Custom Muscle Model Part Two

The steps covered in part two are:

To illustrate the new muscle model, we will modify the tug-of-war example described in Performing a Simulation so that a FatigableMuscle pulls against a Millard2012EquilibriumMuscle. We will run a ten-second forward dynamics simulation to understand the effects of fatigue by plotting quantities such as muscle force, fiber lengths, and activation of the two muscles from the simulation

Modified Tug of War Example

The source file *mainFatigue.cpp* contains the main() function from the tug-of-war simulation, with a few modifications so we can use it with our new muscle model. The source file in the *MuscleExample* folder already contains all of the necessary modifications, but here is a description of how it was changed.

First we need to make to the original example is to use a FatigableMuscle and a Millard2012EquilibriumMuscle. After including the header file, *FatigableMuscle.h*, at the top of *mainFatigue.cpp*, we create instances of the FatigableMuscle and Millard2012EquilibriumMuscle muscles and add them to the model. We also simplify the control of the models and apply two constant excitation functions (one to each muscle).

```
// Define the initial and final simulation times
double initialTime = 0.0;
double finalTime = 10.0;
...

// Create two new muscles
double maxIsometricForce = 1000.0, optimalFiberLength = 0.2,
        tendonSlackLength = 0.1,    pennationAngle = 0.0,
        fatigueFactor = 0.30, recoveryFactor = 0.20;


// fatigable muscle (Millard2012EquilibriumMuscle with fatigue)
FatigableMuscle* fatigable = new FatigableMuscle("fatigable",
        maxIsometricForce, optimalFiberLength, tendonSlackLength,
        pennationAngle, fatigueFactor, recoveryFactor);

// original muscle model (muscle without fatigue)
Millard2012EquilibriumMuscle* original = new Millard2012EquilibriumMuscle("original",
        maxIsometricForce, optimalFiberLength, tendonSlackLength,
        pennationAngle);

// Define the path of the muscles
fatigable->addNewPathPoint("fatigable-point1", ground, Vec3(0.0, halfLength, -0.35));
fatigable->addNewPathPoint("fatigable-point2", *block, Vec3(0.0, halfLength, -halfLength));
original->addNewPathPoint("original-point1", ground, Vec3(0.0, halfLength, 0.35));
original->addNewPathPoint("original-point2", *block, Vec3(0.0, halfLength, halfLength));

// Define the default states for the two muscles
// Activation
fatigable->setDefaultActivation(0.01);
original->setDefaultActivation(0.01);

// Fiber length
fatigable->setDefaultFiberLength(optimalFiberLength);
original->setDefaultFiberLength(optimalFiberLength);

// Add the two muscles (as forces) to the model
osimModel.addForce(fatigable);
osimModel.addForce(original);


//////////////////////////////////
// DEFINE CONTROLS FOR THE MODEL //
//////////////////////////////////

// Create a prescribed controller that simply supplies controls as
// a function of time.
// For muscles, controls are normalized motor-neuron excitations
PrescribedController *muscleController = new PrescribedController();
muscleController->setActuators(osimModel.updActuators());

// Set the prescribed muscle controller to use the same muscle control function for each muscle
muscleController->prescribeControlForActuator("fatigable", new Constant(1.0));
muscleController->prescribeControlForActuator("original", new Constant(1.0));
```

Attaching a Muscle Analysis and Visualizing the Model

To understand the effect of muscle fatigue on the tug-of-war results, attach a MuscleAnalysis to the model. This will report the intrinsic attributes of the muscles during the forward simulation including the fiber lengths, fiber velocities, active and passive fiber forces, as well as whole muscle-tendon length and forces. You can also set the model to visualize during the forward simulation.

```
// Add a Muscle analysis
MuscleAnalysis* muscAnalysis = new MuscleAnalysis(&osimModel);
Array<std::string> coords(jointCoordinateSet[5].getName(),1);
muscAnalysis->setCoordinates(coords);
muscAnalysis->setComputeMoments(false);
osimModel.addAnalysis(muscAnalysis);

// Turn on the visualizer to view the simulation run live.
osimModel.setUseVisualizer(false);
```

