_primary_chain,AP_secondary_chain] =
chain_model(Lce,Vce,Ace);

        if AP_bag1 < 0
            AP_bag1 = 0;
        end

        if AP_primary_bag2 < 0
            AP_primary_bag2 = 0;
```

```
        end

        if AP_primary_chain < 0
            AP_primary_chain = 0;
        end


        if AP_secondary_bag2 < 0
            AP_secondary_bag2 = 0;
        end

        if AP_secondary_chain < 0
            AP_secondary_chain = 0;
        end

        % compare afferent potential of bag1 to the sum of afferent
potentials of bag 2 and chain
        if AP_bag1 > (AP_primary_bag2+AP_primary_chain)
            Larger = AP_bag1;
            Smaller = AP_primary_bag2+AP_primary_chain;
        else
            Larger = AP_primary_bag2+AP_primary_chain;
            Smaller = AP_bag1;
        end
        % Ia afferent firing rate
        OutputPrimary = Larger + S * Smaller;
        % II afferent firing rate
        OutputSecondary = AP_secondary_bag2 + AP_secondary_chain;

        % bound firing rates between 0 and 100000
        if OutputPrimary < 0
            OutputPrimary = 0;
        elseif OutputPrimary > 100000
            OutputPrimary = 100000;
        end
        if OutputSecondary < 0
            OutputSecondary = 0;
        elseif OutputSecondary > 100000
            OutputSecondary = 100000;
        end
    end

%----------------------------
% musculotendon model (He et al. 1991; Brown et al. 1996; Cheng et al.
% 2000; Song et al., 2008a,b)
    function FL = FL(L)
        %---------------------------
        % force length (F-L) relationship for slow-twitch fiber
        % input: normalized muscle length and velocity
        % output: F-L factor (0-1)
        %---------------------------
        beta = 2.3;
        omega = 1.12;
        rho = 1.62;

        FL = exp(-abs((L^beta - 1)/omega)^rho);
    end
```

```
    function FL = FL_fast(L)
        %----------------------------
        % force length (F-L) relationship for fast-twitch fiber
        % input: normalized muscle length and velocity
        % output: F-L factor (0-1)
        %----------------------------
        beta = 1.55;
        omega = 0.75;
        rho = 2.12;

        FL = exp(-abs((L^beta - 1)/omega)^rho);
    end

    function FVcon = FVcon(L,V)
        %----------------------------
        % concentric force velocity (F-V) relationship for slow-twitch
fiber
        % input: normalized muscle length and velocity
        % output: F-V factor (0-1)
        %----------------------------
        Vmax = -7.88;
        cv0 = 5.88;
        cv1 = 0;

        FVcon = (Vmax - V)/(Vmax + (cv0 + cv1*L)*V);
    end

    function FVcon = FVcon_fast(L,V)
        %----------------------------
        % concentric force velocity (F-V) relationship for fast-twitch
fiber
        % input: normalized muscle length and velocity
        % output: F-V factor (0-1)
        %----------------------------
        Vmax = -9.15;
        cv0 = -5.7;
        cv1 = 9.18;

        FVcon = (Vmax - V)/(Vmax + (cv0 + cv1*L)*V);
    end

    function FVecc = FVecc(L,V)
        %----------------------------
        % eccentric force velocity (F-V) relationship for slow-twitch
fiber
        % input: normalized muscle length and velocity
        % output: F-V factor (0-1)
        %----------------------------
        av0 = -4.7;
        av1 = 8.41;
        av2 = -5.34;
        bv = 0.35;
        FVecc = (bv - (av0 + av1*L + av2*L^2)*V)/(bv+V);
    end

    function FVecc = FVecc_fast(L,V)
```

```
        %---------------------------
        % eccentric force velocity (F-V) relationship for fast-twitch
fiber
        % input: normalized muscle length and velocity
        % output: F-V factor (0-1)
        %---------------------------
        av0 = -1.53;
        av1 = 0;
        av2 = 0;
        bv = 0.69;
        FVecc = (bv - (av0 + av1*L + av2*L^2)*V)/(bv+V);
    end

    function Fpe1 = Fpe1(L,V)
        %---------------------------
        % passive element 1
        % input: normalized muscle length
        % output: passive element force (0-1)
        %---------------------------
        c1_pe1 = 23;
        k1_pe1 = 0.046;
        Lr1_pe1 = 1.17;
        eta = 0.01;

        Fpe1 = c1_pe1 * k1_pe1 * log(exp((L - Lr1_pe1)/k1_pe1)+1) +
eta*V;

    end

    function Fpe2 = Fpe2(L)
        %---------------------------
        % passive element 2
        % input: normalized muscle length
        % output: passive element force (0-1)
        %---------------------------
        c2_pe2 = -0.02;
        k2_pe2 = -21;
        Lr2_pe2 = 0.70;

