Mathematical Model for Muscular Movement

Penelope Megan Ellis
Advisor: Dr. Pamela Lockwood

Senior Thesis Project for Partial Fulfillment of Requirements
for Graduation from the Attebury Honors Program
Major: Mathematics
Spring 2008

PURPOSE

The purpose of this project is to apply the particle swarm to a system of differential equations modeling the motion of a human's elbow joint due to tendon forces applied by both the biceps and the triceps muscles. Movement at the joints is caused by muscular contractions, which, in turn, are caused by neural impulses firing for a certain period of time. The distance a muscle contracts is proportional to the period of time that a neuron fires. Using particle swarm theory, we generate a math model that accepts as input the duration of a neural impulse. Given the duration of a neural impulse, the model will indicate how far the forearm should move. The particle swarm program will find the neural impulses that produce the desired arm movement while minimizing the muscle fatigue in the biceps and triceps.

MUSCLE MOVEMENT

The biceps and triceps muscles work together in the movement of a person's arm and are located in the upper portion of the arm. The biceps muscle (or "biceps brachii") is responsible for movements such as flexing the elbow and rotating the forearm. The muscle is made up of two bundles of muscle that have two different insertion points at the scapula and a shared insertion point below the elbow. The triceps muscle (or "triceps brachii") is responsible for such movement as pulling the forearm back to an extended position. Like the biceps muscle, the triceps muscle has three bundles of muscle that have different insertion points at the top of the arm and a single insertion point near the elbow.

MATHEMATICAL MODEL OF ELBOW MOVEMENT

A dynamic model describing the motion of the human forearm developed in the master's thesis of Doug Meador, Texas Tech University, was used to create the computer model simulating elbow movement. A system of differential equations was used to model the muscular dynamics. This system is as follows:

where is the derivative of *x*. The function '*x*' is defined as follows:

.

The information that the elbow equations modeled, as shown above in x(t), respectively, were the angle of the elbow, the velocity of angle movement, the tendon force in the biceps and triceps, the length and velocity of contraction of the biceps and triceps, and the neural activation of both the biceps and triceps. The differential equation information was programmed into MatLab, and comparisons were made between the graphs in the thesis and the graphs that were generated in our MatLab files to ensure the equations were correct. The complete MatLab program can be found in the appendix.

PARTICLE SWARM THEORY

Particle swarm optimization is a theory developed by Dr. Eberhart and Dr. Kennedy that involves using a number of random particles and running them through a mathematical equation called a fitness function. A fitness function is used to decide which particles accomplish a set task the best. For example, if one wanted to find the minimum value of the function , we would use as the fitness function and choose which of the random particles created the smallest value for this function.

The basic process for how the particle swarm works is fairly simple. First, the program initializes the particles it is using. Then, it calculates the fitness value for each particle and determines whether that value is better or worse than that of previous particles in its group. If it is better, it sets the new particle as 'best' for the local group. This step is repeated to determine whether or not the particle has the best fitness value for a whole set of random particles that have been previously used. The best particle of these is called the 'global best' particle. The particle that is labeled as global best is the one that fits the mathematical equation best.

The fitness function that was used for this program came from a thesis written by Collin Witherspoon and has two components. One component finds the neural innervations that minimize muscle fatigue in a person's elbow while the other insures the elbow moves in the desired trajectory.

Witherspoon's thesis states that "Crowninshield and Brand identified [a] physiological relationship between muscle fatigue and the sum of cubed muscle stresses."