Now initialize the model in preparation to simulate. Note, that we removed contact and constraints from the tugOfWar example and we lock the block's Y-coordinate (index 4) to so that it translates only in the ground plane.

```
// Init coords to 0 and lock the rotational degrees of freedom so the block doesn't twist
CoordinateSet& coordinates = osimModel.updCoordinateSet();
coordinates[0].setValue(si, 0);
coordinates[1].setValue(si, 0);
coordinates[2].setValue(si, 0);
coordinates[3].setValue(si, 0);
coordinates[4].setValue(si, 0);
coordinates[5].setValue(si, 0);
coordinates[0].setLocked(si, true);
coordinates[1].setLocked(si, true);
coordinates[2].setLocked(si, true);
coordinates[4].setLocked(si, true); // don't let the block fall through or rise off the ground
```

Finally, we update the output file names and output the MuscleAnalysis results.

```
// Save the simulation results
// Save the states
manager.getStateStorage().print("tugOfWar_fatigue_states.sto");

// Save the forces
reporter->getForceStorage().print("tugOfWar_forces.mot");

// Save the muscle analysis results
IO::makeDir("FatigueResults");
muscAnalysis->printResults("tugOfWar", "FatigueResults");

// Save the OpenSim model to a file
osimModel.print("tugOfWar_fatigue_model.osim");
```

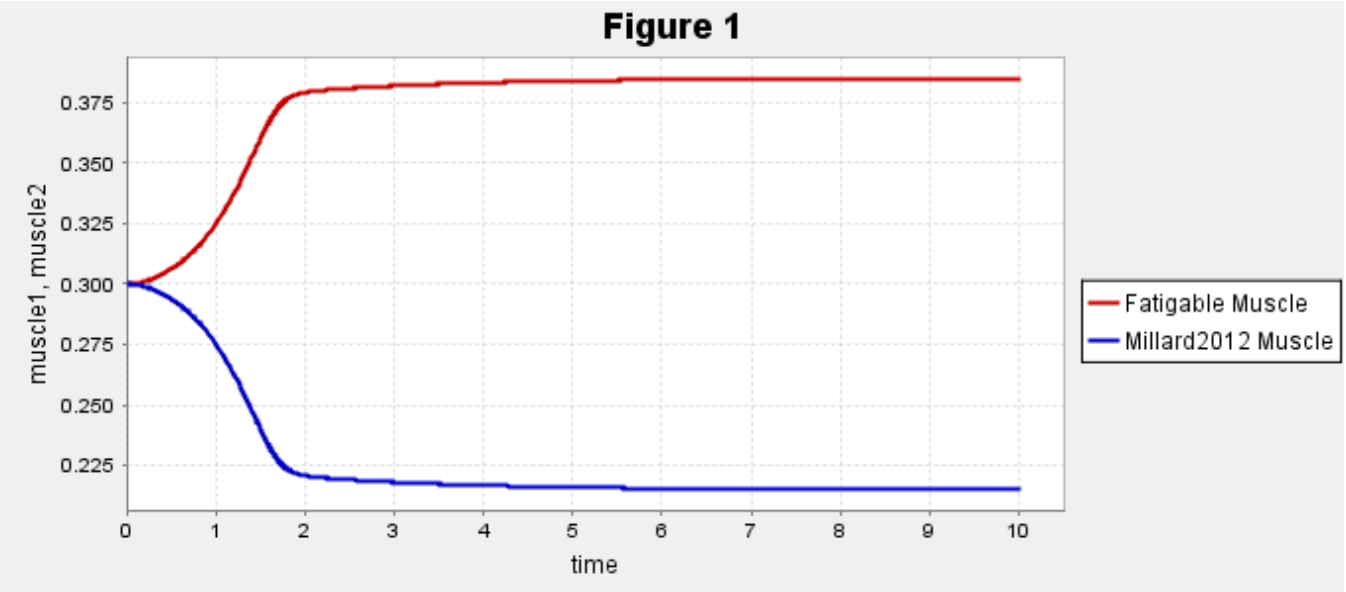
Analyzing the Fatigue Effect

After compiling and running the simulation, we can load the model into the OpenSim GUI and plot muscle lengths to analyze the fatigue effect. The model that is used by the example program is saved to the file *tugOfWar_fatigue_model.osim*. This model contains a Millard2012Equilibrium muscle and a *FatigableMuscle* . However, because *FatigableMuscle* is a new muscle model that OpenSim does not know about, we cannot load that model into OpenSim without first creating a plug-in containing the *FatigableMuscle* class.

