Creating an Optimization

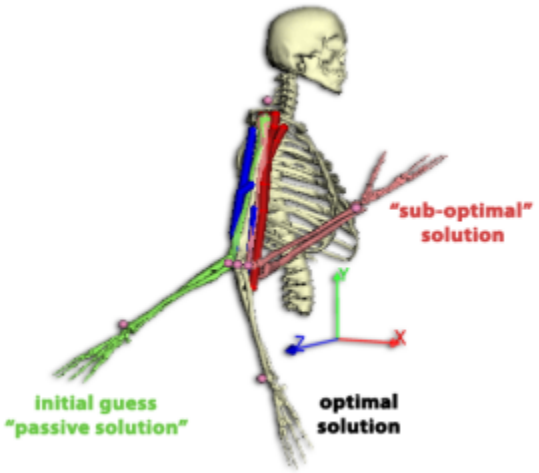
The steps to creating an optimization are:

Overview

In this section, we will write a main program to perform an optimization study using OpenSim. We will build it up in pieces, starting by programmatically loading an existing OpenSim model. The model will be a simple arm model named *Arm26.osim*, consisting of 2 degrees of freedom and 6 muscles. We will then define an optimization problem that finds a set of muscle controls to ***maximize the forward velocity*** of the forearm/hand segment mass center. The resulting source code and associated files for this example come with the OpenSim 2.0 distribution under the directory:

C:\Program Files\OpenSim 2.0\sdk\APIExamples\OptimizationExample_Arm26

As in Performing a Simulation, the following sections explain the steps to create your own main program. Additionally, we will be extending existing optimizer classes in the OpenSim API. For more information on the OptimizerSystem class, see the SimTKmath User's Guide available on the SimTK project site (https://simtk.org/home/simtkcore under the "Documents" tab).



Extending the *OptimizerSystem* class

Before we get into extending the class, we need to include the proper header files and define a few global variables.

```
#include <OpenSim/OpenSim.h>
using namespace OpenSim;
using namespace SimTK;
int stepCount = 0;

// Global variables to define integration time window, optimizer step
// count, the best solution.
double initialTime = 0.0;
double finalTime = 0.25;
double bestSoFar = Infinity;
```

In OpenSim, optimization problems are set up within an OptimizerSystem, which uses the SimTK-level algorithms to determine a solution. To set up our optimization problem, we need to create our own OptimizerSystem, called ExampleOptimizationSystem, by extending the existing base OptimizerSystem class.

```
class ExampleOptimizationSystem : public OptimizerSystem {
  public:
    /* Constructor class. Parameters accessed in objectiveFunc() class */
    ExampleOptimizationSystem(int numParameters, State& s, Model& aModel):
    numControls(numParameters), OptimizerSystem(numParameters), si(s), osimModel(aModel){}

  /* The objectiveFunc() class will go here. */
  private:
    int numControls;
    State& si;
    Model& osimModel;
};
```

Writing the main()

We can perform an optimization by creating our own main program that will invoke our OptimizerSystem.

```
int main()
{
  try {
    // Create a new OpenSim model
    Model osimModel("Arm26_Optimize.osim");

    /* The guts of your main() will go here */
  }

  catch (std::exception ex)
  {
    std::cout << ex.what() << std::endl;
    return 1;
  }

  // End of main() routine.
  return 0;
}
```

Initializing muscle states

We initialize the states for each muscle after setting the states.

```
// Define the initial muscle states
const Set<Muscle> &muscleSet = osimModel.getMuscles();
           for(int i=0; i< muscleSet.getSize(); i++ ){
                ActivationFiberLengthMuscle* mus = dynamic_cast<ActivationFiberLengthMuscle*>(&muscleSet[i]);
                if(mus){
                        mus->setDefaultActivation(0.5);
                        mus->setDefaultFiberLength(0.1);
                }
        }
// Initialize the system and get the state
State& si = osimModel.initSystem();

// Make sure the muscles states are in equilibrium
osimModel.equilibrateMuscles(si);
```

Define the optimizer

In SimTK and OpenSim, an Optimizer operates on an OptimizationSystem, which we will initialize as an ExampleOptimizerSystem . We then define the bounds for the parameters of the problem, the optimizer tolerance, and the numerical gradient flag before finally invoking the optimizer.