The first component of a fitness function is dealt with through this relationship. The mathematical equation is as follows:

$$—$$

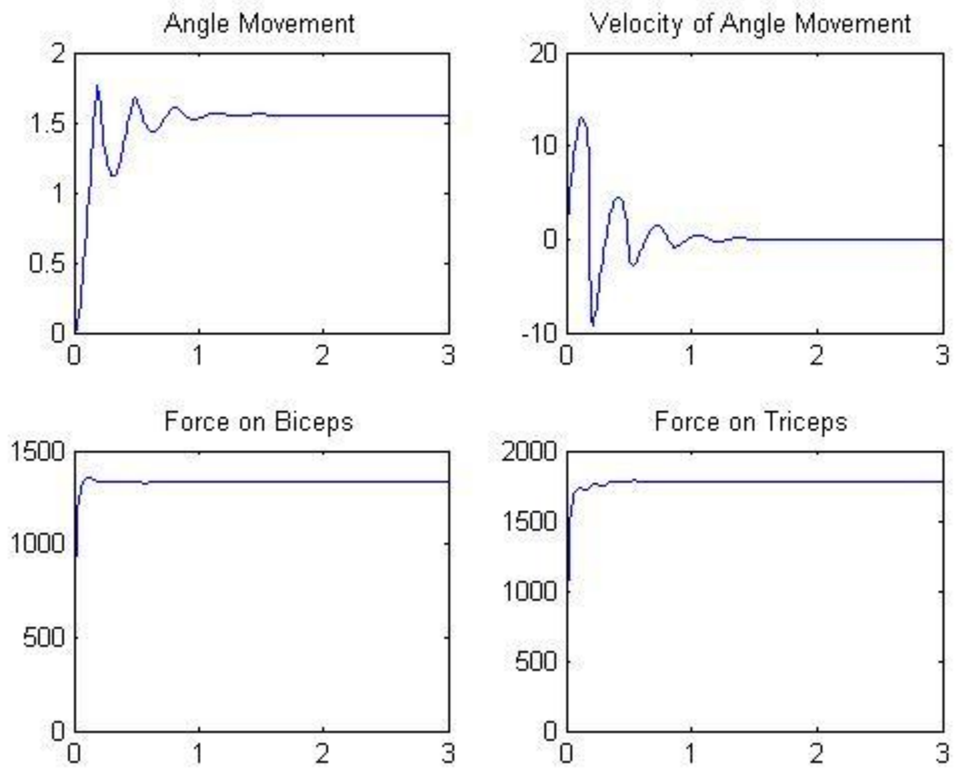
T stands for the internal tension that the muscle produces, A is the cross-sectional area of the muscle, and k is the total number of muscles involved. So, this equation can be interpreted for our project by saying we would take the tension created by the bicep, divide it by the cross-sectional area of the bicep, and cube the result. We would then follow the same steps for the triceps and take the sum these outcomes. This would give us the total fatigue on the system. In a previous paragraph, steps were given for finding the ‘global best’ particle. This particle stands for the optimal neural innervations necessary for minimizing this fatigue function.

The second component is a bit trickier to handle. One thing a person must be careful of is making sure the program allows for muscle movement. Since a person receives the least fatigue in their muscles when there is absolutely no muscle movement at all, the desired answer for a fitness function would be zero movement! However, the purpose of this program is to show the minimized muscle fatigue when there is actual movement due to neural innervations. Therefore, a second equation must be added to the fitness function that actually forces the elbow to move from a stationary position.

The MatLab programs for the Particle Swarm and the Fitness Function can be found in the Appendix.

CONCLUSION

Graphs that model the angle movement, angle velocity, force on the biceps, and force on the triceps were very helpful in determining whether or not the particle swarm was working correctly. Samples of these graphs from the particle swarm are as follows:



The upper graphs labeled ‘Angle Movement’ and ‘Velocity of Angle Movement’ were actually quite close to what they were supposed to be according to the thesis used as a basis for this project. In real life, the model we worked toward began with the forearm at a ninety degree angle to the upper arm with the upper arm resting at the body’s side. We worked in radians and labeled this position ‘zero radians.’ The angle formed by a person with their arm straight down their body was ninety degrees, or about 1.57 radians. We also worked with the difference between the movement of a person’s arm depending on whether they were holding a five kilogram weight or were simply moving their arm. These particular graphs represent a person holding a five kilogram weight.