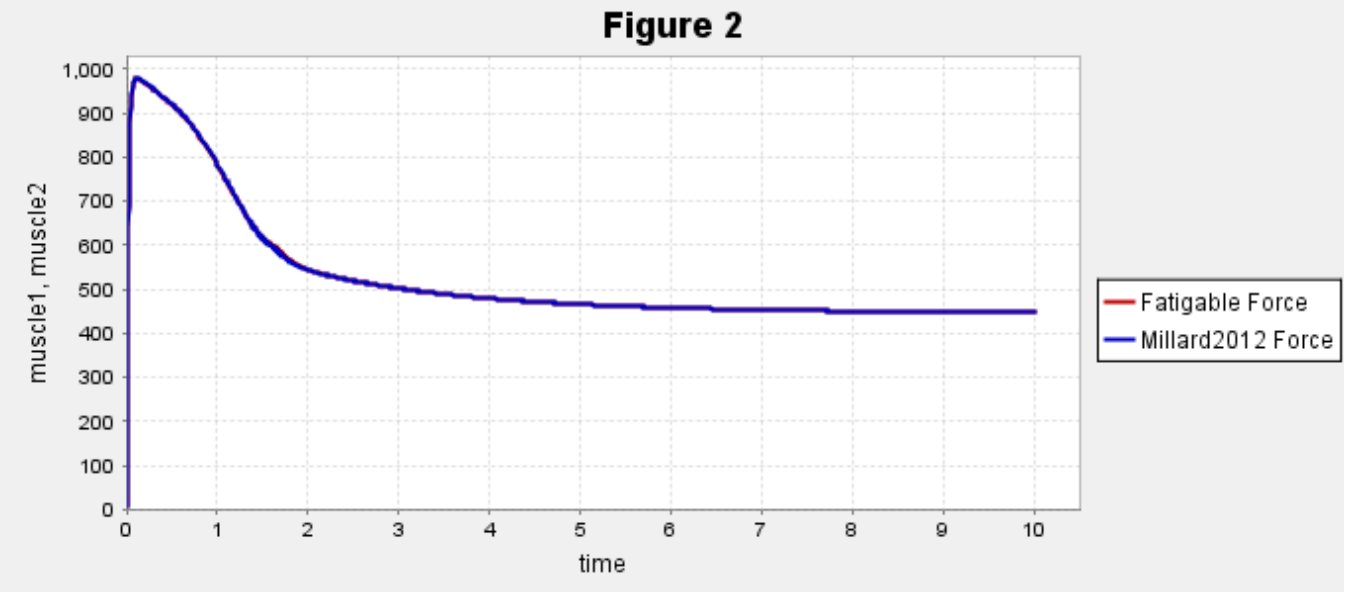
Rather than create a plug-in for this example, we will instead make a small modification to the model file so that it can be loaded into OpenSim. All we need to do is change the XML tag for the muscle named "muscle" from *FatigableMuscle* to *Millard2012EquilibriumMuscle*. This changes its type to a Millard2012EquilibriumMuscle—meaning that it will not include any fatigue effects—but this is not problematic for our example. The example program still uses the correct muscle types, but the OpenSim GUI will load the modified model. The muscles in this modified model have the same paths as in the original model, so it will visualize correctly in the GUI, but all quantities will be plotted from the simulation results. The modified model is *tugOfWar_fatigue_model_GUI.osim* in the MuscleExample folder.

So, we now want to load the model *tugOfWar_fatigue_model_GUI.osim* into OpenSim along with the motion file with the simulation results, *tugOfWar_fatigue_states_degrees.mot*. Play the motion to see the results of the simulation.