        Fpe2 = c2_pe2*exp((k2_pe2*(L-Lr2_pe2))-1);

    end

    function Fse = Fse(LT)
        %---------------------------
        % series elastic element (tendon)
        % input: tendon length
        % output: tendon force (0-1)
        %---------------------------
        cT_se = 27.8; %27.8
        kT_se = 0.0047;
        LrT_se = 0.964;

        Fse = cT_se * kT_se * log(exp((LT - LrT_se)/kT_se)+1);

    end
```

```
    function Fce = forceContractile(L,V,Lmax,Ueff)
        %---------------------------
        % Force output from contractile elements
        % input: muscle length, velocity, maximal muscle length,
effective
        % neural drive
        % output: muscle force
        %---------------------------

        % (Song et al. 2008)
        Ur = 0.8; % activation level at which all the motor units are
recruited (Song et al. 2008)
        U1_th = 0.001; % threshold for slow-twitch fiber
        dif_U_1 = Ueff - U1_th; % difference between effective neural
drive and threshold for slow-twitch fiber
        U2_th = Ur*0.6; % threshold for fast-twitch fiber
        dif_U_2 = Ueff - U2_th; % difference between effective neural
drive and threshold for fast-twitch fiber
        if dif_U_2 < 0
            dif_U_2 = 0;
        end

        W1 = dif_U_1/(dif_U_1+dif_U_2); % proportion of active slow-
twitch fiber of total active muscle (0-1)
        W2 = dif_U_2/(dif_U_1+dif_U_2); % proportion of active fast-
twitch fiber of total active muscle (0-1)

        % activation-frequency relationship (Brown and Loeb 2000)
        f_half = 8.5; % frequency at which the motor unit produces half
of its maximal isometric force
        fmin = 0.5*f_half; % minimum firing frequency of slow-twitch
fiber
        fmax = 2*f_half; % maximum firing frequency of slow-twitch
fiber
        % constants for slow-twitch fiber
        af = 0.56;
        nf0 = 2.11;
        nf1 = 5;
        cy = 0.35;
        Vy = 0.1;
        Ty = 0.2;

        Tf1 = 0.0343;
        Tf2 = 0.0227;
        Tf3 = 0.047;
        Tf4 = 0.0252;

        % Y = yielding factor for slow-twitch fiber
        Y_dot = (1 - cy*(1-exp(-abs(V)/Vy))-Y)./Ty;
        Y = Y_dot*1/Fs + Y;

        % firing frequency input to second-order excitation dynamics of
        % slow-twitch fiber
        fenv = (fmax-fmin)/(1-U1_th).*(Ueff-U1_th)+fmin;
        fenv = fenv/f_half;
```

```
        if feff_dot >= 0
            Tf = Tf1 * L^2 + Tf2 * fenv; % time constant for second-
order excitation dynamics
        elseif feff_dot < 0
            Tf = (Tf3 + Tf4*Af_slow)/L;
        end

        % intermediate firing frequency of second-order excitation
dynamics
        % of slow-twitch fiber (f_half)
        fint_dot = (fenv - fint)/Tf;
        fint = fint_dot*1/Fs + fint;
        % effective firing frequency of slow-twitch fiber (f_half)
        feff_dot = (fint - feff)/Tf;
        feff = feff_dot*1/Fs + feff;

        if feff < 0
            feff = 0;
        end

        % activation-frequency relationship for slow-twitch fiber
        nf = nf0 + nf1*(1/L-1);
        Af_slow = 1 - exp(-((Y*feff/(af*nf))^nf));


        f_half_2 = 34;% frequency at which the motor unit produces half
of its maximal isometric force
        fmin_2 = 0.5*f_half_2; % minimum firing frequency of fast-
twitch fiber
        fmax_2 = 2*f_half_2; % maximum firing frequency of fast-twitch
fiber
        % constants for fast-twitch fiber
        af_2 = 0.56;
        nf0_2 = 2.1;
        nf1_2 = 3.3;
        as1_2 = 1.76;
        as2_2 = 0.96;
        Ts_2 = 0.043;

        Tf1_2 = 0.0206;
        Tf2_2 = 0.0136;
        Tf3_2 = 0.0282;
        Tf4_2 = 0.0151;

        % firing frequency input to second-order excitation dynamics of
        % fast-twitch fiber
        fenv_2 = (fmax_2-fmin_2)/(1-U2_th).*(Ueff-U2_th)+fmin_2;
        fenv_2 = fenv_2/f_half_2;

        if feff_dot_2 >= 0
            Tf_2 = Tf1_2 * L^2 + Tf2_2 * fenv_2; % time constant for
second-order excitation dynamics
        elseif feff_dot_2 < 0
            Tf_2 = (Tf3_2 + Tf4_2*Af_fast)/L;
        end
```