As is evident by the graph, the program showed arm movement from zero radians to straight down the person's side with a little bouncing at the end. This bouncing was caused from the force of the arm in the downward direction. As the forearm was moving, the force actually caused the forearm to bounce in the opposite direction once it reached its fully outstretched angle. This is the reason for the 'wiggles' in the velocity graph as well. Since the velocity is a derivative of the angle movement with respect to time, the effect of the arm bouncing at the bottom of the fall was shown in both graphs. The units used for velocity were radians per second.

The lower graphs differed greatly from what we were supposed to see. In the left-most graph labeled 'Force on Biceps' (tendon force), there was supposed to be a notable force change, but it was only supposed to reach eighty Newtons, while our graph showed forces reaching upwards of 1400 Newtons. Also, once the arm reached its resting angle of 1.57 radians, the force should have dropped to zero radians. The right-hand graph labeled 'Force on Triceps' (tendon force), should have remained at zero Newtons through the whole experiment. There is obviously a problem since the force reaches around 1750 Newtons.

Unfortunately, we were not able to come to any definitive conclusions by the end of the semester when this project came to an end. Every time we ran the particle swarm, the tendon forces were all too high, indicating a problem with the program. Hopefully these issues will be resolved in future projects.

APPENDIX 1: MATLAB FILES FOR MOVEMENT OF ELBOW

The information used to write this program in MatLab was gained from the thesis written by Doug Meador.

```
function dy=elbow_Pen(t,y)
global  neural
dy=zeros(10,1);

%This is the differential equation for the whole model. (pg 34)
% 1 = Biceps and 2 = Triceps

%constants
m=1.43;
l=.333;
c=.165;
I=.0575;
M_m1=.432;
B_m1=150;
M_m2=.357;
B_m2=150;
T_act1=.01;
beta1=.2;
T_act2=.01;
beta2=.2;

J=m*(l-c)^2+I;

dy(1)=y(2);
dy(2)=(1/J)*(N(y(1))-r_1(y(1))*y(3)-r_2(y(1))*y(4)+M_p(y(1),y(2)));
dy(3)=K_t1(y(3))*(v_tm1(y(1),y(2))-y(7));
dy(4)=K_t2(y(4))*(v_tm2(y(1),y(2))-y(8));
dy(5)=y(7);
dy(6)=y(8);
dy(7)=(1/M_m1)*(y(3)-(F_act1(y(9),y(5),y(7))+F_pe1(y(5))+B_m1*y(7)));
dy(8)=(1/M_m2)*(y(4)-(F_act2(y(10),y(6),y(8))+F_pe2(y(6))+B_m2*y(8)));
dy(9)=(-1/T_act1)*(beta1+(1-
beta1)*n_t1PS(t,neural(1),neural(2),neural(3)))*y(9)+(1/T_act1)*n_t1PS(t,neural(1),neural(2),neural(3));
dy(10)=(-1/T_act2)*(beta2+(1-
beta2)*n_t2PS(t,neural(4),neural(5),neural(6)))*y(10)+(1/T_act2)*n_t2PS(t,neural(4),neural(5),neural(6));
```

```
function dy=elbow_prior(t,y)
global  tspan
dy=zeros(10,1);