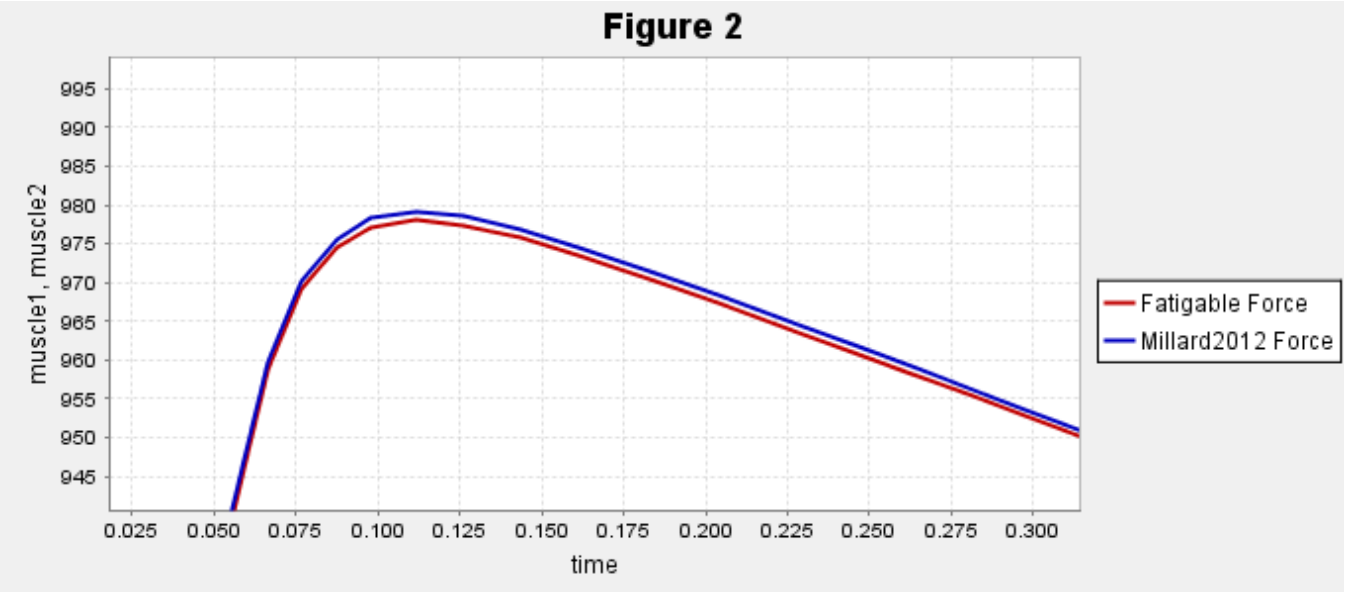
Plot the muscle-tendon length for both muscles using the plot tool. Look in the Muscle Analysis results folder to find the file tugOfWar_MuscleAnalysis_Length.sto file. Plot the length for both muscles.



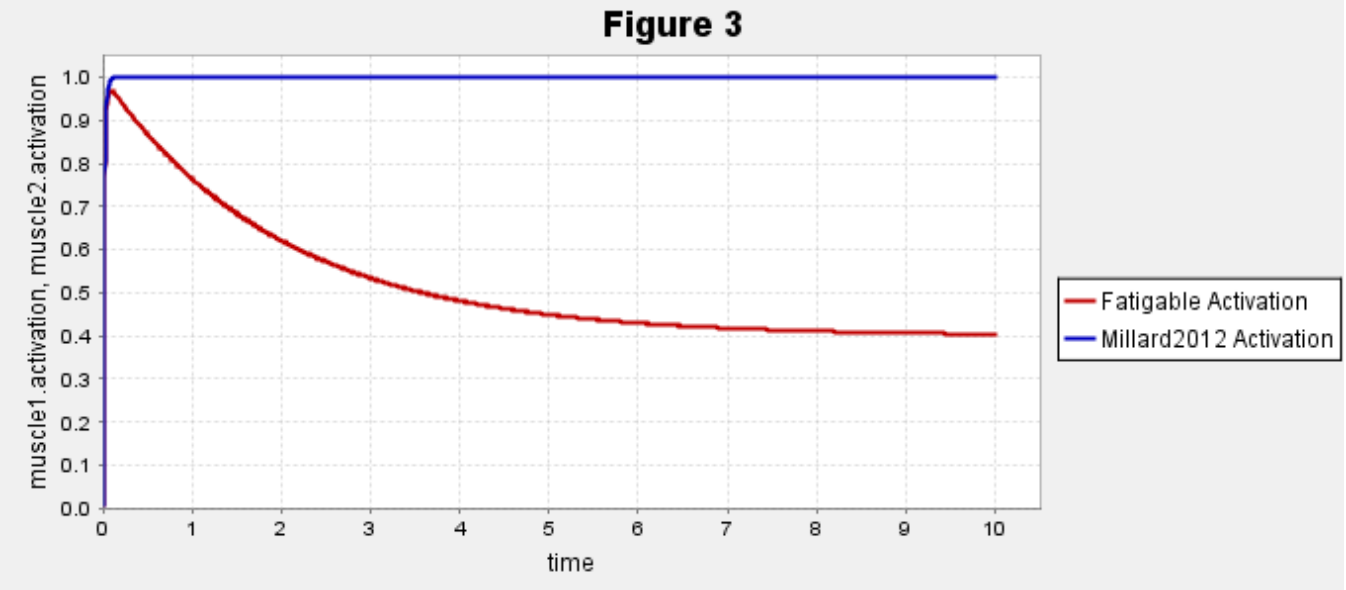
Both muscles start with zero activation at a length of 0.3 meters. Because the Fatigable muscle begins to fatigue, the original muscle "wins" the tug of war and stretches the Fatigable muscle. To see the effect of fatigue we can look at the muscle forces and activations. To plot the forces, use the plotter Tool to open the *tugOfWar_fatigue_forces.mot* file that was also generated by the fatigueMain program.