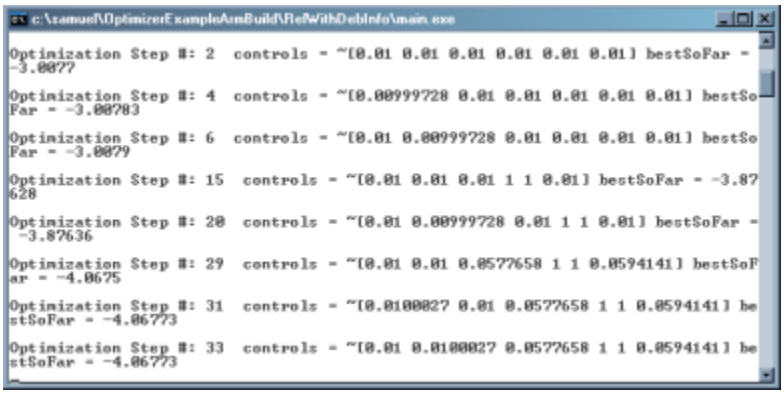
```
// The number of controls will equal the number of muscles in the model!
int numControls = osimModel.getNumControls();

// Initialize the optimizer system we've defined.
ExampleOptimizationSystem sys(numControls, si, osimModel);
Real f = NaN;

/* Define and set bounds for the parameter we will optimize */
Vector controls(numControls, 0.01);
Vector lower_bounds(numControls, 0.01);
Vector upper_bounds(numControls, 0.99);

sys.setParameterLimits( lower_bounds, upper_bounds );

// Create an optimizer. Pass in our OptimizerSystem
// and the name of the optimization algorithm.
Optimizer opt(sys, SimTK::LBFGSB);

// Specify settings for the optimizer
opt.setConvergenceTolerance(0.05);
opt.useNumericalGradient(true);
opt.setMaxIterations(1000);
opt.setLimitedMemoryHistory(500);
// Optimize it!
f = opt.optimize(controls);

catch(const std::exception& e) {
  std::cout << "Caught exception :" << std::endl;
  std::cout << e.what() << std::endl;
}
```

Writing the objective function

Within ExampleOptimizationSystem, we need to define our objective function as a public member of the class. This member function will take the parameters of the muscle controls that we want to vary and will return a real number about the performance we want to optimize (i.e., forward velocity of the hand) , which will then be minimized (*Note: To maximize a value, just multiply it by -1*). In this case, the parameters we want to vary are the muscle control values, and we will return a real number determined by our objective function (i.e., the forearm/hand mass center velocity). Additionally, if we had an analytical gradient or Jacobian function for our system, they could also be defined as member functions of the ExampleOptimizationSystem.

```
int objectiveFunc(const Vector &newControls, const bool new_coefficients, Real& f ) const {

// Make a copy of out initial states
State s = si;

// Update the control values
// newControls.dump("New Controls In:");
osimModel.updDefaultControls() = newControls;

// Create the integrator for the simulation.
RungeKuttaMersonIntegrator integrator(osimModel.getMultibodySystem());
integrator.setAccuracy(1.0e-6);

// Create a manager to run the simulation
Manager manager(osimModel, integrator);

// Integrate from initial time to final time
manager.setInitialTime(initialTime);
manager.setFinalTime(finalTime);
osimModel.getMultibodySystem().realize(s, Stage::Acceleration);
manager.integrate(s);

/* Calculate the scalar quantity for the optimizer to minimize
* In this case, we're maximizing forward velocity of the
* forearm/hand mass center so compute the velocity and
just multiply it by -1.*/
Vec3 massCenter;
Vec3 velocity;
osimModel.getBodySet().get("r_ulna_radius_hand").getMassCenter(massCenter);
osimModel.getMultibodySystem().realize(s, Stage::Velocity);
osimModel.getSimbodyEngine().getVelocity(s,osimModel.getBodySet().get("r_ulna_radius_hand"),
massCenter, velocity);
f = -velocity[0];
stepCount++;

// Store and print the results of a "random sample"
if( stepCount == 23 ){
        Storage statesDegrees(manager.getStateStorage());
        osimModel.updSimbodyEngine().convertRadiansToDegrees(statesDegrees);
        statesDegrees.print("Arm26_randomSample_states_degrees.sto");
}
// Store and print the  results of the first step.
else if( stepCount == 1){
        Storage statesDegrees(manager.getStateStorage());
        osimModel.updSimbodyEngine().convertRadiansToDegrees(statesDegrees);
        statesDegrees.print("Arm26_noActivation_states_degrees.sto");
}
/* Use an if statement to only store and print the results of an
* optimization step if it is better than a previous result.
*/
if( f < bestSoFar){
  Storage statesDegrees(manager.getStateStorage());
  osimModel.updSimbodyEngine().convertRadiansToDegrees(statesDegrees);
  statesDegrees.print("bestSoFar_states_degrees.sto");
  bestSoFar = f;
  std::cout << "\nOptimization Step #: " << stepCount << " controls = " << newControls << " bestSoFar = " << f
<< std::endl;
}

return(0);
}
```

Now you can build and run your main program, and then load the model and results into OpenSim to visualize the optimized control pattern and resulting kinematics.



Next: Creating a Customized Actuator

Previous: Creating a Controller Part Two

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