%This is the differential equation for the whole model. (pg 34)
% 1 = Biceps and 2 = Triceps

%constants
m=1.43;
```

```
l=.333;
c=.165;
I=.0575;
M_m1=.432;
B_m1=150;
M_m2=.357;
B_m2=150;
T_act1=.01;
beta1=.2;
T_act2=.01;
beta2=.2;

J=m*(l-c)^2+I;

dy(1)=y(2);
dy(2)=(1/J)*(N(y(1))-r_1(y(1))*y(3)-r_2(y(1))*y(4)+M_p(y(1),y(2)));
dy(3)=K_t1(y(3))*(v_tm1(y(1),y(2))-y(7));
dy(4)=K_t2(y(4))*(v_tm2(y(1),y(2))-y(8));
dy(5)=y(7);
dy(6)=y(8);
dy(7)=(1/M_m1)*(y(3)-(F_act1(y(9),y(5),y(7))+F_pe1(y(5))+B_m1*y(7)));
dy(8)=(1/M_m2)*(y(4)-(F_act2(y(10),y(6),y(8))+F_pe2(y(6))+B_m2*y(8)));
dy(9)=(-1/T_act1)*(beta1+(1-beta1)*n_t1_old(t))*y(9)+(1/T_act1)*n_t1_old(t);
dy(10)=(-1/T_act2)*(beta2+(1-beta2)* n_t2_old(t))*y(10)+(1/T_act2)* n_t2_old(t);
```

```
function f=F_act1(a_1, l_m1, v_m1)

%This is the activation force for the bicep. (pg 19)
%For the equation to not equal 0, the muscle length must be between .075 and .225.
%l_m1 and v_m1 are normalized in equations for f_l1 and f_v1.

F_max1=1950;

f=a_1*F_max1*f_l1(l_m1)*f_vC1(v_m1);
```

```
function f=F_act2(a_2, l_m2, v_m2)

%This is the activation force for the tricep. (pg 19)
%For the equation to not equal 0, the muscle length must be between .051 and .153.
%l_m2 and v_m2 are normalized in equations for f_l2 and f_v2.

F_max2=2200;

f=a_2*F_max2*f_l2(l_m2)*f_vC2(v_m2);
```

```
function f=f_l1(l_m1)

%This is the force-length equation for the bicep. (pg 88)
%For the equation to not equal 0, the muscle length must be between .075 and .225.

l_opt1=.15;

nlm=l_m1/l_opt1;

if (.5 <= nlm) & (nlm <= 1.5)
   f=1-4*(nlm-1)^2;
else
   f=0;
end;
```

```
function f=f_l2(l_m2)

%This is the force-length equation for the tricep. (pg 88)
%For the equation to not equal 0, the muscle length must be between .051 and .153.

l_opt2=.102;

nlm=l_m2/l_opt2;

if (.5 <= nlm) & (nlm <= 1.5)
   f=1-4*(nlm-1)^2;
else
   f=0;
end;
```

```
function fpe=F_pe1(l_m1)

%This is the passive force for the bicep. (pg 24)

l_opt1=.15;

nlm=l_m1/l_opt1;

if (1 <= nlm) & (nlm <= 1.4)
   fpe=0.00291621*(exp(13.1123*(nlm-1))-1);
else if (nlm > 1.4)
   fpe=7.25*nlm-9.6;
else
   fpe=0;
end;
end;
```

```
function fpe=F_pe2(l_m2)

%This is the passive force for the tricep. (pg 24)

l_opt2=.102;

nlm=l_m2/l_opt2;

if (1 <= nlm) & (nlm <= 1.4)
   fpe=0.00291621*(exp(13.1123*(nlm-1))-1);
else if (nlm > 1.4)
   fpe=7.25*nlm-9.6;
else
   fpe=0;
end;
end;
```

```
function f=f_v1(v_m1)

%This is the velocity force for the bicep. (pg 89)

l_opt1=.15;
v_max1=7*l_opt1;

nvm=v_m1/v_max1;

f=1+atan(1.5574*nvm);
```

```
function f=f_v2(v_m2)

%This is the velocity force for the tricep. (pg 89)

l_opt2=.102;
v_max2=7*l_opt2;

nvm=v_m2/v_max2;

f=1+atan(1.5574*nvm);
```

```