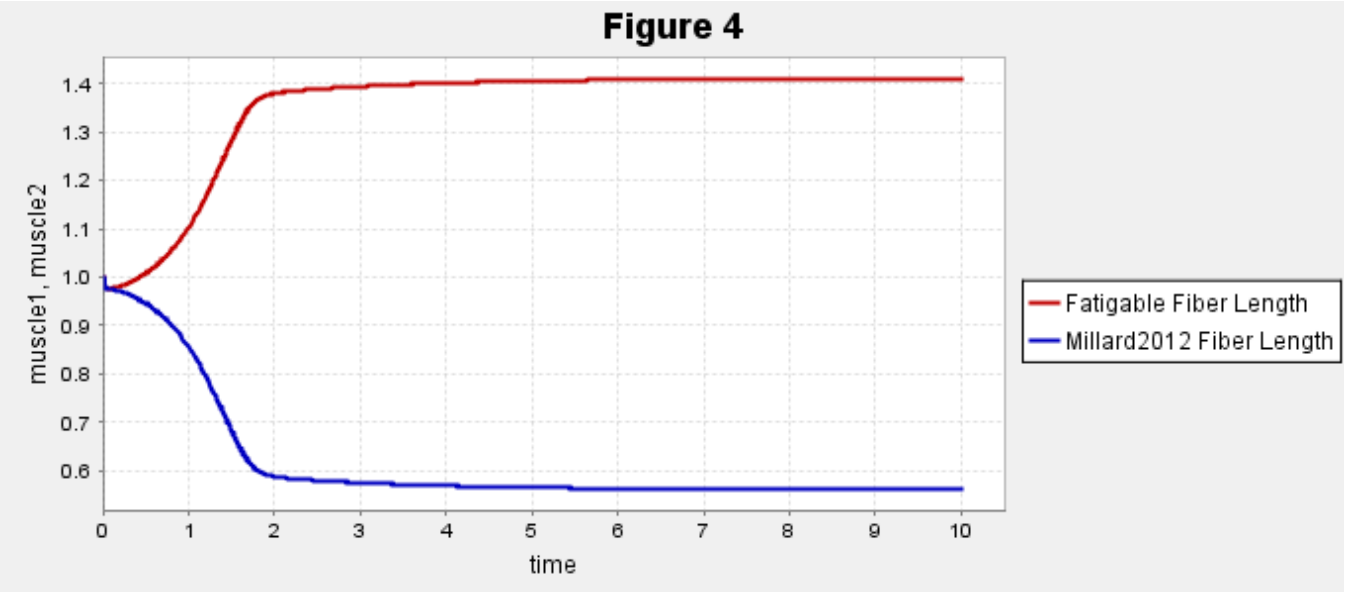
It appears that the muscle forces are identical, but they are not. Zoom into the peak between 0.0 and 0.5s of the plot.