```matlab
        % Sagging factor (Saf) for fast-twitch fiber
        if feff_2 < 0.1
            as_2 = as1_2;
        elseif feff_2 >= 0.1
            as_2 = as2_2;
        end

        Saf_dot = (as_2 - Saf)/Ts_2;
        Saf = Saf_dot*1/Fs + Saf;

        % intermediate firing frequency of second-order excitation
dynamics
        % of fast-twitch fiber (f_half)
        fint_dot_2 = (fenv_2 - fint_2)/Tf_2;
        fint_2 = fint_dot_2*1/Fs + fint_2;
        % effective firing frequency of fast-twitch fiber (f_half)
        feff_dot_2 = (fint_2 - feff_2)/Tf_2;
        feff_2 = feff_dot_2*1/Fs + feff_2;

        if feff_2 < 0
            feff_2 = 0;
        end

        % activation-frequency relationship for fast-twitch fiber
        nf_2 = nf0_2 + nf1_2*(1/L-1);
        Af_fast = 1 - exp(-((Saf*feff_2/(af_2*nf_2))^nf_2));

        % force-velocity relationship
        if V <= 0 % concentric
            FV1 = FVcon(L,V);
            FV2 = FVcon_fast(L,V);
        elseif V > 0 % eccentric
            FV1 = FVecc(L,V);
            FV2 = FVecc_fast(L,V);
        end

        % force-length relationship
        FL1 = FL(L);
        FL2 = FL_fast(L);

        % passive element 1
        FP1 = Fpe1(L/Lmax,V);
        % passive element 2
        FP2 = Fpe2(L);
        if FP2 > 0
            FP2 = 0;
        end

        FCE1 = FL1*FV1 + FP2;
        FCE2 = FL2*FV2 + FP2;
        % activation dependent force of contractile elements
        Fce_temp = Ueff*(W1*Af_slow*FCE1+W2*Af_fast*FCE2);
        if Fce_temp < 0
            Fce_temp = 0;
        end
```

```
        % total force from contractile element
        Fce = Fce_temp + FP1;

    end

    function [Lce_initial,Lse_initial,Lmax] =
InitialLength(muscle_parameter)
        %---------------------------
        % Determine the initial lengths of muscle and tendon and
maximal
        % muscle length
        %---------------------------

        % serires elastic element parameters
        cT = 27.8;
        kT = 0.0047;
        LrT = 0.964;
        % parallel passive element parameters
        c1 = 23;
        k1 = 0.046;
        Lr1 = 1.17;

        % passive force produced by parallel passive element at maximal
        % muscle length
        PassiveForce = c1 * k1 * log(exp((1 - Lr1)/k1)+1);
        % tendon length at the above passive force
        Normalized_SE_Length = kT*log(exp(PassiveForce/cT/kT)-1)+LrT;

        % maximal musculotendon length defined by joint range of motion
        Lmt_temp_max =
muscle_parameter.optimalLength*cos(muscle_parameter.pennationAngle) ...
            +muscle_parameter.tendonLength + 1;

        % optimal muscle length
        L0_temp = muscle_parameter.optimalLength;
        % optimal tendon length (Song et al. 2008)
        L0T_temp = muscle_parameter.tendonLength*1.05;

        % tendon length at maximal muscle length
        SE_Length =  L0T_temp * Normalized_SE_Length;
        % maximal fasicle length
        FasclMax = (Lmt_temp_max - SE_Length)/L0_temp;
        % maximal muscle fiber length
        Lmax = FasclMax/cos(muscle_parameter.pennationAngle);

        % initial musculotendon length defined by the user input
        Lmt_temp = muscle_parameter.muscleInitialLength *
cos(muscle_parameter.pennationAngle) +
muscle_parameter.tendonInitialLength;

        % initial muscle length determined by passive muscle force and
        % tendon force
        InitialLength =  (Lmt_temp-(-L0T_temp*(kT/k1*Lr1-LrT-
kT*log(c1/cT*k1/kT))))/(100*(1+kT/k1*L0T_temp/Lmax*1/L0_temp)*cos(muscl
e_parameter.pennationAngle));
        % normalize the muscle legnth to optimal muscle length
```

```
        Lce_initial = InitialLength/(L0_temp/100);
        % calculate initial length of tendon and normalize it to
optimal
        % tendon length
        Lse_initial = (Lmt_temp -
InitialLength*cos(muscle_parameter.pennationAngle)*100)/L0T_temp;
    end

    function y = smoothSaturationFunction(x)
        %---------------------------
        % Non-linear input-output relationship of a population of
neurons (Tsiabis et al. 2014)
        %---------------------------
        y = 1./(1+exp(-11.*(x-0.5)));
    end

end
```