function k=K_t1(F_t1)

%This is the piecewise spring force coefficient for the bicep. (pg 23)

F_max1=1950;
l_ts1=.198;

nFt=F_t1/F_max1;

if (nFt < 0)
```

```
    k=0;
else if (0 <= nFt) && (nFt <= .5124)
    k=(F_max1/l_ts1)*40.84209*(nFt+0.405671);
else
    k=(F_max1/l_ts1)*37.5;
end;
end;
```

```
function k=K_t2(F_t2)

%This is the piecewise spring force coefficient for the tricep. (pg 23)

F_max2=2200;
l_ts2=.1793;

nFt=F_t2/F_max2;

if (nFt < 0)
    k=0;
else if (0 <= nFt) && (nFt <= .5124)
    k=(F_max2/l_ts2)*40.84209*(nFt+0.405671);
else
    k=(F_max2/l_ts2)*37.5;
end;
end;
```

```
function m=M_p(th, vth)

%This is the passive moment. (pg 27)
%I used the constants from pg 96 in his Mpf equation.
%c is different in this equation than it is for the whole DE.

%I got the constants th_min and th_max from pg 28.

k_1=10;
k_2=25;
k_3=1;
k_4=25;
th_min=-3*pi/10;
th_max=pi/2;
c=.7;

m=k_1*exp(-k_2*(th-th_min))-k_3*exp(-k_4*(th_max-th))-c*vth;
```

```
function n=N(th)

%This is part of dy(2). (pg 32)
%M_w stands for the weight of a mass that a person would be holding.
%This paper used both 0 kg and 5 kg for the M_w, so I have tested with both.

m=1.43;
l=.333;
c=.165;
g=9.81;
M_w=5;

n=m*(l-c)*g*cos(th)+M_w*l*g*cos(th);
```

```
function n=n_t1_old(t)

%This determines the step-input function for neural innervation - lateral rectus
%Initial equilibrium innervation of 12%, then 100%, finally new equilibrium of 10%

t_i=0;
t_1=0.1;
t_2=0.10000001;
t_3=0.3;
t_4=0.30000001;

n_1=0;
n_2=1;
n_3=0;

k_1=(n_2 - n_1)/(t_2 - t_1);
k_2=(n_3 - n_2)/(t_4 - t_3);

if (t_i <= t) && (t <= t_1)
  n=n_1;
else if (t_1 < t) && (t < t_2)
  n=k_1*(t - t_2) + n_2;
else if (t_2 <= t) && (t <= t_3)
  n=n_2;
else if (t_3 < t) && (t < t_4)
  n=k_2*(t - t_4) + n_3;
else
  n=n_3;
end;
end;
end;
end;
```

```
function n=n_t1PS(t)

%This determines the step-input function for neural innervations - lateral rectus
%Initial equilibrium innervations of 12%, then 100%, finally new equilibrium of 10%

t_i=0;
t_1=0.1;
t_2=0.10000001;
t_3=0.3;
t_4=0.30000001;
n_1=0.4430;
n_2=0.1411;
n_3=0.3777;
k_1=(n_2 - n_1)/(t_2 - t_1);
k_2=(n_3 - n_2)/(t_4 - t_3);

if (t_i <= t) && (t <= t_1)
   n=n_1;
else if (t_1 < t) && (t < t_2)
   n=k_1*(t - t_2) + n_2;
else if (t_2 <= t) && (t <= t_3)
   n=n_2;
else if (t_3 < t) && (t < t_4)
   n=k_2*(t - t_4) + n_3;
else
   n=n_3;
end;
end;
end;
end;
```

```
function n=n_t2_old(t)

%This determines the step-input function for neural innervations - triceps
%Initial equilibrium innervations of 12%, then 2%, finally new equilibrium of 10%

t_i=0;
t_1=0.1;
t_2=0.10000001;
t_3=0.3;
t_4=0.30000001;

n_1=0;
n_2=0;
n_3=0;

k_1=(n_2 - n_1)/(t_2 - t_1);
k_2=(n_3 - n_2)/(t_4 - t_3);

