Creating an Optimization

OpenSim Developer’s Workshop | October 2009
Outline

1. Optimizer class in SimMath
   - What is an optimization problem?
   - What is the OptimizerSystem class?

2. Extending the OptimizerSystem class
   - Define a subclass for our optimization
   - What do I pass into the constructor?

3. Writing the main()
   - Set controls with initial guess
   - Set bounds on the controls
   - Creating the optimizer

4. Writing the Objective Function
   - Creating the integrator and manager
   - Finding the objective value
What is an optimizer?

The Optimizer class searches for points which are a minimum of an user-defined objective function.

- Define your objective function
- Start with initial guess
- Iterate to find points that reduce the objective
- Terminate when objective function stops decreasing

Notes:
- May not find global minimum, thus solution can be sensitive to your initial guess.
- To find the maximum, just multiply the objective function by -1.
The OptimizerSystem class

OptimizerSystem is an abstract class which has methods for computing:

- The **objective** function
- The **gradient** function
- The **constraints**
- The **constraint Jacobian**

- It provides an interface to multiple optimization algorithms.

- To use OptimizerSystem we need to define our own concrete subclass...
Extending the OptimizerSystem class

For each new optimization problem, we need to extend OptimizerSystem into our own subclass, like ExampleOptimizationSystem

```cpp
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()`...

In this section of the `main()`, we initialize the optimizer, set the bounds on the controls, and provide an initial guess for the controls.

```c
// Number of controls will equal the number of muscles in the model
int numControls = 6;

// 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 lower_bounds(numControls);
Vector upper_bounds(numControls);

for(int i=0;i<numControls;i++) {
    lower_bounds[i] = 0.01;
    upper_bounds[i] = 1.0;
}
sys.setParameterLimits( lower_bounds, upper_bounds );

// set the initial values (guesses) for the controls
Vector controls(numControls);
controls[0] = 0.01; controls[1] = 0.01;
controls[2] = 0.01; controls[3] = 0.01;
controls[4] = 0.01; controls[5] = 0.01;
```
Writing the `main()`...

In this section of the `main()`, we create an Optimizer, specify settings, and run the optimizer!

```cpp
try {
    // 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.useNumericalJacobian(true);

    // 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

Define **objectiveFunc** as a public member of **ExampleOptimizationSystem**

**Input:** muscle controls (i.e., parameters we want to vary)
**Output:** real number measuring performance (i.e, hand's forward velocity)

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

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

    // Access the controller set of the model and update the control values
    ((ControlSetController *)(osimModel.updControllerSet()[0]))
        ->updControlSet()->setControlValues(initialTime, &newControls[0]);
    ((ControlSetController *)(osimModel.updControllerSet()[0]))
        ->updControlSet()->setControlValues(finalTime, &newControls[0]);

    /* 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.
    */
    // FILL IN THE OBJECTIVE FUNCTION HERE!!!
    // f = ???;
```
Running the Optimization...