Adding a test case to TestCSALT

Steps to Take  
3/15/2017

This document describes the procedure to take when adding a new test case to the CSALT test program, TestCSALT. The instructions here assume that you have the test program configured and building using the Cmake configuration file (CmakeLists.txt) found in the top level folder of the lowthrust repository. The procedure described here uses the Bryson-Denham test case configuration to illustrate the procedure.

# Preliminaries

* Identify the problem to be solved
* Break out the math into CSALT compatible Point, Path, and driver pieces
* (Optional, but extremely useful) Code the problem and have it working in MATLAB
* Update (git pull) your copy of the CSALT repository

For the purposes of this document, all four of these preliminary steps are in place. The problem used here to illustrate the procedure is the Bryson-Denham problem, found in Chapter 3 of the PSOPT User Manual[1].

# File Setup

The CSALT repository includes a set of files that provide a starting point for implementing a test case, in the “CPlusPlus/test/TestOptCtrl/src/Shell” folder. These six files are paired C++ header and implementation files:

* **ShellDriver.hpp/.cpp**: The entry point for the test problem implementation.
* **ShellPathObject.hpp/.cpp**: The path function implementation.
* **ShellPointObject.hpp/.cpp**: The point function implementation.

Begin by copying the ShellDriver files into the “CplusPlus/test/TestOptCtrl/src” folder, and the ShellPathObject and ShellPointObject files into the “CplusPlus/test/TestOptCtrl/src/pointpath” folders. Then rename the copied files to a name appropriate for your problem. This procudure, using the Linux mv command for the renaming, is shown here:

$ cd CPlusPlus/test/TestOptCtrl/src/Shell/

$ cp ShellDriver.\* ..

$ cp Shell\*Object.\* ../pointpath/

$ cd ..

$ mv ShellDriver.cpp BrysonDenhamDriver.cpp

$ mv ShellDriver.hpp BrysonDenhamDriver.hpp

$ cd pointpath/

$ mv ShellPointObject.hpp BrysonDenhamPointObject.hpp

$ mv ShellPointObject.cpp BrysonDenhamPointObject.cpp

$ mv ShellPathObject.hpp BrysonDenhamPathObject.hpp

$ mv ShellPathObject.cpp BrysonDenhamPathObject.cpp

Finally, open each file and update the #include and references to reference the renamed header (\*.hpp) files, rather than the Shell files. I use a global search and replace for this, replacing “Shell” with the prefix (e.g. “BrysonDenham”) for my test case.

# File Edits

Once the core files for the new test problem are in place, they need to be edited to add problem specific details. The shell code can be compiled without any changes, so I start by updating the CmakeLists.txt file.

## The Cmake Configuration File

Start by editing CmakeLists.txt in the root folder of the repository. Add the three new implementation files to the list of files in the TEST\_SRCS section. For this addition, I made the changes shown in bold here:

SET(TEST\_SRCS

CPlusPlus/test/TestOptCtrl/src/TestOptCtrl.cpp

CPlusPlus/test/TestOptCtrl/src/BrachistochroneDriver.cpp

...

CPlusPlus/test/TestOptCtrl/src/SchwartzDriver.cpp

**CplusPlus/test/TestOptCtrl/src/BrysonDenhamDriver.cpp**

...

CplusPlus/test/HelperClasses/GoddardRocketPathObject.cpp

**CPlusPlus/test/TestOptCtrl/src/pointpath/BrysonDenhamPointObject.cpp**

**CPlusPlus/test/TestOptCtrl/src/pointpath/BrysonDenhamPathObject.cpp**)

Once your additions have been made, build the application. The code should build and include your new objects.

Next we need to add the test problem code to each file.

## The Driver, Point Function, and Path Function

The shell code contains comments indicating where you should make adjustments and add implementation code. Update each of the implementation (\*.cpp) files to match your problem. You’ll need to add code in the following methods:

**\*Driver.cpp**: Add code in the Run() method, at the marked places. The code in the Driver method is pretty well fleshed out for a basic problem, and includes comments indicating where changes may be needed.

\***PointObject.cpp** and \***PathObject.cpp**: Add problem specific code for the EvaluateFunctions() and EvaluateJaconians() methods. A start on that code, using common elements, is in place.

TO DO:

* Add notes about working with Rvector and Rmatrix code
* Add try/catch around the call to Optimize

## Application Entry Point

Finally, the new test driver needs to be added to the main() method in TestOptCtrl.cpp, the file containing the application driver.

# Summary

# References

[1]Victor M. Becerra, PSOPT Optimal Control Solver User Manual, Release 3, July 2011.