if (t_i <= t) && (t <= t_1)
   n=n_1;
```

```
else if (t_1 < t) && (t < t_2)
   n=k_1*(t - t_2) + n_2;
else if (t_2 <= t) && (t <= t_3)
   n=n_2;
else if (t_3 < t) && (t < t_4)
   n=k_2*(t - t_4) + n_3;
else
   n=n_3;
end;
end;
end;
end;
```

```
function n=n_t2PS(t)

%This determines the step-input function for neural innervations - triceps
%Initial equilibrium innervations of 12%, then 2%, finally new equilibrium of 10%

t_i=0;
t_1=0.1;
t_2=0.10000001;
t_3=0.3;
t_4=0.30000001;
n_1=0.0761;
n_2=0.3057;
n_3=0.1460;
k_1=(n_2 - n_1)/(t_2 - t_1);
k_2=(n_3 - n_2)/(t_4 - t_3);

if (t_i <= t) && (t <= t_1)
   n=n_1;
else if (t_1 < t) && (t < t_2)
   n=k_1*(t - t_2) + n_2;
else if (t_2 <= t) && (t <= t_3)
   n=n_2;
else if (t_3 < t) && (t < t_4)
   n=k_2*(t - t_4) + n_3;
else
   n=n_3;
end;
end;
end;
end;
```

```
function r=r_1(th)

%This is the moment arm equation for the bicep. (pg 29)

r=(pi/180)*((-2.9883*10^(-8))*((180/pi)*(pi/2-th))^3+(1.8047*10^(-6))*((180/pi)*(pi/2-th))^2+(4.5322*10^(-
4))*((180/pi)*(pi/2-th))+0.014660);
```

```
function r=r_2(th)

%This is the moment arm equation for the triceps. (pg 29)

r=(pi/180)*((-3.5171*10^(-12))*((180/pi)*(pi/2-th))^5+(13.277*10^(-10))*((180/pi)*(pi/2-th))^4+(-
19.092*10^(-8))*((180/pi)*(pi/2-th))^3+(12.886*10^(-6))*((180/pi)*(pi/2-th))^2+(-3.0284*10^(-
4))*((180/pi)*(pi/2-th))-0.023287);
```

```
function solver

global tspan

%tspan is vector of times equation will be solved for [initial time: timestep: final time]
%Y0 is vector of initial conditions Y1(0)=? and Y2(0)=?, etc...
%Page 37 lets us know that the function is EXTREMELY sensitive to initial conditions.
%I got these initial conditions from page 36.

tspan=[0:0.01:3];

Y0=[0; 0; 0; 0; .1248828696; .110530630185; 0; 0; 0; 0];

[t,Y]=ode45(@elbow_prior, tspan, Y0);

subplot(2,2,1); plot(t,Y(:,1))
subplot(2,2,2); plot(t,Y(:,2))
subplot(2,2,3); plot(t,Y(:,3))
subplot(2,2,4); plot(t,Y(:,4))

Fitness_Pen(Y(:,3), Y(:,4), Y(:,1))
```

```
function v=v_tm1(th, vth)

%This is the velocity for the total muscle length of the bicep.
%This was derived by hand from the equation on pg 89.

v=(-4*vth*(5.2156*10^(-10))*((180/pi)*(pi/2-th))^3 + 3*vth*(3.1498*10^(-8))*((180/pi)*(pi/2-th))^2 +
2*vth*(7.9101*10^(-6))*((180/pi)*(pi/2-th)) + vth*2.5587*10^(-4));
```

```
function v=v_tm2(th, vth)

%This is the velocity for the total muscle length of the tricep.
%This was derived by hand from the equation on pg 90.

v=-6*vth*(6.1385*10^(-14))*((180/pi)*(pi/2-th))^5 + 5*vth*(2.3174*10^(-11))*((180/pi)*(pi/2-th))^4 -
4*vth*(3.3321*10^(-9))*((180/pi)*(pi/2-th))^3 + 3*vth*(2.2491*10^(-7))*((180/pi)*(pi/2-th))^2 –
2*vth*(5.2856*10^(-6))*((180/pi)*(pi/2-th)) – vth*(4.0644*10^(-4));
```