The fatigable muscle forces are indeed smaller, which causes the fatigable muscle to lengthen and the original muscle to shorten. Now lets look at what role fatigue dynamics played in reducing the fatigable muscle's force by plotting the activations of the two muscles from the loaded states file.

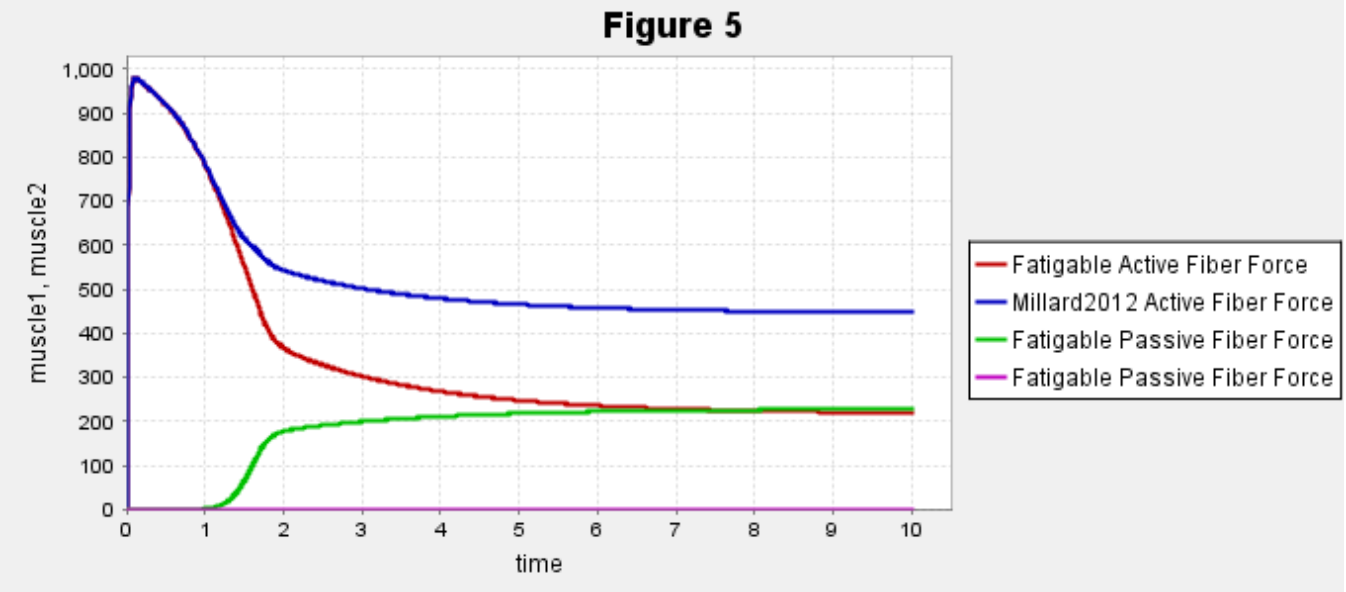


For the same constant and maximal excitation, the fatigable muscle's activation decays with time due to the increasing fraction of fatigued motor units. Given the large differences in activation, why are the muscle forces so similar? Remember, that muscle force is dependent not only on activation but on fiber length and velocity. Plot the muscles' normalized fiber lengths generated by the MuscleAnalysis in the *MuscleAnalysisResults* directory (*fatigue_MuscleAnalysis_NormalizedFiberLength.sto*) created by the mainFatigue program.



The muscles have identical properties (except for fatigue) and , but as the fatigable muscle lengthens it engages the fiber's passive element and has enhanced for generation capability due to lengthening velocity. In contrast the original muscle is shortening and does not have passive forces and the shortening force-velocity relationship diminishes the force generating capability of the muscle.

To see these effects more clearly, plot the active and passive muscle fiber forces for both muscles. The results are found in the *MuscleAnalysisResults* directory within the *fatigue_MuscleAnalysis_ActiveFiberForce.sto* and *fatigue_MuscleAnalysis_PassiveFiberForce.sto* files.



Next: SimTK Basics

Previous: Custom Muscle Model Part One

Home: Scripting and Development | Developer's Guide | Adding New Functionality