APPENDIX 3: MATLAB FILES FOR PARTICLE SWARM

The information used to write this program in MatLab was gained from the Swarm Intelligence website. We used two programs to test the model for the Particle Swarm, and both models are shown below.

```
function [PS,PSF]=Particle_Swarm_Elbow
global  neural

%n is the number of particles we will be using
%m is the number of variables we will be using (we will probably use 6)

clear
n = 1;
m = 6;
w = 0.9;
c1 = 2;
c2 = 2;
V_min=-124.54;
V_max=124.54;
LocalBestX=zeros(n,m);
GlobalBestX=zeros(m);
fitnessX=zeros(n);

for i=1:1:n
for j=1:1:m
   N(i,j)=random('unif',-147,0);
   X(i,j)=1/(1 + exp(-N(i,j)/32));
   V(i,j)=random('unif',-124.54,124.54);
end;
end;

%initialize the global and local fitness to the worst possible values

GlobalBestFitness=8*10^30;

for i=1:1:n
   LocalBestFitness(i)=8*10^30;
end;

 %loop until convergence, and we will initially chose a finite number of
%iterations

for k=1:1:1  %Put number of interations of particle swarm here!!!!

   for i=1:1:n

%Now it is time to solve the ODE for the elbow.
```

```
%_____________________________________________________________________________

tspan=[0:0.01:0.5];

Y0=[pi/2; 0; 12; 12; .1248828696; .110530630185; 0; 0; X(i,1); X(i,4)];
neural=X(i,:);
[t,Y]=ode45(@elbow, tspan, Y0);

%_____________________________________________________________________________

    fitnessX(i)=Fitness(1,Y(:,3), Y(:,4))
  end;
  for i=1:1:n
    if (fitnessX(i) < LocalBestFitness(i))
      LocalBestFitness(i)=fitnessX(i);
      LocalBestX(i,:)=X(i,:);
    end;
  end;
  for i=1:1:n
    if (LocalBestFitness(i) < GlobalBestFitness)
      GlobalBestFitness=LocalBestFitness(i);
      GlobalBestX=X(i,:);
    end;
  end;
  for i=1:1:n
    for j=1:1:m
      r1=random('unif',0,1);
      r2=random('unif',0,1);
      V(i,j)=w*V(i,j) + c1*r1*(LocalBestX(i,j) - X(i,j)) + c2*r2*(GlobalBestX(j) - X(i,j));
%This if-then statement ensures that the particles do not wander off to far from a
%good velocity.
      if (V(i,j) > V_max)
        V(i,j)=V_max;
      else if (V(i,j) < V_min)
        V(i,j)=V_min;
      else
        V(i,j)=V(i,j);
      end;
      end;
      N(i,j)=X(i,j) + V(i,j);
      X(i,j)=1/(1 + exp(-N(i,j)/32))
    end;
  end;
end;

PS=GlobalBestX
PSF=GlobalBestFitness
```

```
function [PS,PSF]=Particle_Swarm_Elbow_Pen

clear all

%n is the number of particles we will be using
%m is the number of variables we will be using (we will probably use 6)

n = 50;
m = 6;
w = 0.9;
c1 = 2;
c2 = 2;
V_min=-124.54;
V_max=124.54;
LocalBestX=zeros(n,m);
GlobalBestX=zeros(m);
fitnessX=zeros(n);

for i=1:1:n
for j=1:1:m
   N(i,j)=random('unif',-147,0);
   X(i,j)=1/(1 + exp(-N(i,j)/32));
   V(i,j)=random('unif',-124.54,124.54);
end;
end;

%initialize the global and local fitness to the worst possible values

GlobalBestFitness=8*10^30;

for i=1:1:n
   LocalBestFitness(i)=8*10^30;
end;

%loop until convergence, and we will initially chose a finite number of
%iterations

for k=1:1:50  %Put number of interations of particle swarm here!!!!

%Now it is time to solve the ODE for the elbow.

%____________________________________________________________________________

tspan=[0:0.01:3];

Y0=[0; 0; 0; 0; .1248828696; .110530630185; 0; 0; X(i,1); X(i,4)];
global neural
neural=X(i,:);
[t,Y]=ode45(@elbow, tspan, Y0);
%____________________________________________________________________________

   for i=1:1:n
      fitnessX(i)=Fitness_Pen(Y(:,3), Y(:,4), Y(:,1));
```

```
  end;
  for i=1:1:n
    if (fitnessX(i) < LocalBestFitness(i))
      LocalBestFitness(i)=fitnessX(i);
      LocalBestX(i,:)=X(i,:);
    end;
  end;
  for i=1:1:n
    if (LocalBestFitness(i) < GlobalBestFitness)
      GlobalBestFitness=LocalBestFitness(i);
      GlobalBestX=X(i,:);
    end;
  end;
  for i=1:1:n
    for j=1:1:m
      r1=random('unif',0,1);
      r2=random('unif',0,1);
      V(i,j)=w*V(i,j) + c1*r1*(LocalBestX(i,j) - X(i,j)) + c2*r2*(GlobalBestX(j) - X(i,j));
%This if-then statement ensures that the particles do not wander off to far from a
%good velocity.
      if (V(i,j) > V_max)
        V(i,j)=V_max;
      else if (V(i,j) < V_min)
        V(i,j)=V_min;
      else
        V(i,j)=V(i,j);
      end;
      end;
      X(i,j)=X(i,j) + V(i,j);
      X(i,j)=1/(1 + exp(-X(i,j)/32));
    end;
  end;
end;

subplot(2,2,1); plot(t,Y(:,1))
subplot(2,2,2); plot(t,Y(:,2))
subplot(2,2,3); plot(t,Y(:,3))
subplot(2,2,4); plot(t,Y(:,4))

PS=GlobalBestX
PSF=GlobalBestFitness
```

APPENDIX 2: MATLAB FILES FOR FITNESS FUNCTION

The information used to write this program in MatLab was gained from the thesis written by Collin Witherspoon. We used two fitness functions in conjunction with the two Particle Swarm programs to test different theories.

```
function F=Fitness(t, Ft_bic, Ft_tri)
global tspan

%This is our fitness function related to muscle stress.

k=length(tspan);
Ft_bic=zeros(k);
Ft_tri=zeros(k);
A_bic=.0005;  %This is an approximate value for the cross-sectional area of the bicep.
A_tri=.0008;  %This is an approximate value for the cross-sectional area of the triceps.
F=zeros(k);

F(1)=0;
p_hat(t,tspan)
for j=1:1:p_hat(t,tspan)
   F(j+1) = F(j)+(Ft_bic(j+1)/A_bic)^3 + (Ft_tri(j+1)/A_tri)^3;
end;
```

```
function F=Fitness_Pen(Ft_bic, Ft_tri, theta)
global tspan

%This is our fitness function related to muscle stress.

k=length(tspan);

A_bic=.0005;  %This is an approximate value for the cross-sectional area of the bicep.
A_tri=.0008;  %This is an approximate value for the cross-sectional area of the triceps.
F=zeros(k);

theta_actual=[theta(1); theta(11); theta(12); theta(31); theta(32); theta(51)];
theta_desired=[0; 0; 0; pi/2; pi/2; pi/2];

angle_fitness=10^30*sum((theta_actual-theta_desired).^2);
F = sum((Ft_bic/A_bic).^3 + (Ft_tri/A_tri).^3) + angle_fitness;
```

```
function p=p_hat(t,time_partition)

k=length(time_partition);
p=0;
sum=0;
for j=1:1:k
   p=sum+q(t,time_partition(j));
```

```
    sum=p;
end;
```

```
function q=q(t,part)

if t>=part
    q=1;
else
    q=0;
end;
```

BIBLIOGRAPHY

Meador, Doug. "The Dynamics and Control of a Planar Articulating Segmental Model." M.S. thesis. Texas Tech University, August, 2000.

Robleto, Robert A., Jr. "An Analysis of the Musculotendon Dynamics of Hill-Based Models." M.S. thesis. Texas Tech University, 1997.

Swarm Intelligence. 08 August 2008. http://www.swarmintelligence.org/tutorials.php

Wikipedia. 5 May 2008. http://en.wikipedia.org/wiki/Biceps_brachii_muscle.

Wikipedia. 23 April 2008. http://en.wikipedia.org/wiki/Triceps.

Wikipedia. 27 April 2008. http://en.wikipedia.org/wiki/Particle_swarm.

Witherspoon, Collin. "A Method to Determine Optimal Neural Innervation for Controlling the Rotation of the Human Eye." M.S. thesis. West Texas A&M University, 2005.